

DXNN All Suite (DX-AllSuite) User Manual

2026/07/29 (v2.4.1)

DEEPX.ai

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Table of contents

1. Introduction 4
1.1 Overview 4
1.2 DX-AS (DEEPX All Suite) Environments and Installation Guide 7
1.3 Documentation 7
2. DEEPX Agent-Driven Development - dx-agent-dev (Beta) 9
2.1 Introduction 9
2.2 Showcases 9
2.3 Prerequisites 14
2.4 Architecture Overview 14
2.5 Available Agents and Skills 15
2.6 Supported AI Tools 18
2.7 Quick Start by Tool 23
2.8 End-to-End Scenarios 25
2.9 Cross-Project Routing 29
2.10 Sub-Project Guides 29
2.11 Internal Reference Documents 30
2.12 Output Isolation 30
2.13 Session Sentinels 31
3. Agent-Driven Development Showcases 33
3.1 dx-agent-dev Showcases 33
4. DX-AllSuite Architecture Overview 40
4.1 Why DX-AllSuite? 40
4.2 System Architecture & Core Modules 41
4.3 Technical Compatibility (Support Matrix) 42
4.4 DEEPX ModelZoo & Supported Tasks 44
5. Setting Up Environment 46
5.1 Prerequisites 46
5.2 Docker Installation 50
5.3 Local Installation 55
6. Running Your First NPU Model 60
6.1 Overall Pipeline 60
6.2 DX-Compiler: AI Model Compilation Scripts Guide (Steps 0–4) 61
6.3 DX-Runtime: Application Execution Scripts Guide (Step 0, 5) 63
7. Version Compatibility 66
7.1 DXNN SDK Version Compatibility Matrix 66

ⓒ Copyright 2026 DEEPX All Rights Reserved.

7.2 How to Check Your System Version 68
8. FAQ Troubleshooting Guide 69
8.1 Q1. Container 'Restarting' Error (dxrtd Conflict) 69
8.2 Q2. X11 Session Warnings & Mount Errors (Wayland Issues) 70
8.3 Q3. Firmware Version Mismatch Error 71
8.4 Q4. Device Driver Update Error 73
8.5 Q5. Model-Runtime Version Compatibility Error 74
8.6 Q6. Insufficient Shared Memory (shm) Error 75

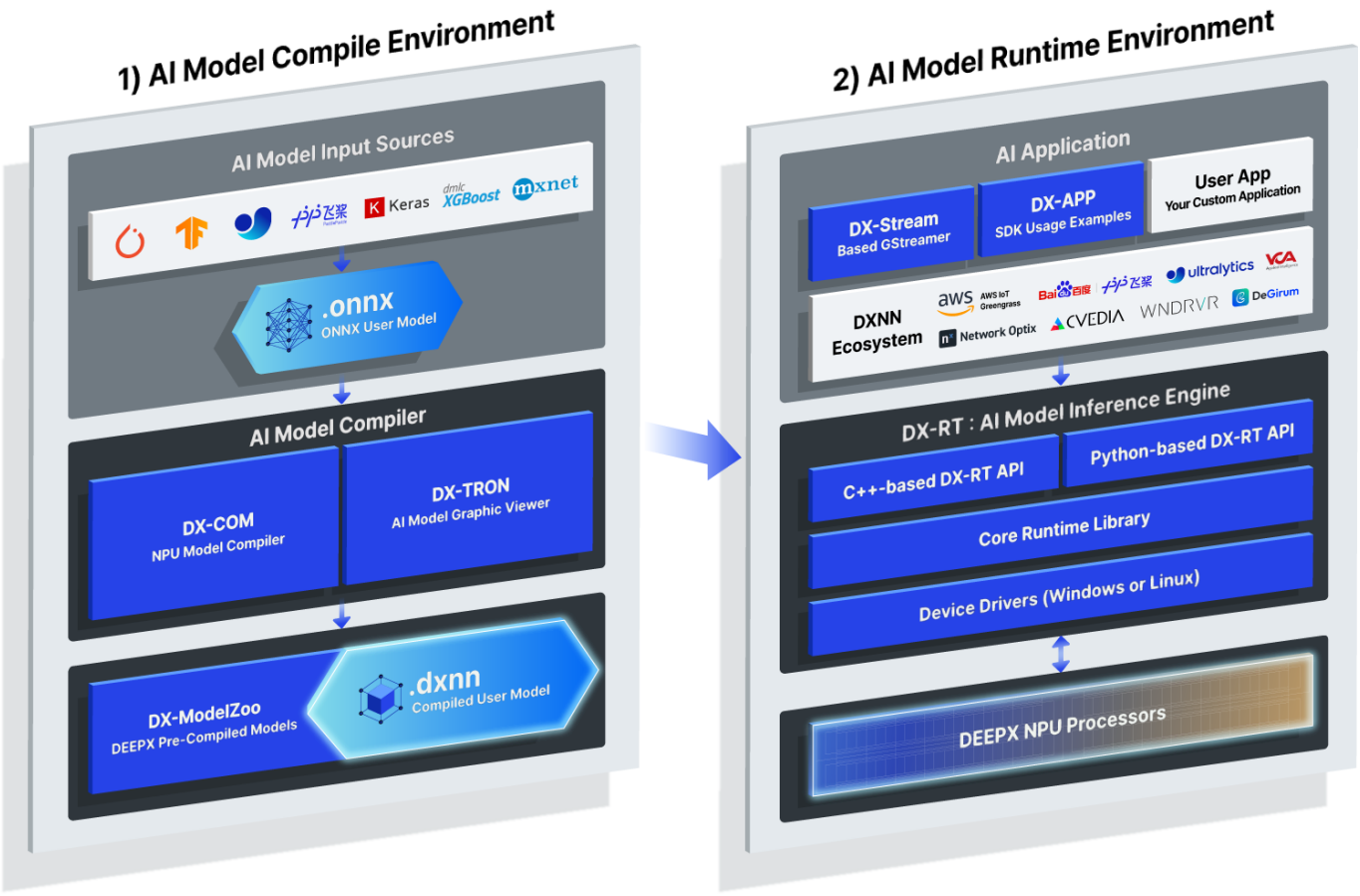
ⓒ Copyright 2026 DEEPX All Rights Reserved.

