

# **DX-AS (DEEPX All Suite)**

## **User Guide**

---



**2025/05/14**

*DEEPX.ai*

© Copyright 2025- DEEPX All Rights Reserved.

# Table of contents

---

1. Introduction	3
1.1 Overview	3
1.2 Installation Guide	6
1.3 Getting Started	6
1.4 Version Compatibility	6
1.5 FAQ	6
2. Installation Guide	7
2.1 Prerequisites	7
2.2 Local Installation	8
2.3 Installation Using Docker	11
2.4 Run Sample Application	15
2.5 Run DX-Compiler	16
3. Getting-Start	18
3.1 Overall	18
3.2 Preparation	18
3.3  DX-Compiler: AI Model Compilation Scripts Guide	19
3.4  DX-Compiler: Application Execution Scripts Guide	22
4. Version Compatibility	25
4.1 DXNN SDK Version	25
5. FAQ	26
5.1 Question #1	26
5.2 Answer #1	26
5.3 Question #2	28
5.4 Answer #2	28
5.5 Question #3	29
5.6 Answer #3	29
5.7 Question #4	30
5.8 Answer #4	30
5.9 Question #5	30
5.10 Answer #5	30

# 1. Introduction

---

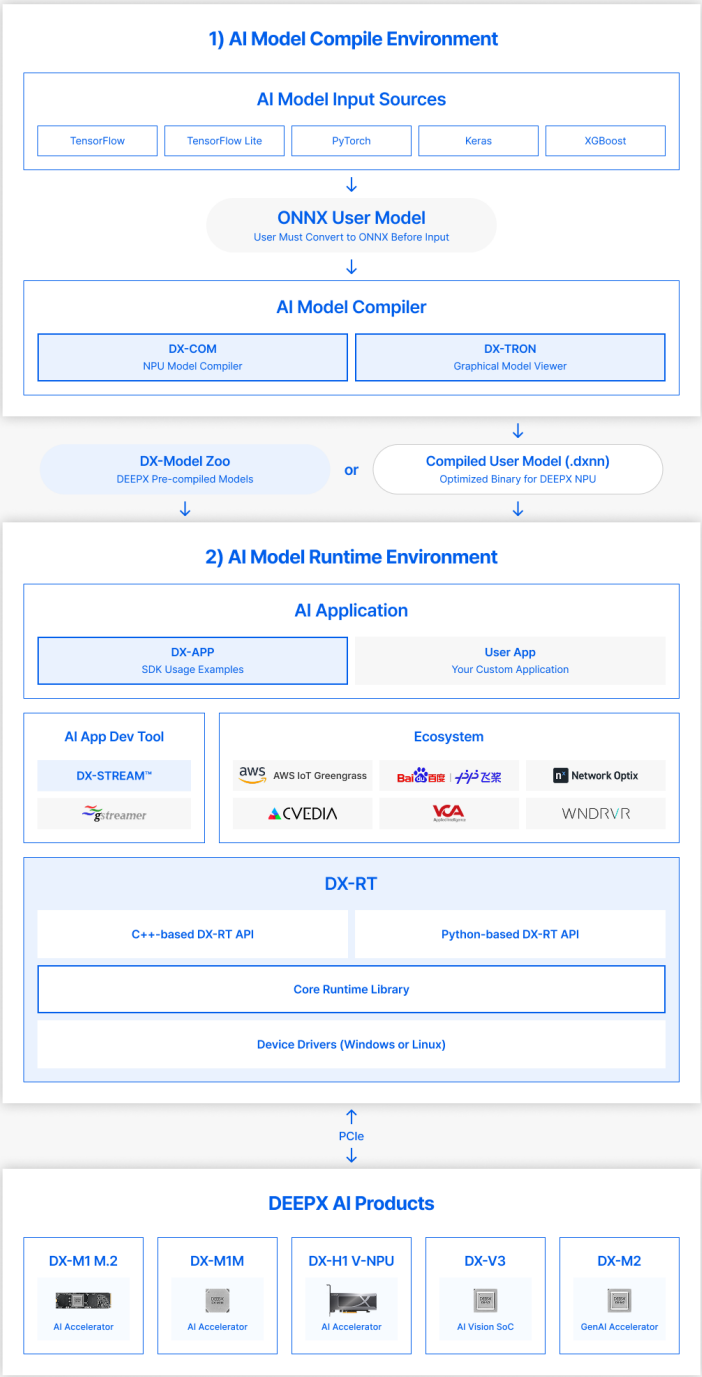
## 1.1 Overview

---

DX-AS (DEEPX All Suite) is an integrated environment of frameworks and tools that enables inference and compilation of AI models using DEEPX devices. Users can build the integrated environment by installing individual tools, but DX-AS maintains optimal compatibility by aligning the versions of the individual tools.

