DXNN All Suite (DX-AllSuite) User Manual

2026/04/10 (v2.3.0)

DEEPX.ai

Copyright 2026 DEEPX All Rights Reserved.

Table of contents

1. DX-AllSuite Architecture Overview 3
1.1 Why DX-AllSuite? 3
1.2 System Architecture & Core Modules 4
1.3 Technical Compatibility (Support Matrix) 5
1.4 DEEPX ModelZoo & Supported Tasks 8
2. Setting Up Environment 11
2.1 Prerequisites 11
2.2 Docker Installation 15
2.3 Local Installation 21
3. Running Your First NPU Model 26
3.1 Overall Pipeline 26
3.2 DX-Compiler: AI Model Compilation Scripts Guide (Steps 0–4) 27
3.3 DX-Runtime: Application Execution (Application Execution Scripts Guide) (Step 0, 5) 29
4. Version Compatibility 33
4.1 DXNN SDK Version Compatibility Matrix 34
4.2 How to Check Your System Version 36
5. FAQ Troubleshooting Guide 37
5.1 Q1. Container 'Restarting' Error (`dxrtd` Conflict) 37
5.2 Q2. X11 Session Warnings & Mount Errors (Wayland Issues) 38
5.3 Q3. Firmware Version Mismatch Error 40
5.4 Q4. Device Driver Update Error 41
5.5 Q5. Model-Runtime Version Compatibility Error 42

Copyright 2026 DEEPX All Rights Reserved.

1. DX-AllSuite Architecture Overview

This guide provides a comprehensive overview of the **DXNN (DEEPX Neural Network)** SDK architecture, its core modules, and supported environments.

1.1 Why DX-AllSuite?

DX-AllSuite integrates the complex pipeline—from model optimization to target hardware deployment—into a **Single Workflow**.

- **Zero-Code Deployment**: Deploy logic verified on a desktop directly to edge devices without any code modifications.
- **Resource Optimization**: Reduce the time spent on environment setup and system integration, allowing you to focus on building high-performance AI applications.
- **End-to-End Solution**: Provides model compilation, simulation, runtime execution, and monitoring within a single package.

Copyright 2026 DEEPX All Rights Reserved.

1.2 System Architecture & Core Modules

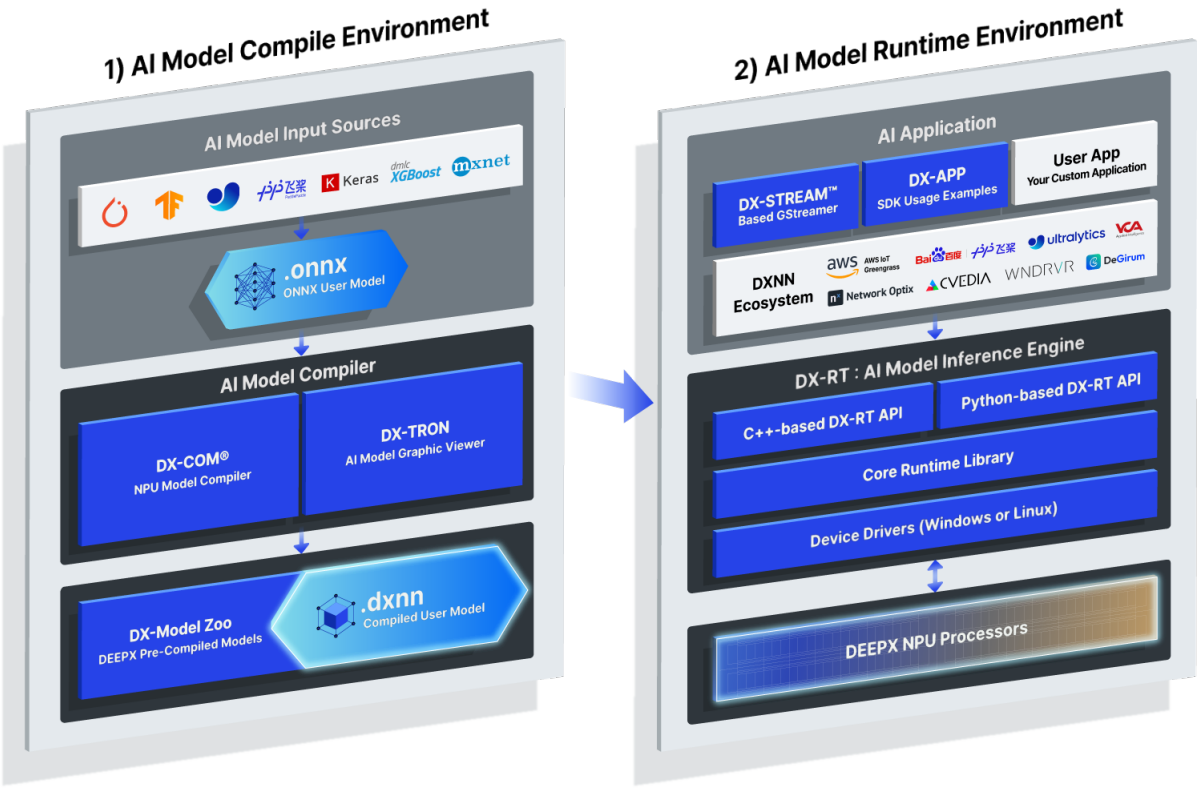

Figure. DXNN SDK Full Architecture Overview.

[1] AI Model Compile Environment (Host Platform)

This environment converts and optimizes trained AI models into DEEPX NPU-proprietary binaries (`.dxnn`). It supports the **x86_64** PC environment.

- **DX-COM (Compiler)**: The core compiler that generates `.dxnn` binaries based on ONNX models and user configuration (JSON) files. It uses proprietary algorithms to convert models into hardware-optimized instructions while minimizing accuracy loss, enabling low-latency and high-efficiency inference.
- **DX-TRON (Visualizer)**: A GUI tool for visualizing and analyzing the structure of `.dxnn` models. It provides color-coded graphs to intuitively show workload distribution between the NPU and CPU, helping developers understand execution flow and optimize performance.
- **DX-ModelZoo**: A repository of pre-trained models optimized for DEEPX NPUs. By providing ONNX models, JSON configs, and pre-compiled `.dxnn` binaries, it allows developers to test and deploy models immediately without going through the compilation process.

Copyright 2026 DEEPX All Rights Reserved.

[2] AI Model Runtime Environment (Target Platform)
This environment runs optimized `.dxnn` models on actual NPU hardware and integrates them into applications. It supports both **x86_64** and **aarch64** environments.

- **DX-RT (Runtime)**: The core software that interacts with firmware and drivers to run `.dxnn` models on the DEEPX NPU. It manages model loading, I/O buffers, inference execution, and real-time monitoring, offering flexible control via C/C++ and Python APIs.
- **DX-APP**: Sample applications built on DX-RT. It provides reference code for key vision tasks (Object Detection, Face Recognition, Image Classification, etc.), enabling developers to quickly build unique AI apps using these templates.
- **DX-Stream**: A GStreamer-based custom plugin that connects real-time video data to DEEPX NPU inference. It provides a modular pipeline covering pre-processing, inference, and post-processing, optimized for high-performance vision AI applications like intelligent cameras.
- **Driver & FW**: Supports high-speed data communication via the PCIe interface and manages NPU resource scheduling and power. It ensures stability between hardware and software while maximizing the hardware's potential for a seamless inference environment.