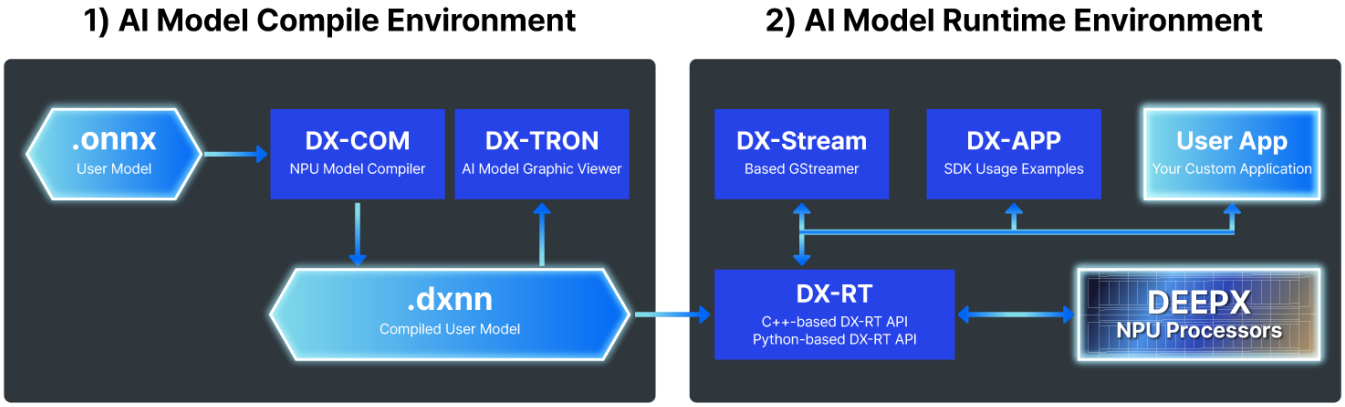
1. Introduction

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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.





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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

1.2 DX-AS (DEEPX All Suite) Environments and Installation Guide

1.2.1 AI Model Compile Environment (Compiler Platform)

Purpose
- Must be installed on the Host machine that will perform the compilation (converting) of ONNX models to our proprietary DXNN (DEEPX format).

Core Components - DX-COM: Converts ONNX models into highly optimized, NPU-ready binaries.

Flexibility & Support - OS: Compatible with Debian-based Linux - Architecture: Supports x86_64 only

Easy Installation - Our single script automates the full setup process - All DX-Compiler components are ready to use upon completion.

You can install dx-compiler by following the instructions at this LINK.

1.2.2 AI Model Runtime Environment (Deployment Platform)

Purpose
- Must be installed on the Target system where the DEEPX M1 M.2 module is attached and the DEEPX AI model (.dxnn) will be executed.

Core Components - DX-RT & DX-FW & NPU Driver: Foundational software for NPU control - DX-APP: C++ and Python examples to jump-start your projects - DX-Stream: GStreamer integration for seamless multimedia pipelines

Flexibility & Support - OS: Compatible with Debian-based Linux - Architecture: Supports both x86_64 and arm64

Easy Installation - Our script automates the entire process - One-time reboot is required after installation to finalize the NPU Driver setup

You can install dx-runtime by following the instructions at this LINK.

1.3 Documentation

1.3.1 Agent-Driven Development (Beta)

Link

ⓒ Copyright 2026 DEEPX All Rights Reserved.

1.3.2 Architecture Overview

Link

1.3.3 Installation Guide

Link

1.3.4 Getting Started

Link

1.3.5 Version Compatibility

Link

1.3.6 FAQ & Troubleshooting

Link

ⓒ Copyright 2026 DEEPX All Rights Reserved.

2. DEEPX Agent-Driven Development - dx-agent-dev (Beta)

> **Beta Feature**
>
> Agent-Driven development support is under active development.
> Skill definitions and routing behavior may change between releases.

2.1 Introduction

DEEPX Agent-Driven Development (`dx-agent-dev`) is here — currently in Beta. Build NPU apps with natural language: describe the app or model task in plain language and an AI coding agent — Claude Code, Cursor, GitHub Copilot, OpenCode, or Codex — drives the DEEPX knowledge base end to end: brainstorm → plan → TDD → verify, from ONNX/ `.pt` compilation to on-device DX-M1 NPU deployment. It is agent-driven development purpose-built for DEEPX NPUs in the **Ultralytics** model ecosystem, and every showcase below was produced this way — checked in with its prompt, measured results, and full build transcript.

Build DEEPX AI applications using natural language instructions. AI coding agents understand the DEEPX SDK ecosystem — GStreamer pipeline construction, `.dxnn` model resolution, InferenceEngine configuration, and DxPreprocess/DxInfer element wiring — so you can describe *what* you want and let the agent handle the implementation details.