# DXNN® Full Stack Architecture

DXNN® (DEEPX Neural Network) SDK is a comprehensive software development environment that streamlines the entire AI deployment pipeline on DEEPX Neural Processing Units (NPUs). DXNN SDK integrates essential tools and frameworks for model compilation, optimization, simulation, and inference, all version-aligned to ensure seamless compatibility. While each tool can be installed separately, DEEPX offers a fully integrated ready-to-use environment optimized package for fast and efficient AI development and deployment, delivered as the DX-AS (All Suite).



### DX-COM

DX-COM is the compiler in the DEEPX SDK that generates a hardware-optimized .dxnn binary from an ONNX model and JSON configuration, enabling low-latency, high-efficiency inference on DEEPX NPUs.

### DX-TRON

DX-TRON is a graphical model viewer for inspecting .dxnn files generated by the DEEPX toolchain. It visualizes the model's layer graph and highlights how workloads are divided between the NPU and CPU.

### DX-Model Zoo

DX-Model Zoo is a curated library of pre-trained models with ONNX files, JSON files, and DXNN binaries, allowing developers to rapidly test and deploy applications.

### DX-APP

DX-APP is a sample application that demonstrates how to run compiled models on DEEPX NPU using DX-RT, providing ready-to-use code for vision tasks and serving as a template for custom AI application development.

### DX-STREAM

DX-STREAM is a custom GStreamer plugin that enables real-time streaming AI inference on DEEPX NPU, featuring modular components for preprocessing, inference, and postprocessing in vision AI applications.

### DX-RT

DX-RT(Runtime) directly interacts with the DEEPX NPU through firmware and device drivers, using PCIe interface for high-speed data transfer between the host and the NPU, and provides C/C++ and Python APIs for application-level inference control.

## 1.1.1 Components

---

### **DX-COM**

DEEPX NPU Compiler (DX-COM) produces the NPU Command Set from the provided ONNX and configuration file. The output file(graph.dxnn), which includes the generated command set and weights, is utilized for operating the NPU.

### **DX-RT**

DX-RT is the DEEPX Runtime SDK for AI inference using DEEPX devices. It supports pre-built models from the DEEPX model zoo and models compiled by DXCOM (the DEEPX Compiler SDK).

DX-RT provides C/C++ APIs, allowing users to develop applications by calling these APIs. It also offers a Python wrapper, enabling users to develop applications using Python scripts.

DX-RT includes command-line interface (CLI) commands for checking model information, running benchmarks with dummy files, and monitoring the status of the NPU.

DX-RT uses the DX NPU Driver to send input data to the NPU via the PCIe interface and receive the model inference results.

### **DX-ModelZoo**

DEEPX aims to provide developers with an effortless experience using DeepX NPUs through DeepX Open Modelzoo, which offers a variety of neural network models supported by DeepX NPUs.

All featured models are provided with pre-trained ONNX models, configuration json files, and pre-compiled binaries named DXNN(DeepX Neural Network).

Developers can also compile featured ONNXs to DXNN, which enables rapid application development accelerated by DeepX NPUs. Comprehensive performance benchmark tools are available for comparing models of quantized INT8 on DeepX NPUs with full precision FP32 on GPUs or CPUs.

### **DX-APP**

As an example of using the DX-RT API, the User Application Example allows users to easily set up the runtime environment and quickly see a demonstration. DX-APP enables rapid demonstrations of deep learning model inference for vision tasks such as object detection, face detection, and classification, utilizing the DEEPX NPU. Users can refer to the DX-APP code as a guide for developing their own applications.

### **DX-Stream**

DX-Stream is a GStreamer Custom Plugin designed to simplify the development of Vision AI Applications using DEEPX's NPU in a GStreamer pipeline format. DX-Stream enables rapid and efficient development of Vision AI Applications by providing a modular, pipeline-based framework powered by DEEPX's NPU. The customizable preprocessing, inference, and postprocessing elements ensure flexibility for a variety of AI scenarios.

## 1.2 Installation Guide

---

[Link](#)

## 1.3 Getting Started

---

[Link](#)

## 1.4 Version Compatibility

---

[Link](#)

## 1.5 FAQ

---

[Link](#)

## 2. Installation Guide

---

DX-All-Suite is a tool for creating an environment to validate and utilize DEEPX devices. DX-All-Suite provides the following below methods for setting up the integrated environment:

**Install on local machine** - Set up the DX-All-Suite environment directly on the host environment (maintaining compatibility between the individual tools).

