Chapter 1

Overview

Clara AGX SDK is a collection of documentation, drivers, and reference applications that has been designed by NVIDIA to help developers build end-to-end streaming workflows for medical imaging applications. From training neural networks using the Transfer Learning Toolkit, to building a streaming inference application using DeepStream, to deployment on a Jetson AGX embedded device; the Clara AGX SDK takes an incremental approach to introduce concepts and NVIDIA technology in a way that can be applied irrespective of previous development experience in the field of medical imaging.

While the ultimate goal of the Clara AGX SDK is to deploy applications on a Jetson AGX embedded device, the SDK includes x86 Linux versions of the sample applications in order to provide a cross-platform development environment that can be used even without access to Jetson AGX device. In any case, an NVIDIA GPU will be required in order to utilize the Transfer Learning Toolkit, DeepStream, TensorRT, CUDA, and any other NVIDIA software components that are used by the Clara AGX SDK (see Prerequisites).
Chapter 2

Clara AGX SDK Components

The Clara AGX SDK has been divided into a number of separate installation packages, as follows.

2.1 Clara AGX TLT

The Clara AGX TLT package provides a sample annotated data set as well as various scripts that are used to train a neural network and output a TensorRT runtime inference engine that can be used with the sample applications.

The sample data set consists of 300 images of the Learning Resources Anatomy Model that has been annotated with stomach and intestines regions. This data set can be used as-is with the DeepStream and Endoscopy Sample Apps, which include prebuilt TensorRT engines generated using this dataset. Alternatively, a custom data set and corresponding inference model may be annotated and generated by the user using utility scripts provided in the AGX TLT package and following the Model Creation section in this documentation.
2.2 Sample Applications

Building a powerful end-to-end inference application using DeepStream and TensorRT requires the use of a number of different software libraries that can be overwhelming if starting with no knowledge of these libraries. To help alleviate this problem, the AGX SDK includes three different sample applications that build on top of one another to incrementally introduce new components and features.

All of the sample applications included with the AGX SDK are written in C and/or C++.

2.2.1 GStreamer Sample

The GStreamer sample application is the most basic sample which demonstrates the use of the GStreamer framework in order to simply capture image frames from a camera and then render the captured video stream onto the display. This sample does not make use of DeepStream, but since DeepStream functionality is exposed to applications via GStreamer framework plugins, a basic understanding
of GStreamer is essential to write DeepStream applications.

The GStreamer sample is available for Jetson AGX, Linux, and Windows platforms, and are located in the `GstreamerJetsonSample`, `GStreamerLinuxHostSample`, and `GStreamerWindowsHostSample` directories, respectively.

### 2.2.2 DeepStream Sample

The DeepStream sample application builds on top of the GStreamer sample by including the DeepStream inference plugin into the pipeline such that the stomach and intestines regions of the sample model are identified by bounding boxes rendered on top of the video stream. Alternatively, the sample model can be replaced with a user-provided model (by changing a configuration file used by the sample) to identify regions specified by the custom model.

This sample also expands on the GStreamer sample by providing multi-camera support, as well as the ability to send the captured and annotated video across a UDP H.264 video stream such that it can be viewed by another remote system.

The DeepStream sample is available for Jetson AGX and Linux, and are located in the `DeepstreamJetsonAGXSample` and `DeepstreamLinuxHostSample` directories, respectively.

![DeepStream Sample Application](image)

**Multi-Stream DeepStream Sample Application**

### 2.2.3 Endoscopy Sample

The Endoscopy sample builds on top of the DeepStream sample by adding runtime controls that are specific to the model and inferencing that is performed by DeepStream. These controls include the ability to switch between multiple inference models, and to enable/disable the annotation of the inference regions that are provided by the models. The UI is also modified to provide a more streamlined experience that might be desirable for an application that is deployed into medical imaging environments.

The Endoscopy sample is available only for Jetson AGX, and is located in the `EndoscopyJetsonAGXSample` directory.
Many applications that are used today in medical environments have been built on and depend on the use of Windows computers. Replacing these applications with ones that support Linux is often prohibitively expensive, and so the Jetson AGX platform offers a high-bandwidth solution for integrating Jetson AGX embedded devices with existing Windows x86 host systems.

By connecting an x86 Windows host to a Jetson AGX target device via a PCIe ribbon cable, the Jetson AGX device will be able to act as a PCIe endpoint device on the Windows host computer, appearing as if it is a network device. This connection is then used to setup a virtual network between the Windows host and the Jetson AGX target, providing a network connection between the two that can leverage the full bandwidth provided by PCIe.

Enabling this PCIe connection requires some additional hardware (see Prerequisites), as well as the two additional software components included with the AGX SDK:

- **Windows PCIe Virtual Ethernet Drivers**. These must be installed onto the Windows host system, and are located in the `Windows10` directory of the AGX SDK.

- **Jetson AGX PCIe Endpoint Scripts**. These scripts are installed and run on the Jetson AGX device, and are include with the Clara AGX Tools package in the `JetsonAGX` directory of the AGX SDK.
PCle Connection Between Jetson AGX and Windows Host
Chapter 3

Prerequisites

The Clara AGX SDK has been optimized to work with either the Jetson AGX Xavier or Clara AGX Development Kits. The following additional components are required to make use of various feature or samples included with the AGX SDK.

3.1 JetPack Software

The following components must be installed onto the AGX device using the NVIDIA SDK Manager.

- JetPack 4.5.0
- DeepStream 5.1

3.2 CSI Camera Module

A CSI camera module is used for the execution of the DeepStream and endoscopy sample applications on the target AGX device. While many CSI camera modules should work with the samples, the following have been tested and are recommended.

- Clara AGX HDMI Input Module (Included with Clara AGX Development Kit)
- Leopard IMX-274
- Leopard IMX-390 with the GMSL kit

Note: The Leopard IMX-390 requires a custom kernel, which can be built using the L4T kernel build instructions

Important: Camera modules with CSI connectors are not supported in dGPU mode on Clara AGX Development kit.
3.3 USB Camera

The DeepStream Sample application may also be run on either the NVIDIA Xavier or the Linux x86 host machine using a USB camera. Many USB cameras should be compatible, but the following have been tested with the AGX SDK.

- Logitech C270
- Logitech C310

3.4 Linux x86 Host

The Clara AGX SDK requires a Linux x86 PC with the following hardware and software.

- An NVIDIA Quadro® or Tesla® GPU that supports CUDA 11.1.
  - If training a new model from a data set, the GPU must have at least 8GB of GDDR memory.
- At least 8GB of RAM
- Ubuntu 18.04
- NVIDIA Container Runtime

The following software is also required and will be installed by SDK Manager as part of the Clara AGX SDK installation.

- CUDA 11.1
- TensorRT 7.2
- DeepStream 5.1

3.5 PCIe Virtual Ethernet

A high-bandwidth virtual ethernet connection can be setup using a PCIe connection between an AGX device and another AGX or x86 based host device using the following connectors.

- PCIe x8 Card Edge to Card Edge Ribbon Cable
- E3317-A01 PCIe RX/TX Swap Board

If using a Windows x86 host for the PCIe virtual ethernet connection, the Windows version installed on this PC must be Windows 10 version 1909.
Chapter 4

SDK Installation

The Clara AGX SDK can be installed either manually, or automated via the NVIDIA SDK Manager.

4.1 Manual Installation

The manual installation package of the Clara AGX SDK consists of a single zip archive that contains various installation packages for the AGX, Linux x86, an Windows 10 platforms. It also contains this documentation and an archive containing the sample data set and scripts that were used to train the models that are used with the sample applications.

Installing this package consists of extracting and/or installing the packages onto the platform for which the packages target (based on the package directory and file name).