Supported workflows include:

- Standalone inference apps with `IFactory`, `SyncRunner`, and `AsyncRunner`
- GStreamer video pipelines using DEEPX's 13 custom elements across 6 categories
- Cross-project builds that span dx_app, dx_stream, and dx-runtime
- Model compilation from ONNX to DXNN format using DX-COM (in dx-compiler)

2.2 Showcases

Every showcase below is a real app built on the DEEPX NPU SDK from a **single natural-language prompt** — fully autonomously — and checked into the suite with the prompt, measured results, a one-command reproduce, and the full recorded build-session transcript.

© Copyright 2026 DEEPX All Rights Reserved.

NPU-powered AI apps (mini-games)

Build a fully autonomous DEEPX-NPU app from natural language — in ~20 minutes, for ~$10. Pose-driven mini-games with arcade HUDs, built end to end from a single prompt.

Showcase	What it is	Build time	Agent turns	Output tokens	~Cost
Squat-Counting Mini-Game	Counts squat reps from knee/hip angles with an arcade HUD (reps / score / DOWN·UP·GOOD!).	≈ 12 min	132	≈ 109K	≈ $7.3
Stretching Coach Mini-Game	Guides 3 stretches with an animated coach avatar that demonstrates each target pose.	≈ 15 min	130	≈ 142K	≈ $8.1

Ultralytics ecosystem integration

Take any Ultralytics YOLO to the DEEPX NPU in one command — or retrain it for your domain — all in natural language. `format=deepx` export + 4-way eval (base/retrained × fp32-GPU / INT8-NPU); INT8 ≈ fp32, and the domain model runs faster on the NPU.

Showcase	What it is	Build time	Agent turns	Output tokens	~Cost
Ultralytics YOLO → DeepX Export	Turns an Ultralytics YOLO `.pt` into a deployable DeepX NPU model (`.dxnn`) in a single `yolo export ... format=deepx`	≈ 12 min	108	≈ 84K	≈ $2.4

© Copyright 2026 DEEPX All Rights Reserved.

Showcase	What it is	Build time	Agent turns	Output tokens	~Cost
	command, then runs NPU inference + verify.				
African Wildlife Monitoring	Retrains `yolo26n` on `african-wildlife` (buffalo/elephant/rhino/zebra) for a safari/conservation camera; 4-way eval base/retrained × fp32/INT8.	≈ 7 min	78	≈ 85K	≈ $3.2
Construction PPE Safety	Retrains `yolo26n` on `construction-ppe` for a site-safety camera (helmet/vest/...); 4-way eval base/retrained × fp32/INT8.	≈ 17 min	102	≈ 93K	≈ $4.0
Brain-Tumor Screening	Retrains `yolo26n` on `brain-tumor` (MRI/CT) for a medical edge device; 4-way eval base/retrained × fp32/INT8.	≈ 9 min	91	≈ 103K	≈ $3.7
Pharmaceutical Pill Inspection	Retrains `yolo26n` on	≈ 8 min	118	≈ 103K	≈ $5.1

© Copyright 2026 DEEPX All Rights Reserved.

Showcase	What it is	Build time	Agent turns	Output tokens	~Cost
	`medical-pills` for a pharma counting station; 4-way eval base/retrained × fp32/INT8.				

PaddlePaddle ecosystem integration

PaddleOCR (PP-OCRv5) on the DEEPX NPU — real-time video & webcam OCR from a single, concise prompt. Baidu's PaddlePaddle OCR (text detection → orientation → recognition) running on the DX-M1 NPU.

Showcase	What it is	Build time	Agent turns	Output tokens	~Cost
Video / Webcam OCR (PP-OCRv5)	Real-time text detection + recognition on the DX-M1 NPU — one code path for a video file and a live webcam, overlaying detected boxes + recognized strings.	≈ 18 min	175	≈ 184K	≈ $12.0

RapidAI ecosystem integration

A PDF → Markdown document-conversion app on the DEEPX NPU — from a single, concise natural-language prompt. RapidAI's RapidDoc (PP-StructureV3): layout, OCR, tables, formulas —

ⓒ Copyright 2026 DEEPX All Rights Reserved.

running PaddlePaddle-trained models on the DX-M1 NPU. A standalone, self-contained app generated from the fork's pipeline.

Showcase	What it is	Build time	Agent turns	Output tokens	~Cost
PDF → Markdown (document conversion app)	Converts a PDF (digital or scanned) to structured Markdown + JSON — layout analysis, OCR, tables and formulas — on the DEEPX DX-M1 NPU via the RapidDoc fork. Supports `--parse-method auto|txt|ocr`.	≈ 12 min	133	≈ 148.5K	≈ $6.2

Full catalog + per-showcase summaries (with build GIFs) → `dx-agent-dev-showcase/README.md`. Each row's link opens that showcase's own README — the verbatim prompt, the 4-way eval / gameplay detail, and its session transcript.

2.2.1 How it works — the harness, not (just) the model

Each app ships its **complete agent session transcript**, the most direct way to see that the result comes less from the raw model and more from the **instructions, skills, and verification gates** the harness imposes. Reading a transcript you can watch:

- **Instruction-following** — the agent honors the suite HARD GATES: session sentinels (`[DX-AGENT-DEV: START]` / `DONE`), output isolated to a session directory (never touching existing source), and no placeholder/stub code.
- **Skill & agent utilization** — it invokes the mandatory sequence as real tool calls — `dx-skill-router → dx-agent-brainstorm → dx-swe-writing-plans → dx-agent-tdd → dx-agent-verify` — rather than just *mentioning* them.
- **Actual reasoning** — inspecting the closest existing example, confirming real framework APIs from the knowledge base, writing validation first (RED), then generating + verifying file-by-file before declaring done.