**Build Docker image and run container** - Build a DX-All-Suite environment within a Docker environment, or load a pre-built image to create a container.

### 2.1 Prerequisites

---

#### 2.1.1 Clone the Main Repository

```
git clone --recurse-submodules https://github.com/DEEPX-AI/dx-all-suite.git
```

or

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

#### (Optional) Initialize and Update Submodules in an Already Cloned Repository

```
git submodule update --init --recursive
```

#### 2.1.2 Check Submodule Status

```
git submodule status
```

#### (Optional) Install Docker & Docker Compose

```
./scripts/install_docker.sh
```

## Python Virtual Environment (Recommended)

The installation scripts will install Python packages via pip. To avoid conflicts with system packages and maintain a clean environment, it is strongly recommended to use a Python virtual environment:

```
# Create a virtual environment
python -m venv dx-venv

# Activate the virtual environment
source dx-venv/bin/activate # On Linux/macOS
# or
dx-venv\Scripts\activate # On Windows

# Verify the virtual environment is active
which python
# Should show: /path/to/dx-venv/bin/python
```

After activating the virtual environment, proceed with the installation steps below.

---

## 2.2 Local Installation

---

### 2.2.1 Install DX-Compiler Environment (dx\_com)

---

The `DX-Compiler` environment provides prebuilt binary outputs and does not include source code. Each module can be downloaded and installed from a remote server using the following command:

```
./dx-compiler/install.sh
```



When executing the above command, **DEEPX Developers' Portal (<https://developer.deepx.ai>)** account authentication may be required to download and install the DX-Compiler modules. The script obtains authentication information based on the following priority:

### 1. Directly specify when executing the command (1st priority):

```
./dx-compiler/install.sh --username=<your_email> --password=<your_password>
```

### 2. Use environment variables (2nd priority):

```
export DX_USERNAME=<your_email>
export DX_PASSWORD=<your_password>
./dx-compiler/install.sh
```

Alternatively, you can add account information to `compiler.properties` as shown below, which will be injected as environment variables:

```
DX_USERNAME=<your_email>
DX_PASSWORD=<your_password>
```

### 3. Prompt for input (3rd priority):

If neither of the above methods is used, the script will prompt you to enter your account information directly in the terminal during execution.

Upon successful installation:

#### 1. The `dx-com` module archive files ( `.tar.gz` ) will be downloaded and saved to:

- `./workspace/release/dx_com/download/dx_com_M1A_v[VERSION].tar.gz`

#### 2. The downloaded modules will be extracted to:

- `./workspace/release/dx_com/dx_com_M1A_v[VERSION]`
- Symbolic links will also be created at `./dx-compiler/dx-com`.

## Archive Mode (`--archive_mode=y`)

The `--archive_mode=y` option is primarily used when building Docker images for the `dx-compiler` environment with `docker_build.sh`. When this mode is activated, only the download of the module's `.tar.gz` file proceeds; no extraction or symbolic link creation is performed.

```
./dx-compiler/install.sh --archive_mode=y
```

When executing the above command, the module archive files ( `*.tar.gz` ) will be downloaded and saved to:

- `../archives/dx_com_M1A_v[VERSION].tar.gz`

These archive files can then be utilized by the Docker image build process.

## 2.2.2 Install DX-Runtime Environment

The `DX-Runtime` environment includes source code for each module. The repositories are managed as Git submodules( `dx_rt_npu_linux_driver` , `dx_rt` , `dx_app` , and `dx_stream` ) under `./dx-runtime` . To build and install all modules, run:

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

This command will build and install the following modules:

`dx_fw` , `dx_rt_npu_linux_driver` , `dx_rt` , `dx_app` , and `dx_stream`

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

You can exclude `dx_fw` from the installation using the `--exclude-fw` option.

### Selective Installation of a Specific Module

You can install a specific module using:

```
./dx-runtime/install.sh --target=<module_name>
```

### Update `dx_fw` (Firmware Image)

The `dx_fw` module does not include source code but provides a `fw.bin` image file.

To update the firmware using `dxrt-cli` , run:

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

Alternatively, you can use:

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

## Sanity check

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

You can use this command to verify that `dx_rt` and `dx_rt_npu_linux_driver` are installed correctly.