For example, the following command can be used to install the DeepStream sample application package:

```
$ sudo apt install ./clara-agx-deepstream-sample-src_*_arm64.deb
```

4.2 SDK Manager Installation

Starting with Clara AGX SDK version 2.0, the NVIDIA SDK Manager can be used to automate the installation process for the Clara AGX SDK. If using the SDK Manager for the installation, please refer to the information button within the SDK Manager installation page to inspect where various components were installed or extracted to.
Chapter 5

Storage Setup (m2 SSD)

The Clara AGX Developer Kit includes a pre-installed 250GB m2 solid-state drive (SSD), but this drive is not partitioned or mounted by default. This page outlines the steps that should followed after the initial SDK installation in order to partition and format the drive for use.

**Note:** The following steps assume that the m2 drive is identified by the Clara AGX Developer Kit as /dev/sda. This is the case if no additional drives have been attached, but if other drives have been attached (such as USB drives) then the disk identifier may change. This can be verified by looking at the symlink to the drive that is created for the m2 hardware address on the system. If the symlink below shows something other than ../../sda, replace all instances of sda in the instruction below with the identifier that is being used by your system:

```
$ ls -l /dev/disk/by-path/platform-14100000.pcie-pci-0001:01:00.0-ata-1
lrwxrwxrwx 1 root root 9 Jan 28 12:24 /dev/disk/by-path/platform-14100000.pcie-pci-0001:01:00.0-ata-1 -> ../../sda
```

5.1 Create Partition

1. **Launch fdisk utility:**

   ```
   $ sudo fdisk /dev/sda
   ```

2. **Create a new primary partition.** Use the command ‘n’, then accept the defaults (press enter) for the next 4 questions to create a single partition that uses the entire drive:

   ```
   Command (m for help): n
   Partition type
   p  primary (0 primary, 0 extended, 4 free)
   e  extended (container for logical partitions)
   Select (default p):
   Using default response p.
   Partition number (1-4, default 1):
   First sector (2048-488397167, default 2048):
   Last sector, +sectors or +size(K,M,G,T,P) (2048-488397167, default → 488397167):
   ```
3. **Write the new partition table and exit.** Use the `w` command:

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

4. **Initialize ext4 filesystem on the new partition:**

```
$ sudo mkfs -t ext4 /dev/sda1
mke2fs 1.44.1 (24-Mar-2018)
Creating filesystem with 486400 4k blocks and 121680 inodes
Filesystem UUID: c3817b9c-eeaa-4423-ad5b-d6bae8aa44ea
Superblock backups stored on blocks:
  32768, 98304, 163840, 229376, 294912
Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done
```

5.2 **Mount Partition**

1. **Create a directory for the mount point.** These instructions will use the path `/media/m2`, but any path may be used if preferred.

```
$ sudo mkdir /media/m2
```

2. **Determine the UUID of the new partition.** The UUID will be displayed as a symlink to the `/dev/sda1` partition within the `/dev/disk/by-uuid` directory. For example, the following output shows that the UUID of the `/dev/sda1` partition is `4b2bb292-a4d8-4b7e-a8cc-bb799dfeb925`:

```
$ ls -l /dev/disk/by-uuid/ | grep sda1
lrwxrwxrwx 1 root root 10 Jan 28 10:05 4b2bb292-a4d8-4b7e-a8cc-bb799dfeb925 -> ../../sda1
```

3. **Add the fstab entry.** Using the mount path and the UUID from the previous steps, add the following line to the end of `/etc/fstab`:

```
UUID=4b2bb292-a4d8-4b7e-a8cc-bb799dfeb925 /media/m2 ext4 defaults 0 2
```

4. **Mount the partition.** The `/etc/fstab` entry above will mount the partition automatically at boot time. To instead mount the partition immediately without rebooting, use the `mount` command (and `df` to verify the mount):

```
$ sudo mount -a
$ df -h /dev/sda1
Filesystem Size Used Avail Use% Mounted on
/dev/sda1 229G 5.6M 229G 0% /media/m2
```
Note: It the Clara AGX Developer Kit is reflashed with a new JetPack image, the partition table of the m2 drive will not be modified and the contents of the partition will be retained. In this case the Create Partition steps can be skipped, however the Mount Partition steps should be followed again in order to remount the partition.

Also note that any state, binaries, or docker images that persist on the m2 drive after flashing the system may be made incompatible with new libraries or components that are flashed onto the system. It may be required to recompile or rebuild these persistent objects to restore runtime compatibility with the system.

### 5.3 Move Docker Storage to m2 Partition

A complete installation of the Clara SDK leaves only about 10GB of storage remaining in the root 32GB filesystem (/dev/mmcblk0p1). When using Docker it is often the case that individual images will be many GB in size, and so this remaining disk space is generally insufficient for the storage needs of Docker images. For this reason it is highly recommended that the Docker daemon data directory be moved to a location on the new m2 partition. This can be done with the following steps:

1. **Create a new Docker data directory.** This is where Docker will store all of its data including build cache and container images. These instructions use the path /media/m2/docker-data, but another directory name can be used if preferred:

   ```
   $ sudo mkdir /media/m2/docker-data
   ```

2. **Configure the Docker Daemon.** Add the following data-root configuration option to the /etc/docker/daemon.json file, pointing to the new data directory created above. Create the /etc/docker/daemon.json file if it does not already exist.

   ```
   {
   "data-root": "/media/m2/docker-data"
   }
   ```

   **Note:** If existing configuration already exists in the daemon.json file, make sure to add a comma to the preceeding line before the data-root configuration, e.g.

   ```
   {
   "default-runtime": "nvidia",
   "data-root": "/media/m2/docker-data"
   }
   ```

3. **Restart the Docker Daemon.**

   ```
   $ sudo systemctl daemon-reload
   $ sudo systemctl restart docker
   ```
Chapter 6

Camera Setup and Verification

This page outlines how to verify the camera setup used with the Jetson target and x86 host devices to ensure they are functioning properly for use with the sample applications.

6.1 Jetson Camera Setup

1. Connect a CSI or USB camera module to your Clara AGX Development Kit or Jetson AGX module.
   - Please see the Prerequisites section for the list of compatible camera devices.

2. Run one of the following commands to prove that capture is working correctly. In all cases, a window should be rendered onto the display with a live stream from the capture.
   - CSI Camera

```
$ gst-launch-1.0 nvarguscamerasrc ! 'video/x-raw(memory:NVMM),
  width=(int)1920, height=(int)1080, format=(string)NV12,' framerate=(fraction)30/1' ! nvoverlaysink -e
```

**Note:** nvarguscamerasrc plugin is not available in dGPU mode on Clara AGX Development Kit.

- CSI HDMI Input Board

```
$ gst-launch-1.0 v4l2src io-mode=mmap device=/dev/video0 ! 'video/x-raw, format=(string)BGRA, width=(int)1920, height=(int)1080, framerate=(fraction)60/1' ! \ncapssetter join=true caps='video/x-raw, format=(string)RGBA' ! \nnvvidconv ! 'video/x-raw(memory:NVMM)' ! nv3dsink sync=false
```

- USB Camera

```
$ gst-launch-1.0 v4l2src device=/dev/video0 ! xvimagesink
```

**Note:** Replace /dev/video0 in the commands above with the path that corresponds to the device being tested.
3. Press Ctrl+C to exit the camera capture.

### 6.2 Linux x86 Camera Setup

1. Connect a USB camera to your Linux x86 host.
2. Run the following command to prove that capture is working correctly. A window should be rendered onto the display with a live stream from the camera.

   ```shell
   $ gst-launch-1.0 v4l2src device=/dev/video1 ! xvimagesink
   ```

   **Note:** Replace `/dev/video0` in the command above with the path that corresponds to the USB camera device being tested.

3. Press Ctrl+C to exit the camera capture.

### 6.3 Streaming Camera Setup Check

The DeepStream sample application offers the ability to stream the camera image from the target device to a remote host system using a UDP data stream. These steps can be followed to ensure that both the target and host systems are setup correctly for this streaming to work.