© Copyright 2026 DEEPX All Rights Reserved.

2.3 Prerequisites

Requirement	Details
DEEPX development environment	DX-RT SDK installed and `setup_env.sh` sourced
AI coding agent (one of)	Claude Code, GitHub Copilot (VS Code), Cursor, OpenCode, or Codex CLI
Python	3.10+ with the dx-all-suite packages installed

2.4 Architecture Overview

The agent-driven knowledge base is organized into three independent layers. Each layer ships its own `.deepx/` directory containing skills, instructions, and memory files that the agent reads at task time.

2.4.1 dx_app — Standalone Inference

Python and C++ applications that run inference without GStreamer. Key abstractions:

- **IFactory** — creates model-specific pre/post-processing pipelines
- **SyncRunner / AsyncRunner** — blocking and non-blocking inference executors
- **DxInfer** — low-level inference wrapper around InferenceEngine

The `.deepx/` knowledge base covers model loading, `.dxnn` resolution, batch processing, and result visualization.

2.4.2 dx_stream — GStreamer Pipelines

Real-time video analytics built on GStreamer. The agent understands all 13 DEEPX elements organized into 6 functional categories (source, inference, overlay, encoding, streaming, and sink) and can assemble multi-branch pipelines from a single natural-language prompt.

2.4.3 dx-runtime — Integration Layer

Cross-project routing and unified validation. dx-runtime sits above the other two layers, dispatching tasks to the correct sub-project builder and applying consistent coding standards, testing patterns, and model-management rules.

© Copyright 2026 DEEPX All Rights Reserved.

2.4.4 dx-compiler — Model Compilation

DXNN model compilation powered by DX-COM. The agent understands the full compilation pipeline — ONNX model validation, config.json generation with auto-inferred parameters, calibration data preparation, INT8 quantization, and PPU configuration — and can compile models from a single natural-language prompt. Before compilation, the agent asks mandatory brainstorming questions about NMS-free model detection, ONNX simplification, and PPU compilation to ensure correct configuration.

2.5 Available Agents and Skills

Agents and skills are available at every level of the repository. The top-level dx-all-suite provides routing agents that classify tasks and dispatch to the correct submodule.

2.5.1 Agents by Level

Level	Agent	Description
dx-all-suite	`@dx-suite-builder`	Top-level router — classifies tasks and routes to the appropriate submodule
dx-all-suite	`@dx-suite-validator`	Suite-wide validation — runs framework checks across all 3 levels
dx-runtime	`@dx-runtime-builder`	Cross-project builder — routes to dx_app or dx_stream
dx-runtime	`@dx-validator`	Unified validation orchestrator with feedback loop
dx_app	`@dx-app-builder`	Standalone inference builder — routes to specialist builders
dx_app	`@dx-python-builder`	Python inference app builder (4 variants: sync, async, cpp_postprocess, async_cpp_postprocess)
dx_app	`@dx-cpp-builder`	C++ inference app builder
dx_app	`@dx-model-manager`	Model download and registry manager

© Copyright 2026 DEEPX All Rights Reserved.

Level	Agent	Description
dx_app	`@dx-validator`	dx_app validation and feedback loop
dx_stream	`@dx-stream-builder`	GStreamer pipeline builder — routes to specialist builders
dx_stream	`@dx-pipeline-builder`	Pipeline construction (6 categories incl. broker)
dx_stream	`@dx-validator`	dx_stream validation and feedback loop
dx-compiler	`@dx-compiler-builder`	Model compilation router — routes to converter or compiler
dx-compiler	`@dx-model-converter`	PyTorch to ONNX model converter
dx-compiler	`@dx-dxnn-compiler`	ONNX to DXNN compiler (DX-COM)

2.5.2 Skills (OpenCode only)

Level	Skill	Description
dx-runtime	`/dx-agent-runtime-validate`	Validate, collect feedback, apply fixes, verify
dx_app	`/dx-agent-app-build-python`	Build Python inference app
dx_app	`/dx-agent-app-build-cpp`	Build C++ inference app
dx_app	`/dx-agent-app-build-async`	Build async high-performance app
dx_app	`/dx-agent-app-model-management`	Download and configure models
dx_app	`/dx-agent-app-validate`	Run validation checks
dx_stream	`/dx-agent-stream-build-pipeline`	Build GStreamer pipeline app
dx_stream	`/dx-agent-stream-build-mqtt-kafka`	Build MQTT/Kafka pipeline app
dx_stream	`/dx-agent-stream-validate`	Run validation checks

© Copyright 2026 DEEPX All Rights Reserved.