Development Workflow
The path from a trained model to NPU acceleration follows four logical steps.

- **Step 1. Model Preparation**: Export a model trained in PyTorch or TensorFlow to the **ONNX** format.
- **Step 2. Optimization (Host)**: Compile the model using **DX-COM** with quantization and hardware optimization to generate a `.dxnn` file.
- **Step 3. Deployment (Target)**: Transfer the generated `.dxnn` file to the target device.
- **Step 4. Execution (Target)**: Run high-speed inference on the DEEPX NPU via **DX-RT**.

1.3 Technical Compatibility (Support Matrix)

DX-AllSuite provides proven compatibility with the latest operating systems and a wide range of hardware architectures, ensuring a stable bridge between your development environment and edge deployment.

Copyright 2026 DEEPX All Rights Reserved.

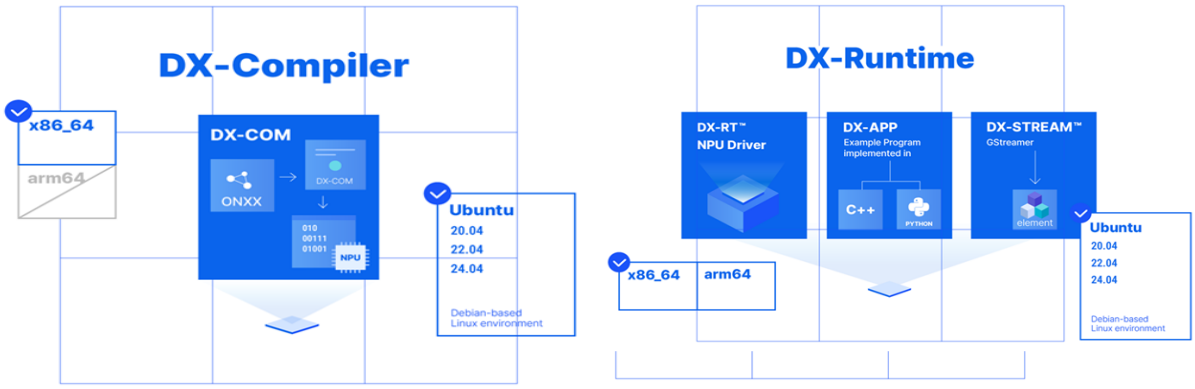

Figure. DX-Compiler & DX-Runtime Support Ecosystem.

1.3.1 Hardware & OS (Platform)

The following table outlines the supported environments for both the compilation (Host) and execution (Target) phases.

Category	DX-Compiler (Host)	DX-Runtime (Target)
Architecture	x86_64	x86_64, aarch64
OS	Ubuntu 24.04/22.04/20.04, Fedora, Redhat, CentOS	Ubuntu 24.04/22.04/20.04/18.04, Debian 13/12, Windows 11/10
Languages	Python 3.8, 3.9, 3.10, 3.11, 3.12	Python 3.8 or higher, C++14 or higher (C++17 for MSVC/Windows)

1.3.2 Model & Software Ecosystem

Supported AI Frameworks

DX-AllSuite seamlessly integrates with industry-standard frameworks to minimize refactoring.

- **Frameworks**: Ultralytics (YOLO), TensorFlow, PyTorch, Keras
- **Formats**: ONNX (Primary exchange format)

Copyright 2026 DEEPX All Rights Reserved.

Global AI Ecosystem Partners

We maintain strong technical partnerships to ensure the DXNN SDK operates flawlessly within various industry-leading platforms.

- **Cloud & Platform**: AWS (IoT Greengrass), Baidu, DeGirum
- **Vision & Algorithms**: Ultralytics (YOLO Series), CVEDIA
- **VMS & Security**: Network Optix (Nx), VCA (Applied Intelligence)
- **Embedded OS**: Wind River (VxWorks)

Copyright 2026 DEEPX All Rights Reserved.

1.4 DEEPX ModelZoo & Supported Tasks

DEEPX ModelZoo is a comprehensive repository providing **over 270 pre-validated models**. It allows users to immediately verify performance and accuracy across various hardware profiles without the need for a manual compilation process.

Copyright 2026 DEEPX All Rights Reserved.

Task	Representative Models
Image Classification	ResNet, ResNeXt, MobileNet, EfficientNet (Lite/V2), ViT/DeiT/BEiT, MobileViT, FastViT, CasViT, RegNet, ShuffleNet, VGG
Object Detection	YOLO families (YOLOv3–YOLOv11, YOLOX, YOLO26), SSD, EfficientDet, NanoDet, DamoYOLO
Instance Segmentation	YOLACT, YOLOv5-Seg, YOLOv8-Seg, YOLO26-Seg
Semantic Segmentation	DeepLabV3/DeepLabV3+, SegFormer, BiSeNet, UNet
Oriented Object Detection (OBB)	YOLO26-OBB
Zero-shot Instance Segmentation	FastSAM
Face Detection	RetinaFace, SCRFD, ULFGED, YOLOv5-Face, YOLOv7-Face
Face Recognition	ArcFace (IResNet50/100, MobileFaceNet, R50)
Face Landmark	TDDFA v2 (MobileNet variants)
Face Attribute	FaceAttrResNetV1-18
Pose Estimation (Human)	CenterPose, YOLO26-Pose, YOLOv8-Pose
Hand Landmark	MediaPipeHandsLite
Person Attribute	DeepMAR (ResNet18/50)
Depth Estimation	FastDepth, SCDepthV3
Image Denoising	DnCNN variants
Low-light Enhancement	ZeroDCE
Super-resolution	ESPCN (x2/x3/x4)

Copyright 2026 DEEPX All Rights Reserved.

Key Features of the ModelZoo Pipeline

- **YAML-Centric Orchestration**: Manage pre-processing, post-processing, evaluation, and compilation settings in a single YAML file for high reproducibility and easy operation.
- **Unified Workflow**: Integrates list, info, eval, compile, and benchmark functions into a single CLI system, allowing flexible combination of stages.
- **Optimized Multi-Profile Support**: All models are quantized and optimized for the DEEPX NPU architecture, ensuring consistent validation from ONNX evaluation to NPU execution.
- **Extensible Registry**: Supports plugin-style extensions for pre/post-processing, datasets, and evaluators, enabling fast onboarding of custom models.
- **Broad Task Coverage**: Extensive support for service-ready CV tasks beyond basic classification and detection, including Face Analysis, OBB, and Image Enhancement.

Copyright © DEEPX. All rights reserved.

Copyright 2026 DEEPX All Rights Reserved.

2. Setting Up Environment

DX-AllSuite is an integrated environment construction tool for verifying and utilizing DEEPX devices. This guide resolves complex dependency issues and provides a consistent development experience across both local hosts and Docker containers.