#### 6.3.1 Jetson

1. Ensure the camera capture is working as described in the *Jetson Camera Setup* section, above.
2. Run the following command to start streaming the camera capture, replacing `<IP_ADDRESS>` with the IP address of the host system that will be receiving the stream.

   ```shell
   $ gst-launch-1.0 nvarguscamerasrc ! 'video/x-raw(memory:NVMM),
   → width=(int)1920, height=(int)1080, format=(string)NV12, → framerate=(fraction)30/1' ! 
   → omxh264enc control-rate=2 bitrate=4000000 ! 'video/x-h264, stream-
   → format=(string)byte-stream' ! 
   → h264parse ! rtph264pay mtu=1400 ! udpsink host=<IP_ADDRESS>
   → port=5000 sync=false async=false
   ```

#### 6.3.2 Linux Host

Use the following command to start streaming the camera capture on a Linux x86 host (close with Ctrl+C):

```shell
$ gst-launch-1.0 udpsrc port=5000 ! application/x-rtp,encoding-name=H264, → payload=96 ! rtph264depay ! queue ! avdec_h264 ! autovideosink sync=false
```
6.3.3 Windows Host

1. Download GStreamer for Windows.
2. Install GStreamer on Windows:
   1. Run the `.msi` installer.
   2. At the Choose Setup Type screen, select Custom.
   3. Locate the “GStreamer 1.0 livab wrapper” item. Use the dropdown to select Will be installed on local hard drive.
   4. Click Next and install GStreamer.
3. Add GStreamer to the Windows path environment variable:
   1. Press Win+S to open the search window and type “env” into the search field. Select Edit the system environment variables.
   2. Click the Environment Variables... button.
   3. Under System Variables, select the Path variable and click Edit....
   4. In the Edit environment variable window, click New. Enter “%GSTREAMER_ROOT_X86%bin” into the field.
   5. Use the OK buttons to close the environment/properties windows you opened.
4. Use the following command to start streaming the camera capture:
   ```bash
   $ gst-launch-1.0 udpsrc port=5000 ! application/x-rtp,encoding-name=H264, payload=96 ! rtph264depay ! queue ! avdec_h264 ! autovideosink sync=false
   ```
   5. When asked, select Allow Access for “gst-launch-1.0.exe”.

You should see the camera-capture stream in a window on your desktop. You can exit the window by clicking the “X” in the upper-right corner of the window.
Chapter 7

Windows PCIe Virtual Ethernet

Perform the following steps to setup the PCIe virtual ethernet connection between an AGX and a Windows 10 host.

7.1 Setup with PCI/vETH

**Important:** Ensure your Windows PC is running Windows 10 version 1909.

1. Connect the RX/TX swap board to the host machine. The swap board is directional and must be plugged directly into a PCIe slot on the host machine.

2. Connect the PCIe jumper cable from the RX/TX swap board to the Jetson module.
The E3317-A01 RX/TX swap board and 8x PCIe jumper cable
Example setup with swap board, jumper cable, Jetson AGX, and camera

3. Configure and flash the Jetson AGX as an endpoint device.

1. Install the clara-agx-tools_<version>_arm64.deb package on the Jetson AGX:

   
   ```
   $ sudo dpkg -i ./clara-agx-tools_<version>_arm64.deb
   ```

2. Turn on PCIe Endpoint mode:

   ```
   $ sudo ./jetson-edit-odmdata.py --pcie-ep
   ```

3. Reboot the Jetson AGX board.

   **Note:** If you don’t have your own device image, skip ahead to step 5.
4. If you have your own device image, you can change the BCT for the board prior to flashing:

   (a) In the image output directory, open the `Linux_for_Tegra/p2972-0000.conf.common` file and change the following line:

   ```
   ODMDATA=0x9190000
   ```

   to

   ```
   ODMDATA=0x9191000
   ```

   (b) Flash the Jetson AGX with the following command. The Jetson AGX will reboot after completing this action.

   ```
   $ sudo ./flash.sh p2972-0000-devkit mmcblk0p1
   ```

   **Important:** The above steps only need to be performed once, but subsequent steps must be repeated every time a reboot occurs on the Jetson AGX module or Windows machine.

4. To start the PCIe endpoint drivers on the Jetson AGX module, run the `enable_pcie_ep.sh` script. Alternatively, you can execute the following commands:

   ```
   $ cd /sys/kernel/config/pci_ep
   $ sudo mkdir functions/pci_epf_tvnet/func
   $ sudo ln -s functions/pci_epf_tvnet/func controllers/141a0000.pcie_ep
   $ echo 1 | sudo tee controllers/141a0000.pcie_ep/start
   ```

5. Boot up the Windows host machine. Once it has finished booting, you should see the following output from the Jetson AGX kernel log, confirming that the host machine sees the Jetson AGX PCIe device:

   ```
   $ dmesg
   ...
   [ 1006.326913] tegra-pcie-dw 141a0000.pcie_ep: EP init done
   ```

6. Boot into Windows with Driver Signature Verification disabled. This will allow you to install unsigned Jetson AGX drivers. You can boot into this mode as a one time activity or set it as a persistent configuration:

   (a) **One-time boot**

   (a) Press *Win+S* to open the search window and type “startup options” into the search field.

   (b) Select *Change advanced startup options*, which should be the top search result.

   (c) Click *Restart Now* under *Advanced Startup*.

   (d) When the restart screen loads, go to *Troubleshooting > Advanced Options > Startup Settings*. Click the *Restart* button to restart the machine.

   (e) When the machine displays a boot menu, press 7 to boot with *Disable driver signature enforcement*.

   (a) **Persistent unsigned driver support**

   (a) Press *Win+X* to open the Windows Power User Task menu. Select *Windows PowerShell (Admin)*.

7.1. Setup with PCI/vETH
(b) Execute the following on the command line:

```
> Bcdedit.exe -set TESTSIGNING ON
```

(c) Reboot the machine. After logging in to Windows, you should see a “Test Mode” string on the bottom-right of the desktop.

2. Install Windows drivers for Jetson AGX using the provided JetsonAGXSetup.msi installer package (this also requires setup.exe).

   **Note:** These drivers only support Windows 10 version 1909.

(a) Open the Device Manager in Windows.

(b) Expand the **Other devices** section.

(c) Right-click the **Network Controller** device and select properties.

(d) Select **Update Driver**, then **Browse my computer for driver software**.

(e) Select the directory where the Jetson AGX drivers are located. The Jetson AGX installer places them in `C:\Program Files\NVIDIA Corporation\NVIDIA Jetson AGX Driver\Driver` by default. You can find a link to this directory in your Start Menu under **NVIDIA Corporation > Browse Jetson AGX Drivers**.

(f) Approve the installation of the unsigned driver by selecting **Install this driver software anyway**.

After installation, the **Jetson AGX Development Kit** device will appear in Device Manager under **Network adapters**.

![Device Manager](image)
3. Bring up the virtual Ethernet connection on both the host and the Jetson AGX endpoint.

Note: Both endpoint and host maintain their local link state independently of each other; the link is only considered connected or active when both endpoint links are up. The interfaces should be brought up with static IPs that are visible to one another—in this example, the host uses 172.22.250.2 and the endpoint uses 172.22.250.1.

1. Windows Host: Bring up the Virtual Network using the Windows tools with command line and the AGX Control App.

   (a) Windows tools and command line
   i. Open the Control Panel.
   ii. Select View network status and tasks under Network and Internet.
   iii. Select Change adapter settings on the left sidebar.
   iv. Right-click the Jetson AGX Development Kit connection and select Properties.
   vi. Set the IP address to “172.22.250.2” and the Subnet mask to “255.255.255.0”.
   vii. Click OK and close all Control Panel windows.
   ix. Execute the following to bring up the link:

       > AGXControl.exe up

       Tip: The link can be brought down with the down argument.