Level	Skill	Description
dx_stream	`/dx-agent-stream-model-management`	Download and configure models
dx-compiler	`/dx-agent-compiler-convert`	Convert PyTorch model to ONNX
dx-compiler	`/dx-agent-compiler-compile`	Compile ONNX model to DXNN
dx-compiler	`/dx-agent-compiler-validate`	Validate compiled DXNN output
DX All Suite	`/dx-swe-brainstorm`	Process: collaborative design session before any work
DX All Suite	`/dx-swe-tdd`	Process: test-driven development — validate incrementally
DX All Suite	`/dx-swe-verify`	Process: verify before claiming completion — evidence before assertions
dx-runtime	`/dx-swe-brainstorm`	Process: collaborative design session before code generation
dx-runtime	`/dx-swe-tdd`	Process: test-driven development — validate each file immediately after creation
dx-runtime	`/dx-swe-verify`	Process: verify before claiming completion — evidence before assertions
dx_app	`/dx-swe-brainstorm`	Process: collaborative design session before code generation
dx_app	`/dx-swe-tdd`	Process: test-driven development — validate each file immediately after creation
dx_app	`/dx-swe-verify`	Process: verify before claiming completion — evidence before assertions
dx_stream	`/dx-swe-brainstorm`	Process: collaborative design session before code generation

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Level	Skill	Description
dx_stream	`/dx-swe-tdd`	Process: test-driven development — validate each file immediately after creation
dx_stream	`/dx-swe-verify`	Process: verify before claiming completion — evidence before assertions
dx-compiler	`/dx-swe-brainstorm`	Process: collaborative design session before compilation
dx-compiler	`/dx-swe-tdd`	Process: test-driven development — validate each step incrementally
dx-compiler	`/dx-swe-verify`	Process: verify before claiming completion — evidence before assertions

Tip

If you are unsure which submodule to target, use `@dx-suite-builder` at the top level — it will classify your task and route to the correct builder.

2.6 Supported AI Tools

Agent-Driven development works with five AI coding tools. Each tool auto-loads the `.deepx/` knowledge base through its own configuration mechanism.

Tool	Type	Auto-Load Mechanism	Agent Invocation	Skill Invocation
Claude Code	CLI	`CLAUDE.md` at project root	Free-form conversation; Context Routing Table dispatches automatically	—
GitHub Copilot	VS Code	`.github/copilot-instructions.md`		—

© Copyright 2026 DEEPX All Rights Reserved.

Tool	Type	Auto-Load Mechanism	Agent Invocation	Skill Invocation
			`@agent-name "prompt"` in Copilot Chat	
Cursor	IDE	`.cursor/rules/*.mdc`	Free-form conversation; rules loaded by `alwaysApply` or `globs`	—
OpenCode	CLI	`AGENTS.md` + `opencode.json`	`@agent-name "prompt"`	`/skill-name` slash command
Codex CLI	CLI	`AGENTS.md` + `.codex/skills/dx-codex-identity/SKILL.md`	Free-form conversation (`~/bin/codex exec ...`)	`cat .deepx/skills/<name>/SKILL.md` (read directly)

2.6.1 What Gets Auto-Loaded

Tool	Global Context	File-Specific Context	Agents	Skills
Claude Code	`CLAUDE.md`	Context Routing Table (manual)	`.claude/agents/*.md` (generated)	`.deepx/skills/` (read directly)
Copilot	`.github/copilot-instructions.md`	`.github/instructions/*.instructions.md` (`applyTo:` glob)	`.github/agents/*.agent.md`	`.github/skills/` (inline copies)
Cursor	`.cursor/rules/dx-*.mdc` (`alwaysApply: true`)	`.cursor/rules/*.mdc` (`globs: [...]`)	`.cursor/rules/` agent `.mdc` files	`.cursor/rules/` skill `.mdc` files
OpenCode	`AGENTS.md` + `opencode.json` instructions	—	`.opencode/agents/*.md`	`.deepx/skills/*/SKILL.md`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Tool	Global Context	File-Specific Context	Agents	Skills
Codex CLI	`AGENTS.md`	—	`.deepx/agents/*.md` (direct `cat`)	`.codex/skills/dx-codex-identity/` (auto) + `.deepx/skills/` (manual `cat`)

2.6.2 First-Time Setup

No additional configuration is needed. Open the project directory in your preferred tool and the configuration files are loaded automatically:

```
# Claude Code
cd dx-all-suite
claude

# OpenCode
cd dx-all-suite
opencode

# Codex CLI
cd dx-all-suite
~/bin/codex

# GitHub Copilot — open folder in VS Code
code dx-all-suite

# Cursor CLI
cd dx-all-suite
cursor-agent
```

2.6.3 Platform File Loading Reference

Each AI coding agent auto-loads different configuration files at the suite level. Files marked **Auto** are loaded on every conversation; **@mention** files are invoked manually via agent or skill commands.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Git submodule boundary*

Copilot Chat/CLI, Claude Code, and Codex CLI only see files at the current git root. When opened at `dx-all-suite/`, they do NOT auto-load sub-project files in `dx-compiler/`, `dx-runtime/`, etc. (these are separate git submodules). OpenCode bridges this boundary via explicit path references in `opencode.json`.

Auto-Loaded Files

File	Auto-loaded by	Loading
`.github/copilot-instructions.md`	Copilot Chat/CLI	Auto
`CLAUDE.md`	Claude Code	Auto
`AGENTS.md` + `opencode.json`	OpenCode	Auto
`AGENTS.md` + `.codex/skills/dx-codex-identity/SKILL.md`	Codex CLI	Auto
`.cursor/rules/dx-all-suite.mdc`	Cursor	Auto

Agent Files (Manual @mention)

Agent	Copilot (`@mention`)	OpenCode (`@mention`)
`dx-suite-builder`	`.github/agents/dx-suite-builder.agent.md`	`.opencode/agents/dx-suite-builder.md`
`dx-suite-validator`	`.github/agents/dx-suite-validator.agent.md`	`.opencode/agents/dx-suite-validator.md`

NOTE

Claude Code has generated agent files in `.claude/agents/` (e.g., `dx-suite-builder.md`).
Cursor has agent `.mdc` files in `.cursor/rules/` (e.g., `dx-suite-builder.mdc`).
Claude Code also uses the Context Routing Table in `CLAUDE.md` to dispatch tasks.