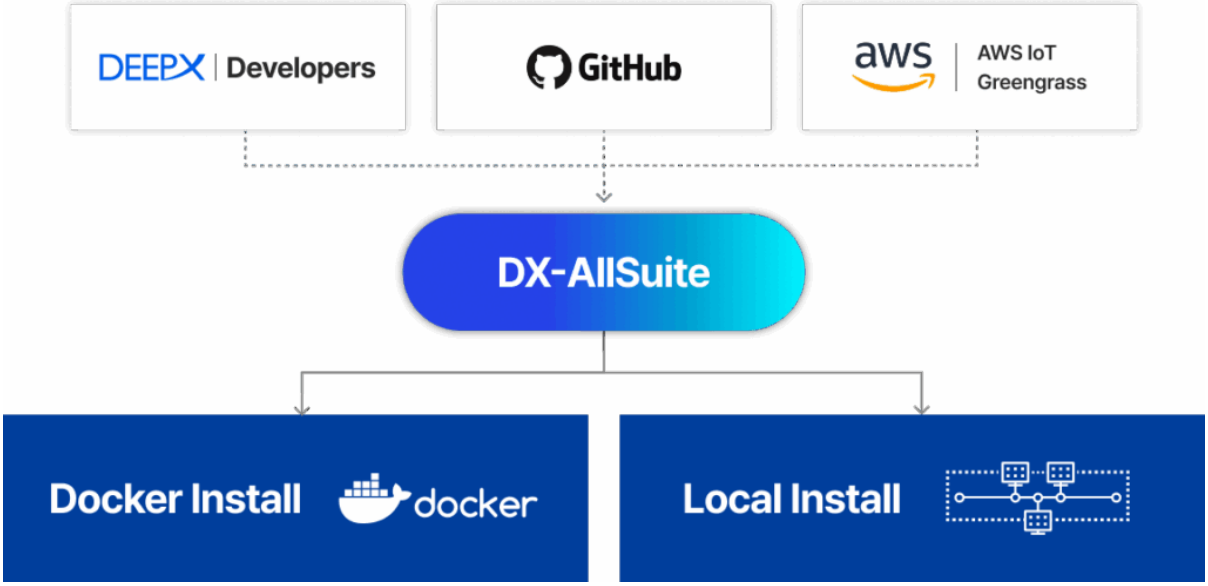

Figure. DX-AllSuite Supported Environments & Integrations.

Installation Summary

[Common] 1. Prerequisites: Acquire source code and understand virtual environment management policies.

[Optional] Choose Installation Path.
- **[Route A] 2. Docker Installation**: Isolated installation based on containers.
- **[Route B] 3. Local Installation**: Direct installation on the host OS.

2.1 Prerequisites

Follow these steps first to ensure a stable installation.

Copyright 2026 DEEPX All Rights Reserved.

2.1.1 Repository Cloning and Submodule Synchronization

DX-AllSuite is a collection of several independent modules. If submodules are missing, compilation and runtime execution will not be possible. Please use the following commands exactly.

A. Clone Repository (Including Submodules)

```
# Via HTTPS (Recommended)
git clone --recurse-submodules https://github.com/DEEPX-AI/dx-all-suite.git

# Via SSH
git clone --recurse-submodules git@github.com:DEEPX-AI/dx-all-suite.git

cd dx-all-suite
```

B. (Optional) Update Existing Repository

If you have already cloned the repository but the subfolders are empty, you **must** perform manual initialization.

```
# Initialize and update submodules to the latest state
git submodule update --init --recursive

# Check submodule status (Success if no '-' prefix exists)
git submodule status
```

C. (Optional) Prepare Docker Environment

If you plan to use the Docker route but do not have Docker installed, use the provided automation script.

```
# Automatic installation of Docker and Docker Compose
./scripts/install_docker.sh
```

2.1.2 Automated Environment Management

DX-AllSuite automates the creation of Python virtual environments (venv) to prevent package conflicts. The installation scripts automatically configure independent environments optimized for each execution context, so users do not need to create virtual environments manually.

- **Compiler Environment**: Created at `dx-compiler/venv-dx-compiler`
- **Runtime Environment**: Created at `dx-runtime/venv-dx-runtime`

Copyright 2026 DEEPX All Rights Reserved.

> **CAUTION**
>
> If you are installing an individual module (e.g., `dx-rt` only), you **must** activate the corresponding virtual environment (`source .../activate`) created by the script to avoid installation failure.

2.1.3 SDK Workflow Guide

DX-AllSuite provides two paths for utilizing the DEEPX NPU, depending on whether you are bringing your own model or using a pre-optimized one. The SDK automatically manages the necessary virtual environments and dependencies to ensure a seamless development experience.

[Path A] Custom Model Inference Path
The standard workflow for converting and deploying user-trained models.

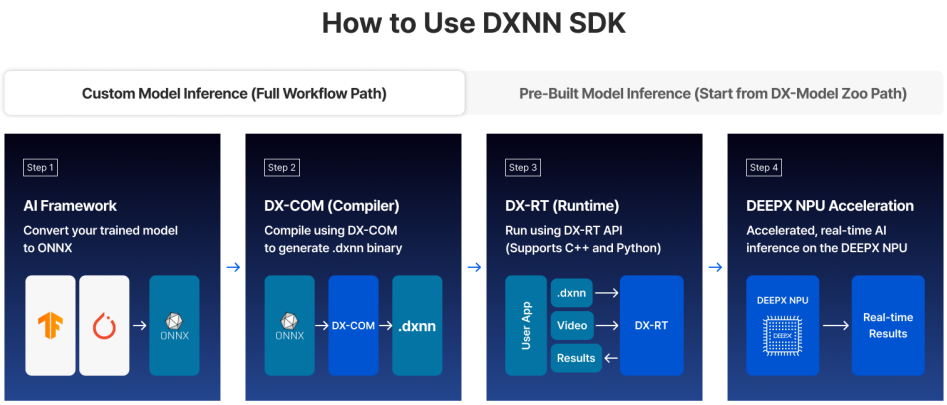

Figure. Custom Model Inference.

Copyright 2026 DEEPX All Rights Reserved.

This path is designed for users who have a specific model architecture trained in frameworks and need to optimize it for DEEPX hardware.

- **Step 1 (Source)**: Secure your trained models and source code from major AI frameworks such as PyTorch or TensorFlow.
- **Step 2 (Export)**: Export the model to **ONNX**, the standard format recognized by **DX-COM**.
  : Tip: Set input tensor size and use **Opset 11 or higher** to ensure maximum compatibility with NPU specifications.
- **Step 3 (DX-COM)**: Convert the ONNX model into an NPU-optimized `.dxnn` binary within the compiler virtual environment (`venv-dx-compiler`).
- **Step 4 (DX-RT)**: Load the generated model and run inference on the target device using the runtime virtual environment (`venv-dx-runtime`).
- **Step 5 (NPU Acceleration)**: Verify real-time AI inference performance and inspect the final output.

[Path B] Pre-Compiled Model Path (Fast Track)
The "Fast Track" for immediate hardware validation.

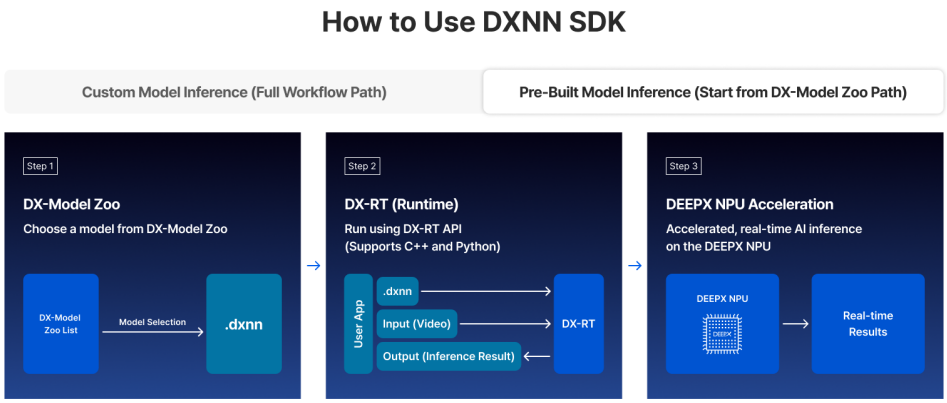

Figure. Pre-Built Model Inference.