**It is recommended to completely shut down and power off the system before rebooting after a firmware update.**

## 2.3 Installation Using Docker

### 2.3.1 Install DX-Compiler, DX-Runtime, and DX-ModelZoo Environment

#### Notes

**1. WHEN USING A DOCKER ENVIRONMENT, THE NPU DRIVER MUST BE INSTALLED ON THE HOST SYSTEM:**

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

**2. IF `dx_rt` IS ALREADY INSTALLED ON THE HOST SYSTEM AND THE `service daemon ( /usr/local/bin/dxrttd )` IS RUNNING, LAUNCHING THE `DX-Runtime` DOCKER CONTAINER WILL RESULT IN AN ERROR (Other instance of `dxrttd` is running ) AND AUTOMATIC TERMINATION.**

Before starting the container, stop the service daemon on the host system.

**3. IF ANOTHER CONTAINER IS ALREADY RUNNING WITH THE `service daemon ( /usr/local/bin/dxrttd )`, STARTING A NEW CONTAINER WILL ALSO RESULT IN THE SAME ERROR.**

To run multiple DX-Runtime containers simultaneously, refer to note [#4](#)

**4. IF YOU PREFER TO USE THE `dxrttd` (SERVICE DAEMON) RUNNING ON THE HOST SYSTEM INSTEAD OF INSIDE THE CONTAINER,**

You can configure this in two ways:

**Solution 1: Modify at Docker image build level**

Update the `docker/Dockerfile.dx-runtime` as follows:

- Before:

```
...
ENTRYPOINT [ "/usr/local/bin/dxrttd" ]
# ENTRYPOINT [ "tail", "-f", "/dev/null" ]
```

• After:

```
...
# ENTRYPOINT [ "/usr/local/bin/dxrttd" ]
ENTRYPOINT [ "tail", "-f", "/dev/null" ]
```

### Solution 2) Modify at Docker container run level

Update the `docker/docker-compose.yml` as follows:

• Before:

```
...
dx-runtime:
  container_name: dx-runtime-${UBUNTU_VERSION}
  image: dx-runtime:${UBUNTU_VERSION}
  ...
  restart: on-failure
  devices:
    - "/dev:/dev" # NPU / GPU / USB CAM
```

• After:

```
...
dx-runtime:
  container_name: dx-runtime-${UBUNTU_VERSION}
  image: dx-runtime:${UBUNTU_VERSION}
  ...
  restart: on-failure
  devices:
    - "/dev:/dev" # NPU / GPU / USB CAM

  entrypoint: ["/bin/sh", "-c"] # ADDED
  command: ["sleep infinity"] # ADDED
```

## Build the Docker Image

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

This command builds a Docker image with both `dx-compiler`, `dx-runtime` and `dx-modelzoo` environments.

You can check the built images using:

```
docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
dx-runtime	24.04	05127c0813dc	41 hours ago	4.8GB
dx-compiler	24.04	b08c7e39e89f	42 hours ago	7.08GB
dx-modelzoo	24.04	cb2a92323b41	2 weeks ago	2.11GB

#### SELECTIVE DOCKER IMAGE BUILD FOR A SPECIFIC ENVIRONMENT

```
./docker_build.sh --target=dx-runtime --ubuntu_version=24.04
```

```
./docker_build.sh --target=dx-compiler --ubuntu_version=24.04
```

```
./docker_build.sh --target=dx-modelzoo --ubuntu_version=24.04
```

Use the `--target=<environment_name>` option to build only `dx-runtime` or `dx-compiler` or `dx-modelzoo`.

## Run the Docker Container

**(Optional) If `dx_rt` is already installed on the host system, please stop the `dxrt` service daemon before running the Docker container.**

(Reason: If the `dxrt` service daemon is already running on the host or in another container, the `dx-runtime` container will not be able to start. Only one instance of the service daemon can run at a time, including both host and container environments.)

(Refer to note #4 for more details.)

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

#### RUN A DOCKER CONTAINER WITH ALL ENVIRONMENTS (`dx_compiler`, `dx_runtime` AND `dx_modelzoo`)

```
./docker_run.sh --all --ubuntu_version=<ubuntu_version>
```

To verify running containers:

```
docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
f040e793662b	dx-runtime:24.04	"/usr/local/bin/dxrtcd"	33 seconds ago
e93af235ceb1	dx-modelzoo:24.04	"/bin/sh -c 'sleep i...'"	42 hours ago
b3715d613434	dx-compiler:24.04	"tail -f /dev/null"	42 hours ago

#### ENTER THE CONTAINER

```
docker exec -it dx-runtime-<ubuntu_version> bash
```

```
docker exec -it dx-compiler-<ubuntu_version> bash
```

```
docker exec -it dx-modelzoo-<ubuntu_version> bash
```

This allows you to enter the `dx-compiler`, `dx-runtime` and `dx-modelzoo` environments via a bash shell.