© Copyright 2026 DEEPX All Rights Reserved.

Skill Files (OpenCode Only — `/slash-command`)

Skill	File
`/dx-swe-brainstorm`	`.deepx/skills/dx-swe-brainstorm/SKILL.md`
`/dx-swe-verify`	`.deepx/skills/dx-swe-verify/SKILL.md`
`/dx-swe-tdd`	`.deepx/skills/dx-swe-tdd/SKILL.md`
`/dx-swe-parallel-agents`	`.deepx/skills/dx-swe-parallel-agents/SKILL.md`
`/dx-swe-executing-plans`	`.deepx/skills/dx-swe-executing-plans/SKILL.md`
`/dx-swe-receiving-review`	`.deepx/skills/dx-swe-receiving-review/SKILL.md`
`/dx-swe-requesting-review`	`.deepx/skills/dx-swe-requesting-review/SKILL.md`
`/dx-skill-router`	`.deepx/skills/dx-skill-router/SKILL.md`
`/dx-swe-subagent-dev`	`.deepx/skills/dx-swe-subagent-dev/SKILL.md`
`/dx-swe-debugging`	`.deepx/skills/dx-swe-debugging/SKILL.md`
`/dx-swe-writing-plans`	`.deepx/skills/dx-swe-writing-plans/SKILL.md`

Shared Knowledge Base (`.deepx/`)

The `.deepx/` directory is the **canonical source** (single source of truth) for all platform-specific files. It contains agents, skills, templates, and fragments in a platform-agnostic format. The `dx-agent-gen` generator transforms this into platform-specific files for Copilot (`.github/`), Claude Code (`.claude/`), OpenCode (`.opencode/`), and Cursor (`.cursor/rules/`).

Directory	Contents
`agents/`	`dx-suite-builder`, `dx-suite-validator`
`skills/`	13 skills (domain + shared process skills)
`templates/`	`{en,ko}/*.tmpl` — instruction file templates

© Copyright 2026 DEEPX All Rights Reserved.

Directory	Contents
`templates/fragments/`	`{en,ko}/*.md` — shared sections reused across repos
`memory/`	Persistent cross-session knowledge
`knowledge/`	Structured reference data
`instructions/`	Internal agent instructions
`toolsets/`	Tool reference documentation

Instruction files (`CLAUDE.md` , `AGENTS.md` , `copilot-instructions.md` , EN+KO) are also generated from templates and fragments — they should not be edited directly.

Platform File Generation

All platform-specific files are generated from `.deepx/` by the `dx-agent-dev-gen` package. Never edit generated files directly.

```
pip install -e .deepx/tools   # Install generator
dx-agent-gen generate                    # Generate platform files
dx-agent-gen check                       # Verify no drift
```

A pre-commit hook enforces that generated files stay in sync:

```
.deepx/tools/scripts/install-hooks.sh   # One-time setup
```

2.7 Quick Start by Tool

2.7.1 From dx-all-suite (Top-Level Routing)

If you are working at the top-level dx-all-suite directory and want the agent to automatically route to the correct submodule:

Prompt:

ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
"Compile yolo26n.onnx to DXNN and build a person detection Python app with it"
```

Tool	How to Use
Claude Code	Open `dx-all-suite/` and type the prompt. `CLAUDE.md` routes to dx-compiler for compilation and dx_app for app generation.
GitHub Copilot	Open Copilot Chat: `@dx-suite-builder` followed by the prompt. The agent classifies the task and routes to the correct submodules.
Cursor	Open `dx-all-suite/` and type the prompt. The `alwaysApply` rule routes to the appropriate submodules.
OpenCode	Open `dx-all-suite/` : `@dx-suite-builder` followed by the prompt. The agent routes automatically.
Codex CLI	Open `dx-all-suite/` and type the prompt (or `~/bin/codex exec "<prompt>"`). `AGENTS.md` is read automatically and routes across submodules.

2.7.2 From a Submodule (Direct Access)

When working directly in a submodule, use prompts tailored to that submodule's scope:

Submodule	Example Prompt
dx-compiler	`"Convert my yolo26x.pt to ONNX and compile it to DXNN for DX-M1"`
dx_app	`"Build a yolo26n person detection app using Python"`
dx_stream	`"Build a detection pipeline with RTSP camera and tracking"`

Tool	How to Use
Claude Code	Open the submodule directory and type the prompt directly. `CLAUDE.md` is read automatically. The

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Tool	How to Use
	Context Routing Table dispatches to the correct `.deepx/` skill files.
GitHub Copilot	Open Copilot Chat: `@dx-app-builder`, `@dx-stream-builder`, or `@dx-compiler-builder` followed by the prompt. Copilot reads `.github/copilot-instructions.md` on every chat.
Cursor	Open the submodule folder and type the prompt directly. Rules with `alwaysApply: true` are loaded on every conversation. Rules with `globs:` patterns activate when editing matching files.
OpenCode	Open the submodule directory and use the appropriate agent (`@dx-app-builder`, `@dx-stream-builder`, or `@dx-compiler-builder`) or the corresponding skill slash command.
Codex CLI	Open the submodule directory and type the prompt (or `~/bin/codex exec "<prompt>"`). `AGENTS.md` is read automatically; `cat` the relevant `.deepx/skills/<name>/SKILL.md` directly as needed.

2.8 End-to-End Scenarios

These scenarios demonstrate cross-project workflows that span multiple submodules. For sub-project-specific scenarios, see the individual guides linked below.

2.8.1 Scenario 1: Custom Model Conversion + SDK Porting + Validation

A full pipeline that compiles a custom model, ports inference code to the DEEPX SDK, and validates the result.

Prompt:

ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
"I have yolo26x-custom.onnx at ./models/ and my inference code at ./inference.py
using onnxruntime. Convert it to DXNN and port my code to DEEPX SDK."
```