This path is ideal for users who want to quickly benchmark DEEPX NPU performance or test hardware integration using industry-standard models.

- **Step 1 (Select)**: Choose a pre-validated `.dxnn` model from the **DEEPX ModelZoo** or sample data.
- **Step 2 (DX-RT)**: Load the selected model immediately in the runtime environment (`venv-dx-runtime`) without a separate compilation process.
- **Step 3 (NPU Acceleration)**: Execute hardware-accelerated inference and analyze key performance indicators like **FPS** and **Latency**.

Copyright 2026 DEEPX All Rights Reserved.

2.2 Docker Installation

Docker allows you to run **DX-AllSuite** in an isolated environment without complex dependency settings.

2.2.1 Host System Preparation (Critical)

Because Docker containers share the host kernel, **the driver must be installed on the host system (PC) first** for NPU hardware recognition.

A. Install NPU Driver (Host)
Execute the installation script on your **Host PC** first.

```
./dx-runtime/install.sh --target=dx_rt_npu_linux_driver
```

B. Prevent Service Daemon (`dxrtd`) Conflicts
Only one instance of `dxrtd` can run across the host and containers. Stop the host service before running the container.

```
sudo systemctl stop dxrt.service
```

2.2.2 Build Docker Image and Run Container

A. Build Image
Using the `--all` option builds an integrated image containing the Compiler, Runtime, and ModelZoo.

```
# Build integrated image (Based on Ubuntu 24.04)
./docker_build.sh --all --ubuntu_version=24.04

# Build specific environment only (Using --target)
./docker_build.sh --target=dx-runtime --ubuntu_version=24.04
```

B. Run Container
Run the container after the image build is complete.

```
./docker_run.sh --all --ubuntu_version=24.04
```


Copyright 2026 DEEPX All Rights Reserved.

> ⚠ **Note on GUI Environments**
>
> If you encounter X11 warnings or mount errors (e.g., cannot open display), it is likely due to the host OS using a **Wayland** session. Refer to **Q2. X11 Session Warnings & Mount Errors (Wayland Issues)** in **05. FAQ Troubleshooting Guide**.

2.2.3 Container Access and Task Guide

A. DX-Compiler Environment (Model Conversion)

The DX-Compiler environment is used to generate hardware-optimized `.dxnn` binaries.

A-1. Accessing the Container

To perform tasks within the container, you **must** first log in to the running container's shell.

```
# 1. Run in Host Terminal: Enter the container
docker exec -it dx-compiler-24.04 bash

# 2. Inside Container: Navigate to the work directory
cd /deepx/dx-compiler/dx_com
```

> ⚠ **Path Logic Notice**
>
> The `/deepx` path is an absolute path **internal** to the container. This path does not exist on your host machine. Before executing commands, ensure your terminal prompt has changed to `root@...` or `user@container_id`.

Copyright 2026 DEEPX All Rights Reserved.

A-2. Compiling Sample Model

Sample models are pre-downloaded to the `./sample_models/` directory upon installation. You can compile them using one of the following two methods.

- **Method 1**: Batch Compilation (Recommended)
  Use the provided script to compile all sample models automatically.

```
../example/3-compile_sample_models.sh
```

- **Method 2**: Manual compilation (CLI)
  For granular control, activate the virtual environment and use the `dxcom` tool directly.

```
source ../venv-dx-compiler/bin/activate  # Activate venv

dxcom -m sample_models/onnx/YOLOV5S-1.onnx \
      -c sample_models/json/YOLOV5S-1.json \
      -o output/YOLOV5S-1
```

A-3. Checking the Results

Upon successful completion, the optimized `.dxnn` binary will be generated in the `output/` directory (or the path specified by the `-o` flag).

- **Output File**: `output/YOLOV5S-1.dxnn`
- **Next Step**: Transfer this file to the **Runtime Environment** for hardware execution.

B. DX-Runtime Environment (NPU Inference & Streaming)

The **DX-Runtime** environment is designed for executing model inference and processing high-performance video streams using DEEPX NPU hardware.

B-1. Accessing the Container and Status Check

Before running any inference, ensure the container can communicate with the NPU hardware.

```
# 1. Run from the host terminal: Enter the container
docker exec -it dx-runtime-24.04

# 2. Inside Container: Verify NPU hardware recognition
dxrt-cli -s
```


Copyright 2026 DEEPX All Rights Reserved.

B-2. Running Sample Applications (`dx_app`)

This module provides inference demos for various vision tasks.

- **Working Directory**: `/deepx/dx-runtime/dx_app`

```
cd /deepx/dx-runtime/dx_app

# 1. Prepare resources (Download .dxnn models and sample images)
./setup.sh

# 2. Run demo
./run_demo.sh           # Run C++ based demo
./run_demo_python.sh        # Run Python demo
```

> **Demo Selection**
>
> Upon execution, the terminal will display a list of available demos (0, 1, 2...). Simply enter the corresponding number and press **Enter** to start.

B-3. Running the Streaming Framework (`dx_stream`)

This module is a GStreamer-based module optimized for real-time multi-channel video stream processing.

- **Working Directory**: `/deepx/dx-runtime/dx_stream`

```
cd /deepx/dx-runtime/dx_stream

# 1. Prepare assets (Downloads streaming-specific models and video assets)
./setup.sh

# 2. Run the streaming demo (C++ based)
./run_demo.sh
```

> **Scenario Selection**
>
> You can select specific streaming scenarios by entering the scenario number displayed in the terminal.

Copyright 2026 DEEPX All Rights Reserved.

Path Precautions

It is critical to distinguish between your **Host** terminal and the **Container** terminal to avoid "`File not found`" errors.

- **Inside Docker Container**: Always use the absolute path starting with `/deepx` (e.g., `cd /deepx/dx-runtime/...`).
- **Local Host Environment**: Use relative paths based on your current directory (e.g., `cd ./dx-runtime/...`).

> ⚠ **Common Error**
>
> If you attempt to enter a path starting with `/deepx` in a Host terminal, the system will return a `No such file or directory` error. Always check if your prompt starts with `root@...` or `user@container_id` before navigating.

2.2.4 [Docker] Advanced Troubleshooting (Multi-Runtime Containers)

The `dxrtd` daemon **must** run as a single instance (Singleton) within the system. To launch multiple containers simultaneously, you **must** modify the `Entrypoint` to prevent automatic execution.

- **Method 1**: Modifying Dockerfile
  Edit the `docker/Dockerfile.dx-runtime` file to disable the default startup command and replace it with a persistent idle state.

```
# 1. Comment out the existing settings
# ENTRYPOINT [ "/usr/local/bin/dxrtd" ]

# 2. Enable infinite wait settings to keep containers running
ENTRYPOINT ["tail", "-f", "/dev/null"]
```

- **Method 2**: Modifying docker-compose
  If you are using Docker Compose, you can overwrite the default Entrypoint directly in `docker/docker-compose.yml` within the corresponding service section.

```
services:
dx-runtime:
        entrypoint: ["/bin/sh", "-c"]
        command: ["sleep infinity"]
```


Copyright 2026 DEEPX All Rights Reserved.