   (b) AGX Control App
   (a) Ensure the Visual Studio Redistributable VC_redist.x64.exe is installed. This executable is delivered as part of the SDK and allows you to use the AGX Control App.
   (b) Go to the Start Menu and locate the NVIDIA Corporation folder.
   (c) Right-click AGX Control App and select Run as administrator.
   (d) Click the UP button. This will enable the virtual Ethernet link as if you just plugged in the cable.

7.1. Setup with PCI/vETH
(e) Configure the **IP Address**, **Default Gateway**, and **Subnet Mask** if they show only “…” . Click **Apply** to configure the new IP parameters.

(f) If the UP button is grayed out and the app shows **Not Found**, click the **Retest** button in the top-right. If that doesn’t work, your device did not come up properly: Investigate the Device Manager first.

(g) On your Windows desktop, right-click the Ethernet network symbol in the Tool-
bars section on the bottom right and select **Open Network & Internet Settings**. Agree to allow all connections to this network. Make sure the network is enabled.

2. **Jetson AGX Endpoint**: Configure the network interface using Network Manager.
   
   (a) **Using the command line**
   
   ```
   $ sudo ifconfig eth1 up 172.22.250.1
   ```

   (b) **Using the Network Manager**

   (a) Click the icon of the two opposing arrows in the top-right.

   (b) Select **Edit Connections**... from the dropdown menu.

   (c) Select the Ethernet connection that is not labeled **Wired connection 1** (e.g. select **Wired connection 3** instead) and click the gear icon on the bottom-left of the window to edit the connection.
(d) In the Editing `<connection_name>` window, go to the IPv4 Settings tab and do the following:

i. Set the Method to Manual.

ii. Click the Add button. Enter a valid IP address, netmask, and gateway. Then hit Enter to confirm.

iii. Click the Save button on the lower right to save the settings.

(e) Confirm that the new settings have taken effect by typing `ifconfig` at a Terminal window (the settings are listed under “eth1”). It may take a little while for the new settings to propagate.

**Tip:** If the Network Manager is not installed, you can use the command
line instead (e.g. $ sudo ifconfig eth1 up 172.22.250.1). We only recommend using the command line if the Network Manager is not installed since the Network Manager may overwrite the command-line setting at runtime.

9. Verify the network setup on the Windows host by navigating to Control Panel > Network and Internet > View Network Status and Tasks > Change Adapter Settings. The Windows > connection to the Jetson AGX module will show as “unplugged” until both the host and endpoint links are up, at which point it will become “connected” and the Ethernet connection will be made available. You can verify the connection using any type of network connection, such as pinging the endpoint or opening an ssh session.

Virtual Ethernet shows as “unplugged” because both links are not up yet.

Virtual Ethernet is connected because both links are up.
7.2 Windows Setup Troubleshooting

7.2.1 Networking/Firewall

The firewall configuration on Windows can cause connection issues: for example, by default Windows does not allow ping messages through. Follow these steps to modify the Windows Firewall:

1. Press Win+S to open the search window and type “firewall” into the search field. Select Windows Defender Firewall from the list.
2. Select Advanced settings on the left sidebar.
4. Select the File and Printer Sharing (Echo Request - ICMPv4-In) rule. There are two of these rules; either will work.
5. From the Actions menu on the right, select Properties.
6. Select the Advanced tab in the Properties window.
7. Ensure all three boxes under Profiles are checked: Domain, Private, and Public.
8. Click OK to save the changes.
9. Select the File and Printer Sharing (Echo Request - ICMPv4-In) rule again and click Enable Rule in the Actions menu.

7.2.2 Power Management

We recommend turning off power management features for Windows 10 to avoid losing PCIe connectivity between Windows and the Jetson AGX endpoint. If the Windows 10 machine goes to sleep, the Jetson AGX will reset, and you will need to reinitialize both devices.

Press Win+S to open the search window and type “power”. Select Power & sleep settings from the list. Under Sleep, set the interval to Never (you may need to scroll down to the bottom of the options list).
Chapter 8

Switching Between iGPU and dGPU

The Clara AGX Developer Kit can use either the Xavier AGX module GPU (iGPU, integrated GPU) or the RTX6000 add-in card GPU (dGPU, discrete GPU). Only one GPU can be used at a time, with the iGPU being the default after flashing the Clara AGX SDK with JetPack. Switching between the iGPU and dGPU is performed using the `nvgpuswitch.py` script contained in `/usr/local/bin`.

**Warning:** The Clara AGX SDK components, such as the DeepStream sample applications and SUPRA, will be uninstalled during the process of switching GPUs. If you have made any changes to the source code installed by these packages (i.e., in `/opt/nvidia/clara-agx-sdk` or `/opt/supra/src`), make sure to back up these changes before switching GPUs. See Reinstalling Clara AGX SDK Packages for more details.

To view the currently installed drivers and their version, use the query command:

```
$ nvgpuswitch.py query
iGPU (nvidia-l4t-cuda, 32.5.0-20201012161040)
```

To install the dGPU drivers, use the install command with the dGPU parameter (note that sudo must be used to install drivers):

```
$ sudo nvgpuswitch.py install dGPU
```

The install command will begin by printing out the list of commands that will be executed as part of the driver install, then will continue to execute those commands. This aids with debugging if any of the commands fail to execute for any reason. The following debug arguments may also be provided with the install command:

- `-d` Does a dry run, showing the commands that would be executed by the install but does not execute them.
- `-v` Enable verbose output (used with `-d` to describe each of the commands that would be run).
- `-i` Run commands interactively (asks before running each command).
- `-l [LOG]` Writes a log of the install to the file `LOG`.

The dGPU driver install may be verified once again using the query command:
$ nvgpuswitch.py query
dGPU (cuda-drivers, 455.32.00-1)

After the dGPU drivers have been installed, rebooting the system will complete the switch to the dGPU. At this point the Ubuntu desktop will be output via DisplayPort on the dGPU, and so the display cable must be switched from the onboard HDMI to DisplayPort on the dGPU.

If at any time you want to switch back to iGPU, use the install command with the iGPU parameter:

$ sudo nvgpuswitch.py install iGPU

After the iGPU drivers have been installed, rebooting the system will complete the switch to the iGPU. At this point the Ubuntu desktop will be output via the onboard HDMI, and so the display cable must be switched from the DisplayPort on the dGPU to the onboard HDMI.

**Note:** The GPU settings will persist through reboots until it is changed again with nvgpuswitch.py.

### 8.1 Reinstalling Clara AGX SDK Packages

When switching between GPUs, CUDA is first uninstalled and then reinstalled by the script in order to provide the correct versions used by iGPU or dGPU (CUDA 10.2 and 11.1, respectively). Since many of the Clara AGX SDK installed packages depend on CUDA, this means that these Clara AGX SDK packages are also uninstalled when the active GPU is switched.

To reinstall the Clara AGX SDK packages after switching GPUs, the corresponding *.deb packages that were downloaded by SDK Manager during the initial installation can be copied to the Clara AGX Developer Kit and installed using apt. By default, SDK Manager downloads the *.deb packages to the following location:

```
~/Downloads/nvidia/sdkm_downloads
```

The current list of packages that must be reinstalled is as follows. Note that the version numbers may differ – if this is the case, use the latest version of the arm64 package that exists in the download directory.

**VisionWorks:** libvisionworks and libvisionworks-dev

**DeepStream 5.1:** deepstream-5.1_5.1.0-1_arm64.deb

**Clara AGX DeepStream Samples:** clara-agx-deepstream-sample-src_1.3.0-0_arm64.deb

**SUPRA:** supra-src_1.0.2-0_arm64.deb

Use `apt` to reinstall these packages:

```
$ sudo apt install -y libvisionworks libvisionworks-dev
$ sudo apt install -y ./deepstream-5.1_5.1.0-1_arm64.deb
$ sudo apt install -y ./clara-agx-deepstream-sample-src_1.3.0-0_arm64.deb
$ sudo apt install -y ./supra-src_1.0.2-0_arm64.deb
```

8.1. Reinstalling Clara AGX SDK Packages
Chapter 9

Rivermax Setup

The Clara AGX Developer Kit can be used along with an NVIDIA Mellanox ConnectX-6 Dx SmartNIC and the NVIDIA Rivermax SDK in order to provide an extremely efficient network connection taking advantage of GPUDirect. This technology avoids unnecessary memory copies and CPU overhead by copying data directly to or from pinned GPU memory, and supports both the integrated GPU or the RTX6000 add-in dGPU.