Tool	How to Use
Claude Code	Open `dx-all-suite/` and type the prompt. The suite builder orchestrates: (a) dx-compiler compiles the ONNX model to DXNN, (b) dx_app ports the inference code, (c) validation confirms the ported app works.
GitHub Copilot	`@dx-suite-builder` followed by the prompt. The agent routes compilation to dx-compiler and porting to dx_app.
Cursor	Open `dx-all-suite/` and type the prompt. The router dispatches to the correct submodules.
OpenCode	`@dx-suite-builder` followed by the prompt.
Codex CLI	Open `dx-all-suite/` and type the prompt. `AGENTS.md` orchestrates the cross-submodule work.

This scenario involves three stages: 1. **dx-compiler**: Compile `yolo26x-custom.onnx` → `yolo26x-custom.dxnn` with auto-inferred config 2. **dx_app**: Generate Python inference app using `InferenceEngine` with the compiled model 3. **Validation**: Run the ported app and compare outputs against the original onnxruntime code

2.8.2 Scenario 2: Model Compilation + Sample App Generation

Compile a model and generate a standalone inference app that uses the compiled output. This cross-project scenario spans dx-compiler and dx_app.

Prompt:

```
"Compile yolo26n.onnx to DXNN and generate a Python detection app that uses the
compiled model"
```

Tool	How to Use
Claude Code	Open `dx-all-suite/` and type the prompt. The suite builder orchestrates: (a) dx-compiler compiles

© Copyright 2026 DEEPX All Rights Reserved.

Tool	How to Use
	ONNX to DXNN, (b) dx_app generates a Python app referencing the compiled model.
GitHub Copilot	`@dx-suite-builder` followed by the prompt. Routes compilation to dx-compiler and app generation to dx_app.
Cursor	Open `dx-all-suite/` and type the prompt. The router dispatches to both submodules.
OpenCode	`@dx-suite-builder` followed by the prompt.
Codex CLI	Open `dx-all-suite/` and type the prompt. `AGENTS.md` orchestrates the cross-submodule work.

This scenario involves two stages: 1. **dx-compiler**: Compile `yolo26n.onnx` → `yolo26n.dxnn` with auto-inferred config 2. **dx_app**: Generate a Python detection app using the compiled `.dxnn` model

2.8.3 Scenario 3: Model Compilation + Streaming Pipeline Generation

Compile a model and generate a GStreamer streaming pipeline that uses the compiled output. This cross-project scenario spans dx-compiler and dx_stream.

Prompt:

```
"Compile yolo26n.onnx to DXNN and build a detection streaming pipeline with RTSP
output"
```

Tool	How to Use
Claude Code	Open `dx-all-suite/` and type the prompt. The suite builder orchestrates: (a) dx-compiler compiles ONNX to DXNN, (b) dx_stream generates a GStreamer pipeline with RTSP output.
GitHub Copilot	`@dx-suite-builder` followed by the prompt. Routes compilation to dx-compiler and pipeline to dx_stream.
Cursor	Open `dx-all-suite/` and type the prompt. The router dispatches to both submodules.
OpenCode	`@dx-suite-builder` followed by the prompt.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Tool	How to Use
Codex CLI	Open `dx-all-suite/` and type the prompt. `AGENTS.md` orchestrates the cross-submodule work.

This scenario involves two stages: 1. **dx-compiler**: Compile `yolo26n.onnx` → `yolo26n.dxnn` with auto-inferred config 2. **dx_stream**: Generate a detection pipeline with DxInfer using the compiled model and RTSP streaming output

2.8.4 Scenario 4: PPU Model Compilation + Detection App

Compile a YOLO model with PPU (Pre/Post Processing Unit) support for hardware-accelerated post-processing, then generate an app that uses the PPU model.

Prompt:

```
"Compile yolo26n.onnx with PPU support and generate a detection app for the PPU
model"
```

Tool	How to Use
Claude Code	Open `dx-all-suite/` and type the prompt. The suite builder orchestrates: (a) dx-compiler compiles with PPU config (auto-detected type based on YOLO version), (b) dx_app generates a PPU-specific app with simplified postprocessing.
GitHub Copilot	`@dx-suite-builder` followed by the prompt. Routes to dx-compiler for PPU compilation and dx_app for PPU app generation.
Cursor	Open `dx-all-suite/` and type the prompt. The router dispatches to both submodules.
OpenCode	`@dx-suite-builder` followed by the prompt.
Codex CLI	Open `dx-all-suite/` and type the prompt. `AGENTS.md` orchestrates the cross-submodule work.

This scenario involves two stages: 1. **dx-compiler**: Compile with PPU config — the agent auto-detects PPU type (Type 0 for anchor-based YOLO, Type 1 for anchor-free YOLO) 2. **dx_app**: Generate a PPU-

ⓒ Copyright 2026 DEEPX All Rights Reserved.

specific detection app under `src/python_example/ppu/` with simplified postprocessing (bounding boxes decoded by hardware)

2.9 Cross-Project Routing

The dx-all-suite meta guide provides routing to all sub-project scenarios. If your task matches a scenario in a sub-project guide, the suite builder will route you there automatically.