Manual Startup

When the above settings are applied, the NPU will be in a waiting state. After entering the container, run it manually using the `dxrtd &` command.

2.2.5 [Docker] Verification of Installation Results (Sanity Check)

This performs a final check to ensure that the installation was completed successfully and that the software and hardware communicate correctly.

A. Hardware Recognition Check (`dxrt-cli`)

Run the following command inside the container to verify that the NPU is visible and operational:

```
dxrt-cli -s
```

Success Checklist
If your output meets the following three conditions, the hardware integration is successful:

- **[x] Device Recognition**: `Device 0: M1` (or your specific model) is displayed.
- **[x] Version Information**: Valid version numbers appear for `RT Driver version`, `FW version`, etc.
- **[x] Daemon Status**: No error messages such as " `Other instance of dxrtd is running` " are present.

[Example of Normal Output]

```
DX-RT v3.2.0
=======================================================
* Device 0: M1, Accelerator type
--------------------- Version ---------------------
* RT Driver version : v2.1.0
* FW version :      v2.5.0
-------------------------------------------------------
... (Continued)
```


Copyright 2026 DEEPX All Rights Reserved.

B. System Consistency Check

This script performs a batch check to ensure all individual modules are correctly placed in their designated paths and are ready for execution.

```
# Check the integrity of the runtime environment
./dx-runtime/scripts/sanity_check.sh
```

If **[OK]** or **PASS** is output for all items, you are ready to start service development.

2.3 Local Installation

Installing **DX-AllSuite** directly on the **Host OS** ensures maximum hardware performance and seamless compatibility between all software modules. This method is recommended for production environments and advanced performance benchmarking.

2.3.1 DX-Compiler Installation (DX-COM, DX-TRON)

DX-Compiler (DX-COM) can be used as a CLI tool or a Python module on supported Linux distributions.

Differences in Usage
- **CLI Tool (Command Line Interface)**: Perform compilation by entering `dxcom` commands directly in the terminal (Bash). Ideal for quick execution and automated shell scripts without additional coding.
- **Python Module (Library)**: Call functions or classes via `import dx_com` within your Python scripts. This is the preferred method for integrating the compiler into your existing AI training or automation pipelines.

> ⚠ **Change in Distribution Method**
>
> The standalone executable distribution method is **no longer supported**. This guide describes the latest **Wheel-based** installation workflow, which ensures better dependency management and Python environment integration.

Copyright 2026 DEEPX All Rights Reserved.

A. Pre-Installation Requirements

Before installing **DX-COM**, you **must** install the following system libraries to support core utilities and graphical operations.

- `libgl1-mesa-glx`: OpenGL runtime support for graphics processing
- `libglib2.0-0`: Core utility library (related to GNOME/GTK)

Installation Command

```
sudo apt-get update
sudo apt-get install -y --no-install-recommends libgl1-mesa-glx libglib2.0-0 make
```

B. Installation Method

Supported Environments

- **OS**: Linux (x86_64)
- **Python Version**: 3.8, 3.9, 3.10, 3.11, 3.12 (The installation script will automatically detect your version)

Integrated Package Installation

The provided install.sh script handles everything in one go, including Python version detection and package installation.

```
# Run the interactive installation script (recommended)
./dx-compiler/install.sh
```

C. Verify and Usage

After installation, activate the virtual environment (`venv-dx-compiler`) to verify the setup.

```
# 1. Activate the Virtual Environment
source ./dx-compiler/venv-dx-compiler/bin/activate

# 2. Verify Installed Version (CLI and Python Modules)
dxcom --version
python3 -c "import dx_com; print(dx_com.__version__)"

# 3. Access Help Documentation
dxcom -h
```

- **Sample Data Location**: `./dx-compiler/dx_com/sample_models/`

Copyright 2026 DEEPX All Rights Reserved.

> **Tip**
>
> If the automatic sample data download fails, you can manually fetch the assets using the following scripts:
> - `./dx-compiler/example/1-download_sample_models.sh` (Model data)
> - `./dx-compiler/example/2-download_sample_calibration_dataset.sh` (Calibration data)

D. DX-TRON (GUI Visualizer)

DX-TRON is a visual analysis tool for inspecting model structures and workload distribution. Choose the execution mode that fits your environment:

- **Local Execution (Desktop)**: Type `dxtron` in the terminal or execute the following script:

```
./dx-compiler/run_dxtron_appimage.sh
```

- **Web Server Execution (Remote/Docker)**: Run the web server script and specify a port:

```
./dx-compiler/run_dxtron_web.sh --port=8080
```

  Then, access **http://localhost:8080** in your browser.

- **Windows Users**: You can download the dedicated Windows installer directly from the **DEEPX Developer Portal**.

2.3.2 DX-Runtime Installation (RT, Driver, FW, App, Stream)

The **DX-Runtime** stack is the core software layer required to control DEEPX NPU hardware and execute AI applications. Each component is managed as a submodule within the `./dx-runtime` directory.

A. Building and Installing Modules

Depending on your requirements, you can perform a full installation or target specific modules to save time.

```
# Option 1: Install all modules (Driver, FW, RT, App, Stream)
./dx-runtime/install.sh --all

# Option 2: Full install excluding firmware
# (Use this if your NPU already has the latest FW version)
```


Copyright 2026 DEEPX All Rights Reserved.

```
./dx-runtime/install.sh --all --exclude-fw

# Option 3: Install a specific module only
./dx-runtime/install.sh --target=<module_name>
```

B. Firmware (DX-FW) Update and Activation

Updating the firmware is a critical process. To ensure the hardware logic is correctly initialized, follow these steps precisely.

Step 1. Update the Firmware
You can update the firmware using the automated installation script or the dedicated CLI tool.

```
# Method 1. Using the installation script
./dx-runtime/install.sh --target=dx_fw

# Method 2. Manual update using dxrt-cli
dxrt-cli -u ./dx-runtime/dx_fw/m1/X.X.X/mdot2/fw.bin
```

Step 2. Perform a Cold Boot
It is **strongly recommended** to completely shut down the system, turn off the power, and then turn it back on. A simple 'Restart' may not be sufficient for hardware initialization.

Step 3. System Reboot
After installation is complete, be sure to perform `sudo reboot` to activate the installed kernel driver.

2.3.3 [Local] Installation Verification (Sanity Check)

Once the local installation is finished, perform a final check to confirm that the hardware and software are communicating correctly.

A. Hardware and Version Check

Run the following command to display information about the NPU devices recognized by the system.

```
dxrt-cli -s
```


Copyright 2026 DEEPX All Rights Reserved.

Success Checklist

- **[x] Device Recognition**: Does it display `Device 0: M1?`
- **[x] Version Info**: Do `RT Driver`, `PCIe Driver`, and `FW version` show valid numbers (e.g., v1.x.x)?
- **[x] Status**: Are real-time metrics for **Voltage**, **Clock**, and **Temperature** visible at the bottom?

B. System Integrity Check

Run the batch sanity script to verify that all modules are located in their designated paths.

```
./dx-runtime/scripts/sanity_check.sh
```

> **Tip**
>
> If any item returns a **FAIL** or **Not Found**, please revisit the module installation steps (Section 3-2) to ensure all components were compiled correctly.