Follow the instructions below in order to setup and use a ConnectX-6 Dx network adapter and the Rivermax SDK with the Clara AGX Developer Kit.

Note: Access to the Rivermax SDK Developer Program as well as a valid Rivermax software license is required to use the Rivermax SDK.

1. Hardware Setup

   Using the Rivermax SDK with GPUDirect requires both the dGPU and ConnectX-6 Dx network adapter to be connected to the same PCI bridge. The Clara AGX Developer Kit has three PCI slots, only two of which share a PCI bridge, meaning that the dGPU and the network adapter need to be in these slots for Rivermax to function properly.

   The two slots that the dGPU and network adapter must be in are the outermost PCI slots on the AGX Developer Kit main board. It is expected that the dGPU is already installed in the PCI slot that has a retaining clip, and so the network adapter must be installed in the PCI slot furthest from that one on the opposite side of the main board.

   The following picture shows the required installation locations of the dGPU and the ConnectX-6 Dx network adapter for Rivermax GPUDirect operation.
2. **Install** Mellanox OpenFabrics Enterprise Distribution Drivers for Linux (OFED)

   (a) **Download OFED version 5.1-2.5.8.0:**

   MLNX_OFED_LINUX_5.1-2.5.8.0-ubuntu18.04-aarch64.tgz

   If the above link does not work, navigate to the **Downloads** section on the main OFED page, select **Archive Versions, 5.1-2.5.8.0, Ubuntu, Ubuntu 18.04, aarch64**, then download the **tgz** file.

   **Note:** Newer versions of OFED have not been tested and may not work.

   (b) **Install OFED:**
3. Install GPUDirect

Note: This step is only required when using an RTX6000 add-in dGPU. When using the iGPU the CPU and GPU share the unified memory and the GPUDirect drivers are not required, so this step may be skipped.

(a) Download GPUDirect Drivers for OFED:

```
nvidia-peer-memory_1.1.tar.gz
```

If the above link does not work, navigate to the Downloads section on the GPUDirect page.

(b) Install GPUDirect:

```
$ mv nvidia-peer-memory_1.1.tar.gz nvidia-peer-memory_1.1.orig.tar.gz
$ tar -xvf nvidia-peer-memory_1.1.orig.tar.gz
$ cd nvidia-peer-memory-1.1
$ dpkg-buildpackage -us -uc
$ sudo dpkg -i ../nvidia-peer-memory_1.1-0_all.deb
$ sudo dpkg -i ../nvidia-peer-memory-dkms_1.1-0_all.deb
$ sudo service nv_peer_mem start
```

Verify the `nv_peer_mem` service is running:

```
$ sudo service nv_peer_mem status
```

Enable the `nv_peer_mem` service at boot time:

```
$ sudo systemctl enable nv_peer_mem
$ sudo /lib/systemd/systemd-sysv-install enable nv_peer_mem
```

4. Install Rivermax SDK

(a) Download version 1.7.10 or newer of the Rivermax SDK from the NVIDIA Rivermax SDK developer page.

(b) Install Rivermax SDK:

```
$ tar -xvf rivermax_ubuntu1804_1.7.10.tar.gz
$ sudo dpkg -i 1.7.10/Ubuntu.18.04/deb-dist/aarch64/rivermax_11.2.8.10_arm64.deb
```

(c) Install Rivermax License

Using Rivermax requires a valid license, which can be purchased from the Rivermax Licenses page. Once the license file has been obtained, it must be placed onto the system using the following path:

```
/opt/mellanox/rivermax/rivermax.lic
```
5. Run the Rivermax SDK Sample Application

Running the sample applications requires two systems, a sender and a receiver, connected via the ConnectX-6 network adapters. The following instructions assume that all of the above installation steps have been followed on both systems and then one of the sender or receiver steps below are followed on each system.

Note: This assumes that the ConnectX-6 port being used by both devices use the logical name `eth1`. If this is not the case, refer to the output of the command `lshw -class network` to see the logical names that are being used.

Sender:

(a) Bring up the network:

```
$ sudo ifconfig eth1 up 1.1.3.102
```

(b) Build the sample apps:

```
$ cd 1.7.10/apps
$ make
```

(c) Download the sample media file.

On the NVIDIA Rivermax SDK page, download the file `mellanoxTV_2160p50.7z` then extract and move it to the application directory with the following:

```
$ 7za e mellanoxTV_2160p50.7z
$ mv mellanoxTV_2160p50.ycbcr 1.7.10/apps/
```

(d) Populate the SDP configuration file.

Write the file `1.7.10/apps/sdp_gap_flow_4K50fps_1.1.3.102.txt` with the following contents:

```
v=0
o=- 1443716955 1443716955 IN IP4 1.1.3.102
s=ST2110-20 2160p60 Video Stream
t=0 0
m=video 2000 RTP/AVP 96
c=IN IP4 239.1.1.1/64
a=source-filter:incl IN IP4 239.1.1.1 1.1.3.102
a=rtpmap:96 raw/90000
a=fmtp:96 sampling=YCbCr-4:2:2; width=3840; height=2160;
  →exactframerate=60; depth=10; TCS=SDR; colorimetry=BT709;
  →PM=2110GPM; SSN=ST2110-20:2017; TP=2110TPN;
a=mediaclk:direct=0
a=ts-refclk:localmac=40-a3-6b-a0-2b-2d
```

(e) Launch the sender application:

```
$ cd 1.7.10/apps
$ sudo ./media_sender -s sdp_gap_flow_4K50fps_1.1.3.102.txt -m -f
  →mellanoxTV_2160p50.ycbcr
```

Receiver:
(a) Bring up the network:

```sh
$ sudo ifconfig eth1 up 1.1.3.100
```

(b) Build the sample apps:

```sh
$ sudo apt install -y libglew-dev
$ cd 1.7.10/apps
$ make VIEWER=y CUDA=y
```

(c) Launch the receiver application:

```sh
$ export DISPLAY=:0.0
$ xhost +si:localuser:root
$ sudo ./media_receiver -i 1.1.3.100 -m 239.1.1.1 -p 2000 --
  display -x UHD -a 6 -c 7 -b 8 -r 20 --memory -d 1200 -g 0 -k
  131072
```

Once both the sender and receiver are operating, the media stream being sent by the sender should be rendered on the display of the receiver:

![Media Stream Rendering](image)

**Troubleshooting**

If running the sample applications do not work, check the following.