#### CHECK DX-RUNTIME INSTALLATION INSIDE THE CONTAINER

```
dxrt-cli -s
```

Example output:

```
DXRT v2.6.3
=====
* Device 0: M1, Accelerator type
----- Version -----
* RT Driver version   : v1.3.1
* PCIe Driver version : v1.2.0
-----
* FW version          : v1.6.0
----- Device Info -----
* Memory : LPDDR5 5800 MHz, 3.92GiB
* Board  : M.2, Rev 10.0
* PCIe   : Gen3 X4 [02:00:00]

NPU 0: voltage 750 mV, clock 1000 MHz, temperature 29'C
NPU 1: voltage 750 mV, clock 1000 MHz, temperature 28'C
NPU 2: voltage 750 mV, clock 1000 MHz, temperature 28'C
```

```
DVFS Disabled
```

```
=====
```

---

## 2.4 Run Sample Application

---

### 2.4.1 dx\_app

---

#### Installation Path

##### 1. On the Host Environment:

```
cd ./dx-runtime/dx_app
```

##### 2. Inside the Docker Container:

```
docker exec -it dx-runtime-<ubuntu_version> bash
# cd /deepx/dx-runtime/dx_app
```

#### Setup Assets (Precompiled NPU Model and Sample Input Videos)

```
./setup.sh
```

#### Run dx\_app

```
./scripts/run_detector.sh
fim ./result-app1.jpg
```

For more details, refer to [dx-runtime/dx\\_app/README.md](#).

---

## 2.4.2 dx\_stream

---

### Installation Path

#### 1. On the Host Environment:

```
cd ./dx-runtime/dx_stream
```

#### 2. Inside the Docker Container:

```
docker exec -it dx-runtime-<ubuntu_version> bash  
cd /deepx/dx-runtime/dx_stream
```

### Setup Assets (Precompiled NPU Model and Sample Input Videos)

```
./setup.sh
```

### Run dx\_stream

```
./run_demo.sh
```

For more details, refer to [dx-runtime/dx\\_stream/README.md](#).

---

## 2.5 Run DX-Compiler

---

### 2.5.1 dx\_com

---

#### Installation Path

##### 1. On the Host Environment:

```
cd ./dx-compiler/dx_com
```





## 3. Getting-Start

---

### 3.1 Overall

---

#### Full Execution Order

```
# Compiler Steps
bash compiler-1_download_onnx.sh
bash compiler-2_setup_calibration_dataset.sh
bash compiler-3_setup_output_path.sh
bash compiler-4_model_compile.sh

# Runtime Steps
bash runtime-1_setup_input_path.sh
bash runtime-2_setup_assets.sh
bash runtime-3_run_example_using_dxrt.sh
```

#### Folder Structure (After Execution)

```
getting-start/
├── calibration_dataset
├── dxnn
├── forked_dx_app_example      # ← Model output symbolic link created by dx-compiler
│   └── example               # ← Example execution target (forked)
│       ├── bin
│       ├── run_classifier
│       ├── run_detector
│       └── sample
│           └── ILSVRC2012
├── modelzoo
│   ├── json
│   └── onnx
```

## 3.2 Preparation

---

### 3.2.1 DX-AS (DEEPX All Suite) Installation

---

Refer to <https://github.com/DEEPX-AI/dx-all-suite> to install **DXNN® - DEEPX NPU SDK** on your local system or inside a Docker container.

1. [Install on local machine](#)
2. [Build Docker image and run container](#)

## 3.3 DX-Compiler: AI Model Compilation Scripts Guide

This section explains the purpose and flow of each script from `compiler-1_download_onnx.sh` to `compiler-4_model_compile.sh`.

### Execution Order

```
./getting-start/compiler-1_download_onnx.sh
./getting-start/compiler-2_setup_calibration_dataset.sh
./getting-start/compiler-3_setup_output_path.sh
./getting-start/compiler-4_model_compile.sh
```

### Tip

- The `.dxnn` file is the final executable generated by `dx_com`.
- Each script can be run independently, but following the order above ensures the entire process works as intended.

### 3.3.1 1. compiler-1\_download\_onnx.sh

Downloads model files ( `.onnx` , `.json` ) and links them into the workspace.