Copyright © DEEPX. All rights reserved.

Copyright 2026 DEEPX All Rights Reserved.

3. Running Your First NPU Model

This guide covers the end-to-end practical process for running actual AI models on **DEEPX NPU** after completing all installations. By running the automation scripts in sequence, you can experience the entire pipeline from model compilation to successful inference.

3.1 Overall Pipeline

DEEPX NPU model deployment follows a structured 6-step workflow (Steps 0–5), from optimization (Compiler) to execution (Runtime).

Step	Task	Related Script	Remarks
Step 0	Environment Check	`compiler-0...sh`, `runtime-0...sh`	Verify SDK installation and NPU driver
Step 1	Model Download	`compiler-1_download_onnx.sh`	Download original trained models (ONNX)
Step 2	Data Preparation	`compiler-2_setup_calibration_dataset.sh`	Prepare reference dataset for quantization optimization
Step 3	Path Configuration	`compiler-3_setup_output_path.sh`	Configure storage path for compiled outputs (`.dxnn`)
Step 4	NPU Compilation	`compiler-4_model_compile.sh`	Generate hardware-optimized `.dxnn` binaries
Step 5	NPU Inference	`runtime-3_run_example_using_dxrt.sh`	Execute the model on NPU and verify results

Copyright 2026 DEEPX All Rights Reserved.

Directory Structure

Upon completion, the getting-started/ folder will be organized with symbolic links to actual data sources.

```
getting-started/
├── calibration_dataset             # dx-compiler/dx_com/calibration_dataset
├── dxnn                            # dx-compiler/dx_com/output
├── forked_dx_app_example           # Example execution target (forked)
│  ├── bin
│  │  ├── efficientnet_async
│  │  └── yolov5_async
│  │  └── yolov5face_async
│  └── sample
│     └── ILSVRC2012
└── sample_models                   # dx-compiler/dx_com/sample_models
   ├── json
   └── onnx
```

Preparation

Before you begin, ensure your environment meets the following criteria:

- **Prerequisite**: The DX-AllSuite installation **must** be fully completed.
- **Environment Selection**
  : **Local**: For users who have installed drivers and the SDK directly on the Host OS.
  : **Docker**: For users working inside the container provided by the DEEPX Docker image.
  Tip: All scripts **must** be executed inside the container (via docker exec).

3.2 DX-Compiler: AI Model Compilation Scripts Guide (Steps 0–4)

This section details the process of converting standard AI models into NPU-proprietary binaries (`.dxnn`) through a sequential workflow from Step 0 to Step 4.

3.2.1 A. Execution Order

The scripts **must** be executed in the following order within the getting-started directory.

```
./getting-started/compiler-0_install_dx-compiler.sh         # Environment Setup
./getting-started/compiler-1_download_onnx.sh               # Model Acquisition
./getting-started/compiler-2_setup_calibration_dataset.sh   # Data Preparation
./getting-started/compiler-3_setup_output_path.sh           # Path Configuration
./getting-started/compiler-4_model_compile.sh               # Final Compilation
```


Copyright 2026 DEEPX All Rights Reserved.

3.2.2 B. Common Principles & Tips

All compiler scripts follow a "smart delegation" design to keep your workflow consistent.

- **Logic Delegation**: Actual processing is offloaded to the original scripts located in `dx-compiler/example/`.
- **Auto-Environment Detection**: The scripts automatically identify whether they are running in a **Host** or **Docker** environment to configure paths and symbolic links.
- **Final Asset**: Every step is a milestone toward producing the `.dxnn` file, which is the essential asset of your NPU inference.

3.2.3 C. Detailed Script Guide

[Step 0] `compiler-0_install_dx-compiler.sh` (Package Installation)
Sets up the environment for the model compiler (`dxcom`).

- **Core Function**: Installs the `dx-compiler` environment and runs a quick health check.
- **Smart Check**: It probes for existing installations using `dxcom -v`. If it's already there, it won't waste your time with a redundant setup (unless you use the `--force` flag).

[Step 1] `compiler-1_download_onnx.sh` (Model Download)
Prepares the sample model files (`.onnx` and `.json`) for compilation.

- **Supported Models**: `YOLOV5S-1, YOLOV5S_Face-1, MobileNetV2-1`
- **Mapping**: Creates a symbolic link between `getting-started/sample_models` and the SDK's internal `dx-compiler/dx_com/sample_models`.

[Step 2] `compiler-2_setup_calibration_dataset.sh` (Calibration Data Setup)
Acquires a reference dataset to maintain accuracy during the model quantization process.

- **Purpose**: Downloads and extracts representative image datasets. These are used to calibrate the model so that accuracy remains high even after optimization.
- **Result**: Links your local `getting-started/calibration_dataset` to the SDK core.

Copyright 2026 DEEPX All Rights Reserved.

[Step 3] `compiler-3_setup_output_path.sh` **(Path Configuration)**

Environment	Physical Path (Actual Storage)	Logical Path (Access Point)
Docker	`${DOCKER_VOLUME_PATH}/dxnn`	`getting-started/dxnn/`
Local	`${DX_AS_PATH}/workspace/dxnn`	`getting-started/dxnn/`

> **Tip**
>
> This "**Logical Path**" abstraction allows subsequent runtime scripts to look for models in the same place (`dxnn/`) regardless of whether you're running on a server or a local PC.

[Step 4] `compiler-4_model_compile.sh` **(Execute Model Compilation)**
The final stage where the prepared model and data are combined to generate the NPU-optimized binary.

- **Core Action**: Invokes the `dxcom` engine to perform the following fusion:

  ONNX (The Structure) + **JSON** (The Config) + **Calibration** (The Precision) = **.dxnn** (The Optimized Binary)

- **Output**: Your shiny new `.dxnn` files are stored in `dx-compiler/dx_com/output/` and are immediately accessible via the `getting-started/dxnn` link.

3.3 DX-Runtime: Application Execution (Application Execution Scripts Guide) (Step 0, 5)

This chapter focuses on Step 0 (Environment Setup) and Step 5 (Inference), where you deploy and run your optimized models on the actual DEEPX NPU hardware.

Copyright 2026 DEEPX All Rights Reserved.

3.3.1 A. Full Execution Order

To ensure stable operation, execute these scripts sequentially from within the `getting-started/` directory.

```
./getting-started/runtime-0_install_dx-runtime.sh # Set up environment
./getting-started/runtime-1_setup_input_path.sh # Connect model
./getting-started/runtime-2_setup_assets.sh # Prepare resources
./getting-started/runtime-3_run_example_using_dxrt.sh # Run example
```

3.3.2 B. Common Principles & Tips

All runtime scripts are engineered with the following logic to ensure a seamless deployment experience.

- **Intelligent Diagnosis & Recovery**: Scripts automatically verify the NPU driver and installation integrity using `sanity_check.sh`. If an environment is corrupted, use `-f` or `--force` to re-initialize.
- **Environment-Adaptive Logic**: Paths are dynamically assigned based on your Host or Docker environment. In headless (TTY) environments, the scripts automatically apply the `--no-display` flag to prevent GUI errors.
- **Resource Delegation**: The scripts trigger `dx_app/setup.sh` to automatically populate the `forked_dx_app_example` folder with required binaries and sample data.
- **Visual Verification**:

  CLI Environment: Includes a `fim` (fbi improved) installation routine to view result images directly in the terminal.