1. The ConnectX-6 Dx network adapter is recognized by the Clara AGX Developer Kit:

```sh
$ lspci
...
If the network adapter is not recognized, try rebooting the system and/or reseating the card in the PCI slot.

2. The ConnectX-6 Dx is recognized by OFED and using the correct firmware:

   $ sudo mlxfwmanager
   ...
   Device Type: ConnectX6DX
   Part Number: MCX623106AC-CDA_Ax
   Description: ConnectX-6 Dx EN adapter card; 100GbE; Dual-port QSFP56; PCIe 4.0 x16; Crypto and Secure Boot
   PSID: MT_00000000436
   PCI Device Name: 0000:05:00.0
   Base GUID: 0c42a1030024053a
   Base MAC: 0c42a124053a
   Versions: Current | Available
     FW  22.28.2006 | N/A
     PXE 3.6.0102  | N/A
     UEFI 14.21.0017 | N/A
   ...

   If the device does not appear or the versions do not match, first try rebooting and then reinstalling OFED (**2b**, above).

3. The sender and receiver systems can ping each other:

   $ ping 1.1.3.100
   PING 1.1.3.100 (1.1.3.100) 56(84) bytes of data.
   64 bytes from 1.1.3.100: icmp_seq=1 ttl=64 time=0.205 ms
   64 bytes from 1.1.3.100: icmp_seq=2 ttl=64 time=0.206 ms
   ...

   If the systems can not ping each other, try bringing up the network interfaces again (**5a**, above).

4. The `nv_peer_mem` service is running:

   $ sudo service nv_peer_mem status
   * nv_peer_mem.service - LSB: Activates/Deactivates nv_peer_mem to \_
     start at boot time.
     Loaded: loaded (/etc/init.d/nv_peer_mem; generated)
     Active: active (exited) since Mon 2021-01-25 16:45:08 MST; 9min ago
     Docs: man:systemd-sysv-generator(8)
     Process: 6847 ExecStart=/etc/init.d/nv_peer_mem start (code=exited,\_
     status=0/SUCCESS)
   Jan 25 16:45:08 mccoy systemd[1]: Starting LSB: Activates/Deactivates\_
     nv_peer_mem to \ start at boot time....
   Jan 25 16:45:08 mccoy nv_peer_mem[6847]: starting... OK
   Jan 25 16:45:08 mccoy systemd[1]: Started LSB: Activates/Deactivates\_
     nv_peer_mem to \ start at boot time..

   If the service is not running, try starting it again (**3b**, above).
Chapter 10

Data Set Generation

The Clara AGX SDK includes a sample data set that consists of 300 images of the Learning Resources Anatomy Model that has been annotated with stomach and intestines regions. The sample applications by default use a model that has been trained using this sample dataset, allowing the samples to be run out of the box without having to generate a new data set or train a new model. This sample data set has been included in the Clara AGX TLT package in the data/organs subdirectory, and is also the default in all training-related scripts included in this package.

Once the Clara AGX SDK sample applications have been installed and tested, a common next step for the user is to create their own data set and to train a new inference model that can be loaded by the sample application in order to perform inference using the user-provided data set.

There are many ways for data sets to be captured and annotated for training purposes, and these instructions describe just one method that has been used and simplified using the scripts included in the Clara AGX TLT package.

These instructions must be followed on an x86 Linux PC with a camera device available.

10.1 Capturing Images

The bin/capture.sh script assists in capturing images for the data set from a camera attached to the PC. When run, this script will run continuously and take pictures whenever the spacebar is pressed (or until ESC is used to quit).

This script depends on FFMPEG, which can be installed with the following command:

```
$ sudo apt install ffmpeg
```

We recommend capturing at least 300 images in a wide range of angles and lighting conditions in order to generate a useful inference model. If 300 pictures does not produce acceptable inference results once the model has been trained, it may be desirable to revisit this step and take even more.

Store the images output by this step into a single directory (e.g. /shared/tlt/workspace/data/organs).

Note: If the camera you are using for this step is not /dev/video0, you will need to change the device via the --dev <dev> parameter.
10.2 Labeling the Data Set

Once you have a data set, you will need to label it. These instructions use CVAT from OpenCV to perform labeling.

1. Install CVAT using these instructions.
2. Use Google Chrome to navigate to http://localhost:8080/dashboard/. This will launch CVAT.
3. Register a new user for CVAT. See the CVAT User Guide for more details.
4. Click Create New Task and specify the following:
   1. **Name**: Enter a name for the task (e.g. “Organs”).
   2. **Labels**: Add all labels that will be used with the data set as a space-separated list (e.g. “stomach intestines liver”).
   3. **Select Files**: Select the folder containing your image data set.
5. Click **Submit** to create the task.

6. Click the **Jobs** link associated with your new task.

7. Click **Create Shape** to create a bounding box for the selected label. Repeat this step for all labels added in step 4b.

8. Go to the next image and repeat step 7. Repeat this step until all images are labeled.

9. Select **Open Menu > Save Work**.

10. Select **Open Menu > Dump Annotation**. Then select **PASCAL VOC ZIP 1.0**. This should generate a ZIP file containing the annotations as XML files.
11. Copy the generated ZIP file to the same directory as your original image data set (e.g. /shared/tlt/workspace/data/organs).
Chapter 11

Training the Neural Network

Once a data set has been created and annotated, the Transfer Learning Toolkit (TLT) can be used to train a neural network and output a model that can be used for inference by DeepStream and the sample application.

These instructions can be used to train a model using a custom data set that was generated in the previous step, or can simply be followed as-is as an exercise to regenerate a model from the sample data set. Note that the sample applications already include the sample models, however, so this step is optional if not replacing the sample model in the application.

---

**Note:** These instructions assume the following local file structure: `/shared/tlt/workspace`. If your directory is structured differently, you will need to alter the file paths used below and edit the script files used in these instructions.


2. Once logged in, go to [https://ngc.nvidia.com/setup/installers/cli](https://ngc.nvidia.com/setup/installers/cli) to download and install the `ngc` command line tool.

3. After installing the `ngc` command line tool, fetch the API key via [https://ngc.nvidia.com/setup/api-key](https://ngc.nvidia.com/setup/api-key). The API key will be needed in step 5 to log in through the container (via `ngc config set`).

4. Install the Transfer Learning Toolkit for Video Streaming Analytics Docker image with the following command:

   ```bash
   $ sudo docker pull nvcr.io/nvidia/tlt-streamanalytics:v2.0_py3
   ```

4. Run the TLT Docker image using the following command:

   ```bash
   $ sudo docker run --gpus all -it --rm --name tlt-2.0 -v /shared/tlt/workspace:/workspace/tlt-experiments -w /workspace/tlt-experiments nvcr.io/nvidia/tlt-streamanalytics:v2.0_py3 /bin/bash
   ```

   **Note:** The TLT containers may not work when VPN is connected. If running the TLT containers does not work, disconnect the VPN connection, if connected, and try again.

5. Log into NGC using the API key that you got from step 3 (accept defaults for other options):
6. Copy ~/.ngc/config to ngc_config:

$ cp ~/.ngc/config ngc_config

7. Run the bin/detectnet.sh script and provide the name of the dataset.

$ bin/detectnet.sh --dataset <dataset name>

Where <dataset name> would be organs if using the sample dataset.

**Note:** The SDK Manager downloads detectnet.sh script during the initial installation at default location ~/.nvidia/nvidia_sdk/Clara_AGX_2.2_Linux/Clara_AGX_TLT/bin/ on the host machine. The detectnet.sh script assumes your data set is located in /workspace/tlt-experiments/data/<dataset-name>. You will provide the name of the dataset in the next step via the parameter to the detectnet.sh script. The following are additional requirements for the dataset:

- All images start with the <dataset-name>-
- All annotations start with the <dataset-name>-
- The images and annotation files start at '0' and be numbered consecutively without missing numbers.
- The annotations ZIP file is named <dataset-name>.zip.
- The model-specific spec files are located in spec/<dataset-name>/.

8. The previous step trains the neural network using TLT and outputs the trained models (the *.etlt files) to /workspace/tlt-experiments/results/experiment_dir_final.

In addition to the *.etlt model files, the models are also run through the x86 version of the tlt-converter in order to generate the TensorRT engine files that can be used with the TensorRT version and GPU architecture of the Linux host on which the script was run.

If the TensorRT engine files (i.e. *.trt) are to be used on another x86 machine with a different GPU or TensorRT version, the **Creating Linux TensorRT Engines** steps below must be followed to generate the engine files specific to that platform.

**Note:** The Clara AGX 2.2 release continues to use TLT version 2.0 for the above model training steps, which is based on TensorRT 7.0, but SDK Manager will install TensorRT 7.2 on the Linux host. The *.trt engine files that are output by TLT 2.0 are not compatible with the TensorRT 7.2 runtime, and so the engine files must be recreated using the TLT 3.0 version of tlt-converter as described in **Creating Linux TensorRT Engines**, below.
Chapter 12

Creating Jetson TensorRT Engines

Once the TLT model files (*.etlt) have been created, they must be converted to the platform-specific TensorRT engine files on the Jetson AGX device that they will be used. Follow these steps to transfer the training data to the Jetson AGX and create the TensorRT engine files.

1. Download the Jetson-specific TensorRT 7.1 tlt-converter binary and copy it to the Jetson device to a location accessible through $PATH (e.g. /usr/bin).

2. Copy the following files from the /experiment_dir_final directory to a local directory on Jetson:
   - calibration.bin
   - resnet18_detector.etlt
   - resnet18_detector_int8.etlt

3. Copy the bin/detectnet_convert.sh script to the same directory.

4. Run the detectnet_convert.sh script.

The Jetson *.trt engine files will be output into the same directory as the script. To use these new engine files, copy them to the DeepStream application directory or reference them from the DeepStream application configuration.
Chapter 13

Creating Linux TensorRT Engines

If the x86 Linux host does not execute the generated TRT engine binaries due to a diverging GPU architecture or TensorRT version, please follow these steps on the host that the engines will be used:

1. Copy the tlt-converter binary from the Transfer Learning Toolkit container image into a location accessible through $PATH (e.g. /usr/bin):

   $ sudo docker run -it --rm -v /usr/bin:/out nvcr.io/nvidia/tlt-
   →streamanalytics:v3.0-dp-py3 cp '/opt/nvidia/tools/tlt-converter /out

2. Copy the following files from the /experiment_dir_final directory to a local directory:
   • calibration.bin
   • resnet18_detector.etlt
   • resnet18_detector_int8.etlt

3. Copy the bin/detectnet_convert.sh script to the same directory.

4. Run the detectnet_convert.sh script.

The x86 *.trt engine files will be output into the same directory as the script. To use these new engine files, copy them to the DeepStream application directory or reference them from the DeepStream application configuration.

Note: If the engine files included with the sample application do not work on an x86 host due to diverging GPU architecture or TensorRT version, the above steps can be followed using the calibration.bin and *.etlt files that are included with the sample application in the directory /opt/nvidia/clara-agx-sdk/clara-agx-deepstream-sample/model/organs/.
Chapter 14

DeepStream and Endoscopy Sample Apps

14.1 Clara AGX Developer Kit and Jetson AGX

14.1.1 Installation

Install the Debian file for the Clara AGX DeepStream sample using the following command. If Clara AGX SDK was installed using NVIDIA SDK Manager, this step can be skipped.

$ sudo apt install ./clara-agx-deepstream-sample-src_*_arm64.deb

**Note:** This package has the following dependencies, which will be automatically installed with the above command:

- cmake
- v4l-utils
- libgtk2.0-dev
- libgtk-3-dev
- libgstreamer1.0-dev
- libgstreamer-plugins-base1.0-dev

**Important:** If you are re-installing or updating the Clara AGX DeepStream Sample package, you need to remove the old version first. See *Updating the DeepStream or Endoscopy Sample Apps* section below for more information.

14.1.2 Running Apps on Clara AGX Development Kit dGPU mode

The DeepStream and Endoscopy sample apps work in dGPU mode on Clara AGX Development Kit; however, the TensorRT engine files that are included with the sample apps are only compatible with the iGPU. Running the apps with the dGPU requires new TensorRT engine files to be generated. To do so, follow instructions given under *Creating Jetson TensorRT Engines.*
**Important:** Camera modules with CSI connectors are not supported in dGPU mode on Clara AGX Development Kit.

### 14.1.3 Endoscopy Sample Application

Follow these steps to run the Endoscopy Sample Application:

1. Navigate to the installation directory:

   ```
   $ cd /opt/nvidia/clara-agx-sdk/clara-agx-deepstream-sample/build/
   → EndoscopyJetsonAGXSample
   ```

2. Run the application:

   ```
   $ ./EndoscopyJetsonAGXSample
   ```

   The application window should appear.

   ![EndoscopySampleApplication](image)

   - All connected cameras should be visible on the top right.
   - Select a source by clicking one of the buttons named "Endoscope <N>". where <N> corresponds to the video source.
   - The main window area should show “Loading…” on the bottom while the camera is being initialized. When ready, the main window area will start displaying the live video stream from the camera.
• Click any of the "Class Control" buttons to dynamically enable or disable the bounding boxes and labels of detected objects.

• Click the button on the bottom right to quit the application.

**Tip:** By default, the app will launch with the FP16 network enabled. You can switch the network to FP32 or INT8 by changing the "model-engine-file" and "network mode" in the ejas_nvinfer_config.txt file.

**Tip:** By default, the app launches with the configuration parameters defined in the ejas_config.txt file. You can either modify start-up parameters there or create your own configuration file and start the application with $ ./EndoscopyJetsonAGXSample -c <your_config_file>

### 14.1.4 DeepStream Sample Application

Follow these steps to run the DeepStream Sample Application:

1. Navigate to the installation directory:

   ```bash
   $ cd /opt/nvidia/clara-agx-sdk/clara-agx-deepstream-sample/build/
   $ DeepstreamJetsonAGXSample
   ```

2. Run the application:

   ```bash
   $ ./DeepstreamJetsonAGXSample
   ```

3. The application window should appear.
• All connected cameras should be visible on the right.

• Use **Enable** to select which camera streams to process. This only works before you play the stream the first time after starting the application.

• Use **Flip** to flip the video stream across the horizontal axis (if available). This only works before you play the stream the first time after starting the application.

• Check the **Enable Streaming** box on the bottom to stream the processed output as a raw h.264 UDP stream. Once this box is checked, you can define the **Host Port** and **Host IP** address of the streaming destination. See the *Streaming DeepStream Sample to Host* section below for details on how to receive the stream on the host.

• Click the green arrow to start video capture and object detection.

**Important:** If streaming video from multiple camera sensors at the same time, ensure that they have compatible capture mode. To check capture modes, use: $ v4l2-ctl -d /dev/
video<N> --list-formats-ext where N corresponds to the video source.

- Close the app to end the capture session.

**Tip:** By default the app launches with the FP16 network enabled. You can switch the network to FP32 or INT8 by changing the "model-engine-file" and "network mode" in the “ds-jas_nvinfer_config.txt” file.

### 14.2 Linux x86 DeepStream Sample

**Note:** The Linux x86 AGX DeepStream Sample App only works with USB cameras. To avoid bandwidth issues, ensure that each USB camera is connected to its own USB controller. Depending on the motherboard in use, it is typically sufficient to connect one camera to the front and one to the back of the PC.

**Note:** On Linux x86, you may need to re-convert the TRT engines for your specific GPU architecture. Unlike with Tegra, it is difficult to predict the exact version of GPU and software running on your PC. In this case, the `DeepstreamLinuxHostSample` application may fail during playback with errors like the following:

```plaintext
ERROR: ../nvdsinfer/nvdsinfer_model_builder.cpp:1523 Deserialize engine
→ failed because file path: /opt/nvidia/clara-agx-sdk/clara-agx-deepstream-
→ sample/build/DeepstreamLinuxHostSample/model/resnet18_detector_fp16.trt
→ open error
0:00:05.776069825 25480 0x55de1633ed00 WARN nvinfer gstnvinfer.
→ cuda
→ NvDsInferContext[UID 1]: Warning from NvDsInferContextImpl::deserializeEngineAndBackend() <nvdsinfer_context_impl.cpp:1691> [UID = 1]: deserialize engine from file :/opt/nvidia/clara-agx-sdk/clara-agx-deepstream-sample/build/DeepstreamLinuxHostSample/model/resnet18_detector_fp16.trt failed
0:00:05.776097743 25480 0x55de1633ed00 WARN nvinfer gstnvinfer.
→ cuda
→ NvDsInferContext[UID 1]: Warning from NvDsInferContextImpl::generateBackendContext() <nvdsinfer_context_impl.cpp:1798> [UID = 1]: deserialize backend context from engine from file :/opt/nvidia/clara-agx-sdk/clara-agx-deepstream-sample/build/DeepstreamLinuxHostSample/model/resnet18_detector_fp16.trt failed, try rebuild
0:00:05.776107361 25480 0x55de1633ed00 INFO nvinfer gstnvinfer.
→ cuda
→ NvDsInferContext[UID 1]: Info from NvDsInferContextImpl::buildModel() <nvdsinfer_context_impl.cpp:1716> [UID = 1]: Trying to create engine from model files
ERROR: ../nvdsinfer/nvdsinfer_model_builder.cpp:934 failed to build network since there is no model file matched.
0:00:05.776325716 25480 0x55de1633ed00 ERROR nvinfer gstnvinfer.
→ cuda
→ NvDsInferContext[UID 1]: Error in NvDsInferContextImpl::buildModel() <nvdsinfer_context_impl.cpp:1736> [UID = 1]: build engine file failed
```

14.2 Linux x86 DeepStream Sample
If this is case, please follow the instructions under `Model Creation > Creating Linux TensorRT Engines`

1. Ensure all Linux x86 prerequisites are installed as described on the `Prerequisites` page.

2. Install the Debian file for the Clara AGX DeepStream Sample:

   ```bash
   $ sudo apt install ./clara-agx-deepstream-sample-src_*_amd64.deb
   ``

**Important:** If you are re-installing or updating the Clara AGX DeepStream Sample package, you need to remove the old version first. See the `Updating the DeepStream or Endoscopy Sample Apps` section below for more information.

3. Navigate to the installation directory:

   ```bash
   $ cd /opt/nvidia/clara-agx-sdk/clara-agx-deepstream-sample/build/DeepstreamLinuxHostSample
   ```

4. Run the application:

   ```bash
   $ ./DeepstreamLinuxHostSample
   ```

   The application window should appear. See Step 5 of the `Clara AGX Developer Kit and Jetson AGX` section above for more details about the application.
14.3 Streaming DeepStream Sample to Host

Run the following command on the x86 Linux or Windows host in order to receive the video stream output by the DeepStream sample application:

```
$ gst-launch-1.0 udpsrc port=5000 ! application/x-rtp,encoding-name=H264,.payload=96 ! rtph264depay ! queue ! avdec_h264 ! autovideosink sync=false
```

14.4 Updating the DeepStream or Endoscopy Sample Apps

If you wish to update the Clara AGX DeepStream Sample package to a new version, you must remove the currently installed Debian package first. To do so, use the following command:

```
$ sudo apt remove clara-agx-deepstream-sample-src
```

Then, perform the installation as described in the steps above.

14.5 Using Docker for the DeepStream or Endoscopy Sample Apps

Docker provides lightweight reusable containers for deploying and sharing your applications.

Due to the way containers are used with Jetson, with host-side libraries mounted into the container in order to force library compatibility with the host drivers, there are two ways to build a container image containing the AGX samples:

1. Build the sample applications on the Jetson, outside of a container, then copy the executables into an image for use in a container.

2. Build the sample applications within a container.

These instructions document the process for #2, since it should be relatively straightforward to adapt the instructions to use method #1.

14.5.1 Configure the Container Runtime

The DeepStream libraries are not mounted into containers by default, and so an additional configuration file must be added to tell the container runtime to mount these components into containers.

1. Add the following to a new file, `/etc/nvidia-container-runtime/host-files-for-container.d/deepstream.csv`:

```
dir, /opt/nvidia/deepstream/deepstream-5.1
sym, /opt/nvidia/deepstream/deepstream
```

2. Configure Docker to default to use `nvidia-docker` with "default-runtime": "nvidia" in `/etc/docker/daemon.json`:

```json
{
    "runtimes": {
        "nvidia": {
            "path": "nvidia-container-runtime"
        }
    }
}
```
3. Restart the Docker daemon:

```bash
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

### 14.5.2 Build the Container Image

After installing the SDK sample code via instructions outlined in *Installation*, navigate to the `/opt/nvidia/clara-agx-sdk/clara-agx-deepstream-sample` directory and run:

```bash
$ sudo docker build -t clara-agx-samples -f Dockerfile.igpu .
```

### 14.5.3 Running the Samples in a Container

Since the samples are graphical applications and need permissions to access the X server, the DISPLAY environment must be set and root (the user running the container) must have permissions to access the X server:

```bash
$ export DISPLAY=:0.0
$ xhost +si:localuser:root
```

Then, using the clara-agx-samples image that was previously built, the samples can be run using the following:

```bash
$ sudo docker run --rm --device /dev/video0:/dev/video0
 --env DISPLAY=$DISPLAY --volume /tmp/.X11-unix/:/tmp/.X11-unix
 --network host --volume /tmp:/tmp
 --workdir /opt/nvidia/clara-agx-sdk/clara-agx-deepstream-sample/build/
 \DeepstreamJetsonAGXSample
 clara-agx-samples ./DeepstreamJetsonAGXSample
```

If you have no `/dev/video0` device this means you do not have a camera attached. Please see *Jetson Camera Setup*.

The various arguments to `docker run` do the following:

- **--device /dev/video0:/dev/video0**: Give the container access to the camera device `/dev/video0`
- **--env DISPLAY=$DISPLAY --volume /tmp/.X11-unix/:/tmp/.X11-unix**: Give the container access to the X desktop
- **--network host --volume /tmp:/tmp**: Give the container access to the Argus socket used by the Camera API
  DeepstreamJetsonAGXSample: Run the DeepstreamJetsonAGXSample from the clara-agx-samples image.
Chapter 15

SUPRA - Open Source SW Beamforming Application

1. Install and build prerequisites:

   $ sudo apt install cmake cmake-gui qt5-default libtbb-dev
   → libopenigtlink-dev libcpprest-dev libboost-all-dev

2. Install the nvSUPRA Debian file, which will self-build:

   Note: If you are re-installing or updating SUPRA, you need to remove the old version first. Use the following command to uninstall SUPRA:

   $ sudo apt remove supra-src

   1. For Linux x86:

      Important: You must either set CUDA_TOOLKIT_ROOT_DIR in your environment variables or create a symbolic link from /usr/local/cuda to your CUDA installation directory (e.g. ln -s /usr/local/cuda-10.2 /usr/local/cuda).

      $ dpkg -i supra-src_*_amd64.deb

   2. For Jetson AGX:

      $ dpkg -i supra-src_*_arm64.deb

3. Navigate to the installation directory:

   $ cd /opt/supra/src/build

4. Run the GUI application:

   $ src/GraphicInterface/SUPRA_GUI -c data/configDemo.xml -a

5. Enable the visualization for all phases except “MHD”. If it doesn’t autostart, run execution by clicking Start.
15.1 Running SUPRA in a Container

1. Configure the container runtime by following the steps given in the Configure the Container Runtime section.

2. Build the supra container image. The Dockerfile that is used to build the image depends on the current system configuration:

   (a) Clara AGX Developer Kit (arm64) with iGPU:
   
   ```
   $ sudo docker build -t supra -f /opt/supra/src/Dockerfile.igpu /opt/supra/src/
   ```

   (b) Clara AGX Developer Kit (arm64) with dGPU:
   
   ```
   $ sudo docker build -t supra -f /opt/supra/src/Dockerfile.dgpu /opt/supra/src/
   ```

   (c) Host (amd64):
   
   ```
   $ sudo docker build -t supra -f /opt/supra/src/Dockerfile.amd64 /opt/supra/src/
   ```

3. Run the SUPRA_GUI demo in a container.

   First, set the DISPLAY environment variable and grant the root user permissions to access the X server:

   ```
   $ export DISPLAY=:0.0
   $ xhost +si:localuser:root
   ```

   Then, run the SUPRA_GUI demo:

   ```
   $ sudo docker run --rm --env DISPLAY=$DISPLAY --volume /tmp/.X11-unix:/tmp/.X11-unix supra src/GraphicInterface/SUPRA_GUI -c data/configDemo.xml -a
   ```