- **Purpose:** Automatic model download
- **Description:**
  - Downloads files into `modelzoo/onnx` and `modelzoo/json` directories.
  - Supports models: `YOLOV5S-1` , `YOLOV5S_Face-1` , `MobileNetV2-1` .
  - `--force` option allows overwriting existing files.

### Key Functions

- `show_help([type], [message])`
- Displays usage and exits when invalid options are passed.
- Supports `--force` , `--help` .
- `download(model_name, ext_name)`
- `download(model_name, ext_name)`

- Downloads resources based on the given model name and extension.
  - Calls `get_resource.sh` and saves to `modelzoo/{ext_name}/{model_name}.{ext_name}`.
  - Also creates a symbolic link to the workspace (`workspace/modelzoo/`).
  - `main()`  
Iterates through model list and extension list, calling `download()`.
- 

### 3.3.2 📁 2. compiler-2\_setup\_calibration\_dataset.sh

---

Creates a symbolic link to the calibration dataset and modifies `.json` config files to use it.

- **Purpose:** Calibration dataset setup
- **Description:**
  - Creates symlink: `./calibration_dataset` → `dx_com/calibration_dataset`
  - Forcefully changes (hijacks) the `dataset_path` entry in `modelzoo/json/*.json` to `./calibration_dataset`.
  - Configured to use the sample calibration dataset included within `dx_com`.

#### 📌 Key Functions

- `make_symlink_calibration_dataset()`
  - Creates a symbolic link from `dx_com/calibration_dataset` to `./calibration_dataset`.
  - Recreates the link if the existing one is broken.
  - `hijack_dataset_path(model_name)`
  - Forcefully changes the `"dataset_path"` value in `json/{model_name}.json` to `./calibration_dataset`.
  - Backs up the original file as `.bak` and modifies the value using the `sed` command.
  - Outputs the `diff` showing changes before and after.
  - `main()`
  - Executes `make_symlink_calibration_dataset()`.
  - Runs `hijack_dataset_path()` for each of the three models.
-

### 3.3.3 📁 3. compiler-3\_setup\_output\_path.sh

Creates a symbolic link from `./dxnn` to the workspace output path.

- **Purpose:** Setup path for compiled model output
- **Description:**
  - In host mode: `./dxnn` → `workspace/dxnn`
  - In container mode: `./dxnn` → `${DOCKER_VOLUME_PATH}/dxnn`
  - Automatically detects environment and handles broken symlinks

#### 📌 Key Function

- `setup_compiled_model_path()`
- Checks if the environment is a container and determines the output path accordingly:
- Container: `${DOCKER_VOLUME_PATH}/dxnn`
- Host: `${DX_AS_PATH}/workspace/dxnn`
- Creates a symbolic link `./dxnn` pointing to the appropriate workspace directory.
- Handles recreation if the existing link is broken.

### 3.3.4 📁 4. compiler-4\_model\_compile.sh

Compiles `.onnx` files into `.dxnn` format using `dx_com`.

- **Purpose:** Run model compilation
- **Description:**
  - Uses `dx_com` to compile `onnx + json` → `dxnn`
  - Outputs result to `./dxnn/`

#### 📌 Key Functions

- `compile(model_name)`
- Runs `dx_com` to convert `.onnx + .json` into a `.dxnn` file.
- The output is saved in the `./dxnn` directory.
- Exits on failure.

- `main()`
- Iterates through the list of models and calls `compile()` for each.

---

## 3.4 DX-Runtime: Application Execution Scripts Guide

---

This document explains the role and execution flow of the scripts from

`runtime-1_setup_input_path.sh` to `runtime-3_run_example_using_dxrt.sh`.

It serves as a guide for running compiled `.dxnn` models from the `dx-compiler` in an actual runtime environment based on example executions.

### Runtime Execution Order

```
bash runtime-1_setup_input_path.sh
bash runtime-2_setup_assets.sh
bash runtime-3_run_example_using_dxrt.sh
```

#### Tip

- Run `runtime-*` scripts **after** `.dxnn` models have been compiled successfully.
- `fm` is a CLI tool for viewing image output. It will be automatically installed if missing.
- Before running examples, make sure to run `dx_app/setup.sh` to download required models and sample data.

---

### 3.4.1 1. runtime-1\_setup\_input\_path.sh

---

Creates a symbolic link to the compiled model directory for runtime access.

- **Function:** Set runtime model path
- **Description:**
  - Creates a symbolic link for `./dxnn` to point to `workspace/dxnn`.
  - Automatically detects and supports both host and Docker container environments.