  Docker Environment: Since GUI output is limited, results can be retrieved using `docker cp <container_id>:/path/to/result ./local_path`.

> ⚠ **Prerequisite**
>
> You **must** have successfully generated `.dxnn` files in the previous **Compiler** steps. The runtime engine cannot operate without these hardware-optimized binaries.

Copyright 2026 DEEPX All Rights Reserved.

3.3.3 C. Detailed Script Guide

[Step 0] `runtime-0_install_dx-runtime.sh` **(Package Installation)**
Builds the core software stack for DEEPX NPU control.

- **Key Functions**: Installs runtime libraries and verifies driver operation.
- **Optimization**: If `sanity_check.sh` detects that drivers are already correctly loaded, it skips redundant installation steps.
- **Time-Saving Option**: Use `--exclude-fw` to install the software stack while skipping firmware updates (useful if your firmware is already up to date).

[Preparation for Step 5] `runtime-1_setup_input_path.sh` **(Model Path Synchronization)**
Synchronizes paths so that the runtime engine can load the `.dxnn` models generated during the compilation phase.

Environment	Physical Path (Actual Storage)	Logical Path (Access Point)
Docker	`${DOCKER_VOLUME_PATH}/dxnn`	`getting-started/dxnn/`
Local	`${DX_AS_PATH}/workspace/dxnn`	`getting-started/dxnn/`

[Preparation for Step 5] `runtime-2_setup_assets.sh` **(Resource Preparation)**
Prepare the dependency files and sample data required to run the inference example.

- **Action**: Sequentially calls the `setup.sh` scripts within the `dx_app` and `dx_stream` directories.
- **Note**: This script manages resources within their respective modules; it does not create redundant copies in the `forked_dx_app_example` folder.

Copyright 2026 DEEPX All Rights Reserved.

[Step 5] `runtime-3_run_example_using_dxrt.sh` **(Execute Example and Check Results)**
This script loads your models onto the NPU, measures real-world performance, and generates visualized outputs across three core tasks.

Model	Input Data	Task	Iterations
YOLOV5S_Face-1	`face_sample.jpg`	[1] Face Detection	30
YOLOV5S-1	`1.jpg`	[2] Object Detection	300
MobileNetV2-1	`1.jpeg`	[3] Image Classification	300

- **Result Report**: After execution is complete, the script outputs average **FPS** (Frames Per Second) and **Latency** (ms) to the terminal.

> ⚠ **Essential Check**
>
> This pipeline assumes all three sample models from Step 1 were successfully compiled in Step 4.
> - If a model is missing, you will encounter a "**File Not Found**" error during inference.
> - We strongly recommend compiling **all 3 models** before running this runtime script to verify the entire end-to-end process.

Copyright © DEEPX. All rights reserved.

Copyright 2026 DEEPX All Rights Reserved.

4. Version Compatibility

Since the **DEEPX SDK** maintains strict version dependencies between modules, it is essential to consult the compatibility matrix below when performing updates. **DX-AllSuite** ensures that all components are integrated and managed in validated, optimal combinations.

Copyright 2026 DEEPX All Rights Reserved.

4.1 DXNN SDK Version Compatibility Matrix

Copyright 2026 DEEPX All Rights Reserved.

Release Date	DX-AllSuite						
	DX-Compiler		DX-Runtime				
	DX-COM	DX-TRON	DX-FW	NPU Driver	DX-RT	DX-Stream	DX-APP
2026-04-10	v2.3.0						
	v2.3.0		v2.3.0				
	v2.3.0	v2.0.1	v2.5.6	v2.4.0	v3.3.0	v3.0.0	v3.1.0
2026-02-26	v2.2.2						
	v2.2.1		v2.2.2				
	v2.2.1	v2.0.1	v2.5.0	v2.1.0	v3.2.0	v2.2.1	v3.0.2
2026-02-09	v2.2.1						
	v2.2.0		v2.2.1				
	v2.2.0	v2.0.1	v2.5.0	v2.1.0	v3.2.0	v2.2.0	v3.0.1
2026-01-16	v2.2.0						
	v2.2.0		v2.2.0				
	v2.2.0	v2.0.1	v2.5.0	v2.1.0	v3.2.0	v2.2.0	v3.0.0
2025-11-28	v2.1.0						
	v2.1.0		v2.1.0				
	v2.1.0	v2.0.0	v2.4.0	v1.8.0	v3.1.0	v2.1.0	v2.1.0
2025-09-08	v2.0.0						
	v2.0.0		v2.0.0				
	v2.0.0	v2.0.0	v2.1.4	v1.7.1	v3.0.0	v2.0.0	v2.0.0
	v1.0.0						

Copyright 2026 DEEPX All Rights Reserved.

Release Date	DX-AllSuite						
	DX-Compiler		DX-Runtime				
	DX-COM	DX-TRON	DX-FW	NPU Driver	DX-RT	DX-Stream	DX-APP
2025-07-23	v1.0.0		v1.0.0				
	v1.6.0.1	v0.0.8	v2.1.0	v1.5.0	v2.9.5	v1.7.0	v1.1 1.0

4.2 How to Check Your System Version

To ensure system stability, identify the versions of your currently installed components and compare them with the **Version Compatibility Matrix** provided above.

Step 1. Hardware & Drivers (Target Side)
You can immediately verify the **Firmware (FW)** and **NPU Driver** versions using the NPU status utility.

```
# Display NPU status and driver information
dxrt-cli -s
```

Step 2. Compiler (Host Side)
Verify the version of the model conversion tool (`dxcom`). Note that the virtual environment **must** be activated for the command to be recognized.

```
# Execute within an activated virtual environment
dxcom -v
```

Alternative (via Python)
If you are using the compiler as a library, you can also check the version directly via Python.

```
python3 -c "import dx_com; print(dx_com.__version__)"
```

Copyright © DEEPX. All rights reserved.

Copyright 2026 DEEPX All Rights Reserved.

5. FAQ Troubleshooting Guide

This guide addresses common errors and symptoms encountered while using the **DXNN SDK** and provides step-by-step solutions.

Table of Contents

- Q1. Container 'Restarting' Error (#dxrtd-conflict)
- Q2. X11 Session Warnings & Mount Errors (Wayland Issues)
- Q3. Firmware Version Mismatch Error
- Q4. Device Driver Update Error
- Q5. Model-Runtime Version Compatibility Error

5.1 Q1. Container 'Restarting' Error (`dxrtd` Conflict)

This issue occurs when the container status is continuously displayed as Restarting after running docker_run.sh, making it impossible to enter the container.

5.1.1 Diagnostic Steps

- **Step 1. Check Status**: Run `docker ps` and check if the `STATUS` is repeatedly showing `Restarting (255)`.
- **Step 2. Check Logs**: Run `docker logs <container_name>` and check for the message "`Other instance of dxrtd is running`".

```
# Check container status
docker ps

# Example Output: STATUS repeats 'Restarting (255)'
CONTAINER ID   IMAGE             COMMAND
STATUS                          NAMES
041b9a4933e3   dx-runtime:24.04  "/usr/local/bin/dxrtd"  Restarting (255) 4 seconds
ago  dx-runtime-24.04

# Check container logs
docker logs dx-runtime-24.04
# Output: "Other instance of dxrtd is running"
```


Copyright 2026 DEEPX All Rights Reserved.