#### Key Function

- `setup_compiled_model_path()`

- Automatically sets the path based on whether the environment is a container or not.
  - Container: `${DOCKER_VOLUME_PATH}/dxnn`
  - Host: `${DX_AS_PATH}/workspace/dxnn`
  - Links `./dxnn` to the respective workspace path (also restores broken symlinks).
- 

### 3.4.2 📁 2. runtime-2\_setup\_assets.sh

Prepares the configuration files and model resources for running examples.

- **Function:** Prepare configuration files and resources for running examples.
- **Description:**
  - Calls the `setup.sh` of `dx_app` and `dx_stream` to download and copy necessary resources for example execution.
  - Automatically prepares required models, configuration files, and sample images.

#### Key Function

- `setup_assets(target_path)`
  - Executes the `setup.sh` of each module (`dx_app`, `dx_stream`).
  - Internally copies or links sample images, JSON configurations, models, etc.
- 

### 3.4.3 📁 3. runtime-3\_run\_example\_using\_dxrt.sh

Runs the `.dxnn` model using `dx_app` examples and checks the results.

- **Function:** Run runtime examples (Object Detection, Face Detection, Classification).
- **Description:**
  - Forks the `dx_app` example into the `forked_dx_app_example` folder.
  - Hijacks the model path in the `.json` configuration file to the user's compiled model.
  - Executes `run_detector` and `run_classifier` binaries.
  - Checks image results (using `fim`) or logs.

## 📌 Key Functions

- `fork_examples()`
  - Copies `dx_app/bin` binaries, `example/*`, and `sample/*` resources.
  - Initializes Git and commits to track diffs.
  - `hijack_example(file_path, source_str, target_str, commit_msg)`
  - Replaces `"./assets/models/*.dxnn"` path in `.json` configuration file with the actual generated model path.
  - Shows the diff for verification.
  - `run_hijacked_example(exe, config, save_log)`
  - Runs the binary and checks the results.
  - Object/Face Detection: Displays the result image and checks with `fm`.
  - Classification: Logs results to the `result-app.log` file.
  - `main()`
  - Forks, hijacks, and runs the YOLOV5S\_Face, YOLOV5S, and MobileNetV2 models.
-



## 4. Version Compatibility

DEEPX software components have version compatibility dependencies between them. Therefore, when updating the version of a specific component, you must also update the versions of other components, referring to the table below. DX-All-Suite simplifies this version compatibility management by integrating compatible components into a unified environment.

### 4.1 DXNN SDK Version

Release Date	DX-All-Suite	DX-Compiler		DX-Runtime				
		DX-COM	DX-TRON	DX_FW	NPU Driver	DX-RT	DX-Stream	DX-APP
2025-09-08	v2.0.0	v2.0.0		v2.0.0				
		v2.0.0	v0.8	v2.1.4	v1.7.1	v3.0.0	v2.0.0	v2.0.0
2025-07-23	v1.0.0	v1.0.0		v1.0.0				
		v1.60.1	v0.8	v2.1.0	v1.5.0	v2.9.5	v1.7.0	v1.11.0

## 5. FAQ

### 5.1 Question #1

When running the `dx-runtime` or `dx-modelzoo` container using the `docker_run.sh` script, the container stays in a **Restarting** state, and you cannot execute `docker exec -it <container-name> bash`.

```
./docker_run.sh --target=dx-runtime --ubuntu_version=24.04
```

```
[INFO] UBUNTU_VERSION(24.04) is set.
[INFO] TARGET_ENV(dx-runtime) is set.
[INFO] XDG_SESSION_TYPE: x11
Installing dx-runtime
[INFO] XAUTHORITY(/run/user/1000/gdm/Xauthority) is set
docker compose -f docker/docker-compose.yml up -d --remove-orphans dx-runtime
[+] Running 1/1
 ✓ Container dx-runtime-24.04
Started
0.1s
```

```
docker exec -it dx-runtime-24.04 bash
```

```
Error response from daemon: No such container: dx-runtime-24.04
```

### 5.2 Answer #1

First, check if the **dxrtd** (dx-runtime service daemon) is already running on the host system:

```
ps aux | grep dxrtd
```

```
root      60451  0.0  0.0 253648  6956 ?        Ssl  10:52   0:00 **/usr/local/bin/
dxrtd**
```

If **dxrtd is already running on the host**, the attempt to run `dxrtd` inside the Docker container will fail, causing the container to enter a **restart loop**.

Check the container status with:

```
docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
041b9a4933e3	dx-runtime:24.04	"/usr/local/bin/dxrttd"	18 seconds ago
**Restarting (255) 4 seconds ago**			
		dx-runtime-24.04	

```
docker logs dx-runtime-24.04
```

```
Other instance of dxrttd is running
Other instance of dxrttd is running
Other instance of dxrttd is running
Other instance of dxrttd is running
Other instance of dxrttd is running
Other instance of dxrttd is running
Other instance of dxrttd is running
Other instance of dxrttd is running
Other instance of dxrttd is running
Other instance of dxrttd is running
```

## 5.2.1 Solution 1:

**Stop the `dxrttd` service running on the host**, then rerun the `docker_run.sh` script so that `dxrttd` runs **only inside the Docker container**.

Refer to the installation guide [Link](#) for more details.

```
sudo systemctl stop dxrt.service
./docker_run.sh --target=dx-runtime --ubuntu_version=24.04
```

## 5.2.2 Solution 2:

Keep `dxrttd` running **on the host**, and **prevent it from running inside the Docker container**.

Refer to this guide [Link](#) for configuration instructions.

## 5.3 Question #2

5.3.1 Q2.1 When running `docker_run.sh`, the following warning message may appear:

```
[WARN] it is recommended to use an **X11 session (with .Xauthority support)** when
working with the 'dx-all-suite' container.
```

5.3.2 Q2.2 After a system reboot or session logout, the container failed to restart due to the following issue: `error mounting /tmp/.docker.xauth`.

Example error message:

```
Error response from daemon: failed to create task for container: failed to create shim
task: OCI runtime create failed: runc create failed: unable to start container process:
error during container init: error mounting "/run/user/1000/.mutter-Xwaylandauth.N9UI62"
to rootfs at "/tmp/.docker.xauth": create mountpoint for /tmp/.docker.xauth mount:
cannot create subdirectories in "/var/lib/docker/
overlay2/00690e188f08ad4bad24fbc8786b00653e76b44f32d9b88b1ae5ed1e2d7654c8/merged/
tmp/.docker.xauth": not a directory: unknown: Are you trying to mount a directory onto a
file (or vice-versa)? Check if the specified host path exists and is the expected type
Error: failed to start containers: dx-runtime-22.04
```

## 5.4 Answer #2

The `dx-all-suite` Docker container uses **X11 forwarding** to display GUI windows when running sample applications or example code. For this, it relies on `xauth` to manage authentication.

However, if the user's host environment is not based on **X11 (with .Xauthority)** but instead uses **Xwayland** or similar, the `xauth` data may be lost after a system reboot or session logout. As a result, the authentication file mount between the host and the container may fail, making it impossible to restart or reuse the container.

Therefore, it is recommended to use an **X11 session (with .Xauthority support)** when working with the `dx-all-suite` container.

You can resolve this issue by setting **X11 as the default session**, as described in the instructions above.

## 5.4.1 Set X11 as the Default Session

---

To make X11 the default session (and disable Wayland), modify the GDM configuration file.

### For GNOME (Ubuntu-based systems):

1. Open the GDM configuration file with root permissions:

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

1. Find the following line and uncomment it (remove the #), or add it if it doesn't exist:

```
WaylandEnable=false
```

3. Save the file and exit (Ctrl+O, Enter, then Ctrl+X in nano).

1. Restart the GDM service to apply the changes:

```
sudo systemctl restart gdm3
```

After reboot or GDM restart, the system will use the X11 session by default instead of Wayland.

---

## 5.5 Question #3

---

When running the example code, the following error message appears:

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

## 5.6 Answer #3

---

**dx\_rt** depends on both **dx\_rt\_npu\_linux\_driver** and **dx\_fw**. To use the current version of **dx\_rt** properly, you need to update the **dx\_fw**.

Please refer to [Link](#) to update your **dx\_fw** and resolve the issue.

---

## 5.7 Question #4

---

When running the example code, the following error message appears:

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

## 5.8 Answer #4

---

**dx\_rt** depends on both **dx\_rt\_npu\_linux\_driver** and **dx\_fw**. To use the current version of **dx\_rt** properly, you need to update the **dx\_rt\_npu\_linux\_driver**.

Please refer to [Link](#) to update your **dx\_rt\_npu\_linux\_driver** and resolve the issue.

---

## 5.9 Question #5

---

When running the example code, the following error message appears:

```
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.
```

## 5.10 Answer #5

---

This error occurs due to an incompatibility between the **dx\_rt** runtime and the model file compiled with **dx\_com**.

To resolve the issue, either downgrade **dx\_rt** to a compatible version, or recompile the model file (*.dxnn*) *using a dx\_com version that matches your current dx\_rt\**.

Refer to [Link](#) for version compatibility details between modules.