5.1.2 Cause

The DEEPX Runtime Daemon (`dxrtd`) is designed as a **singleton**. Only one instance can run on the entire system (Host + all Containers). If `dxrtd` is already running on the host, the daemon inside the container will fail to initialize, causing a crash loop.

5.1.3 Solution: Stopping the Host Runtime Service (`dxrtd`)

Method 1: Stop the Host Service (Recommended)
Stop the service on the host to allow the containerized daemon to gain control over the NPU.

```
# Stop the host service
sudo systemctl stop dxrt.service

# Re-run the container
./docker_run.sh --target=dx-runtime --ubuntu_version=24.04
```

For more details, refer to the **Host System Preparation** in **02. Setting Up Environment**.

Method 2: Prevent Auto-execution in the Container
If you **must** keep the host service running, configure the container so that the daemon (`dxrtd`) does not start automatically upon launch.

For more details, refer to the **Docker Advanced Troubleshooting** in **02. Setting Up Environment**.

5.2 Q2. X11 Session Warnings & Mount Errors (Wayland Issues)

This occurs when X11 forwarding authentication fails, leading to GUI tool errors or container startup refusal.

5.2.1 Diagnostic Steps

- **Q2.1 Warning**: `[WARN] it is recommended to use an X11 session` appears during startup.
- **Q2.2 Error**: The container fails with `error mounting /tmp/.docker.xauth` after a system reboot or logout.

Copyright 2026 DEEPX All Rights Reserved.

5.2.2 Cause

- **Q2.1:** GUI tools such as **DX-TRON** are optimized for X11. Wayland sessions can cause `xauth` processes to be unstable.
- **Q2.2:** Ending a Wayland session may purge X authentication data or change directory paths, causing Docker mount points to become invalid.

5.2.3 Solution: Set the Default System Session to X11

The most reliable solution is to set the login session to X11 (`Xorg`). This is the standard procedure for Ubuntu/GNOME environments.

Step 1. Modify GDM Configuration
Open the GDM3 configuration file.

```
sudo nano /etc/gdm3/custom.conf
```

Uncomment `WaylandEnable=false` to force the login screen to use Xorg.

```
# Force the login screen to use Xorg
WaylandEnable=false
```

Step 2. Apply Settings
Restart GDM or reboot your system.

```
sudo systemctl restart gdm3
# Or simply reboot the PC
```

Step 3. Resource Cleanup
Remove orphan containers before re-running.

```
# Clean up existing containers
docker compose -f docker/docker-compose.yml down --remove-orphans

# Re-run the container
./docker_run.sh --target=dx-runtime --ubuntu_version=24.04
```

For more details, refer to the **Docker Installation** section in **02. Setting Up Environment**.

Copyright 2026 DEEPX All Rights Reserved.

5.3 Q3. Firmware Version Mismatch Error

This error occurs when the application stops because the firmware on the NPU hardware is older than the required software stack version.

5.3.1 Diagnostic Steps

Check the error message in your terminal.

```
The current firmware version is X.X.X.
Please update your firmware to version Y.Y.Y or higher.
```

5.3.2 Cause

Mismatch between the installed **DX-RT** (Runtime) library and the **DX-FW** (Firmware) flashed onto the NPU.

5.3.3 Solution: Updating Firmware (DX-FW) and Cold Booting

Method 1: Using the Integrated Installation Script (Recommended)
Update the firmware using the all-in-one setup script.

```
./dx-runtime/install.sh --target=dx_fw
```

Method 2: Using the Dedicated CLI Tool (`dxrt-cli`)
Update the firmware manually by pointing to the specific binary file.

```
dxrt-cli -u ./dx-runtime/dx_fw/m1/X.X.X/mdot2/fw.bin
```

Critical Post-Update Action: Cold Booting
To completely initialize the hardware logic and ensure the new firmware version is applied, please follow this procedure.

Option 1. [Recommended] Cold Boot
- Method: Shut down the system completely, **unplug the power cable** to drain any residual power, then plug it back in and turn it on.
- Reason: This is the most reliable way to guarantee a complete hardware-level reset for the NPU.

Copyright 2026 DEEPX All Rights Reserved.

Option 2. For Remote/SSH Environments
- If physical power disconnection is impossible (e.g., in a remote server room), perform a system restart at the OS level using the `sudo reboot` command. In most standard scenarios, a system reboot is sufficient to refresh the firmware.

Final Verification
After rebooting, enter the following command in the terminal to verify that the firmware has been updated successfully.

```
dxrt-cli -s
```

For more details, refer to the **Firmware (DX-FW) Update and Activation** section in **02. Setting Up Environment**.

5.4 Q4. Device Driver Update Error

This issue occurs when the application execution is interrupted because the kernel driver version is outdated.

5.4.1 Diagnostic Steps

Check the terminal for the following error message.

```
The current device driver version is X.X.X.
Please update your device driver to version Y.Y.Y or higher.
```

5.4.2 Cause

The installed kernel driver (`dx_rt_npu_linux_driver`) does not meet the minimum requirements of the current **DX-RT**.

5.4.3 Solution: Update the Device Driver on the Host OS

To resolve this, you **must** update the driver module directly on your host operating system.

Copyright 2026 DEEPX All Rights Reserved.

Step 1. Install the Driver Module
Run the installation script specifically targeting the driver.

```
./dx-runtime/install.sh --target=dx_rt_npu_linux_driver
```

Step 2. System Reboot
Since the driver needs to be reloaded into the Linux kernel, a system reboot is mandatory.

```
sudo reboot
```

Step 3. Verify Status
After the reboot, confirm the update was successful.

```
dxrt-cli -s
```

> **Warning for Docker Users**
>
> Because Docker containers share the host's kernel, **driver updates must be performed on the Host OS**, not inside the container. Updating the driver inside a container will not affect the hardware communication layer.

For more details, refer to the **Module Build and Installation** section in **02. Setting Up Environment**.

5.5 Q5. Model-Runtime Version Compatibility Error

This error occurs when the application stops because of the incompatibility between the compiler version (used to create the `.dxnn` file) and the runtime library version.

5.5.1 Diagnostic Steps

Check the terminal for the following error message.

```
The model's compiler version(X.X.X) is not compatible in this RT library.
Please downgrade the RT library version to X.X.X or use a model file generated with a
compiler version X.X.X or higher.
```


Copyright 2026 DEEPX All Rights Reserved.

5.5.2 Cause

This happens when there is a version mismatch between the **DX-COM** (Compiler) and the **DX-RT** (Runtime).

5.5.3 Solution: Synchronizing Versions via Re-compilation

To resolve this, you **must** ensure that your model and runtime environments are using compatible versions.

Method 1: Re-compile the Model (Recommended)
Re-generate the `.dxnn` file using a compiler version that is compatible with your current system's **DX-RT** version. This is the safest way to ensure optimal performance and feature support.

Method 2: Adjust the Runtime Library Version
Reinstall **DX-RT** to a version that matches the model's requirements.

> **Note**
>
> If you choose this method, you **must** also re-verify the compatibility of your NPU Driver and Firmware.

Version Verification Guide
To find the exact compatible combinations for each module, please refer to the **DXNN SDK Component Version Compatibility Matrix**.

Copyright © DEEPX. All rights reserved.

Copyright 2026 DEEPX All Rights Reserved.