- **dx-runtime scenarios** (cross-project builds, unified validation): See the dx-runtime guide
- **dx_app scenarios** (Python/C++ inference apps): See the dx_app guide
- **dx_stream scenarios** (GStreamer pipelines): See the dx_stream guide
- **dx-compiler scenarios** (model compilation): See the dx-compiler guide

> **Tip**
>
> You don't need to navigate to sub-project directories. Use `@dx-suite-builder` at the dx-all-suite level — it routes to any sub-project automatically.

2.10 Sub-Project Guides

Each sub-project has a detailed agent-driven development guide covering its specific skills, element catalogs, and worked examples:

Sub-Project	Guide
dx-runtime	`dx-runtime/docs/source/agent_development.md`
dx_app	`dx_app/docs/source/docs/13_DX-APP_Agent_Driven_Development.md`
dx_stream	`dx_stream/docs/source/docs/08_DX-STREAM_Agent_Driven_Development.md`
dx-compiler	`dx-compiler/source/docs/05_DX-COMPILER_Agent_Driven_Development.md`

© Copyright 2026 DEEPX All Rights Reserved.

2.11 Internal Reference Documents

For a deeper view of the `.deepx/` canonical source, generator pipeline, and harness development model (intended for contributors, not end users):

Document	Scope
`.deepx/docs/dx-agent-dev-overview.md`	Comprehensive walk-through of every `.deepx/` directory across all 5 repos
`.deepx/README.md`	Top-level master index for the `.deepx/` knowledge base
`.deepx/docs/skill-architecture.md`	3-tier skill model (SWE / Agent-Driven / Harness)
`.deepx/tools/README.md`	`dx-agent-gen` generator package guide
`.deepx/tools/scripts/README.md`	Operational scripts (`run_all.sh` , hooks, E2E loop)

2.12 Output Isolation

By default, all agent-generated code is placed in `dx-agent-dev/<session_id>/` within the target sub-project. This prevents accidental modifications to existing production code.

Output Type	Path	When
Default (isolated)	`dx-agent-dev/<session_id>/`	Always, unless user says otherwise
Production	`src/`	Only when explicitly requested by the user

Session ID format: `YYYYMMDD-HHMMSS_<agent>_<model>_<task>` where `<agent>` is `claude` , `codex` , `copilot` , `cursor` , or `opencode` .

Each session directory contains: - `README.md` — session metadata, generated file list, run instructions - `session.json` — machine-readable session configuration

The `dx-agent-dev/` directory is git-ignored in both dx_app and dx_stream.

© Copyright 2026 DEEPX All Rights Reserved.

2.12.1 dx-compiler Session Directories

For dx-compiler, session directories additionally contain: - `calibration_dataset` — symlink to `dx_com/calibration_dataset/` - `config.json` — auto-generated DX-COM config with relative calibration path - `compiler.log` — compilation log (when `--gen_log` is used)

The agent automatically sets up calibration data (checking `dx_com/calibration_dataset/`, running setup scripts if needed, and creating symlinks with relative paths).

2.12.2 Suite-Level Cross-Project Output

When running cross-project tasks from the dx-all-suite level (e.g., compile + deploy), artifacts are created in each target sub-project's `dx-agent-dev/` directory. Additionally, symbolic links are created in `dx-all-suite/dx-agent-dev/` for unified access:

```
dx-all-suite/dx-agent-dev/
├── dx-compiler_20260409-070940_yolo26n_pt_to_dxnn -> ../dx-compiler/dx-agent-dev/
20260409-...
└── dx_app_20260409-071500_yolo26n_detection_app -> ../dx-runtime/dx_app/dx-agent-
dev/20260409-...
```

Symlink naming convention: `{subproject}_{session_id}`.

2.13 Session Sentinels

Agents output fixed markers at the start and end of each task for automated testing:

Marker	When
`[DX-AGENT-DEV: START]`	**CRITICAL** — Absolute first line of the agent's first response, before ANY other text, tool calls, or reasoning. Non-negotiable even if the user says "just proceed" — automated tests WILL fail without it.
`[DX-AGENT-DEV: DONE (output-dir: <relative_path>)]`	Last line after all work is complete. `<relative_path>` is the session output directory relative to the project root. If no files were generated, omit the `(output-dir: ...)` part.

Important: DONE means all deliverables are produced — implementation code, scripts, configs, and validation results. If the agent only produced planning artifacts (specs, plans, design documents) without implementing actual code, DONE must NOT be output.

© Copyright 2026 DEEPX All Rights Reserved.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

3. Agent-Driven Development Showcases

3.1 dx-agent-dev Showcases

> Real apps built on the **DEEPX NPU SDK** by an AI coding agent from a **single natural-language prompt** — each checked in with the prompt, measured results, a one-command reproduce, and the full recorded build-session transcript.

These demonstrate **dx-agent-dev (Beta)**: you describe the app/task in plain language and the agent drives the DEEPX knowledge base end to end (brainstorm → plan → TDD → verify). What the feature is and how it works → Agent-Driven Development docs. Each card below links to that showcase's own README (full detail + transcript).

> **Note on build time & cost** — The build time, output-token count, and cost shown for each showcase are **actual measurements** taken from that showcase's recorded build-session transcript (`*-session.md`), priced with the coding agent's published billing policy — not estimates, and not cherry-picked best runs. Because AI coding agents are non-deterministic, the same prompt does not always consume the same output tokens, so your own build time and cost may differ depending on environment and model version.

3.1.1 NPU-powered AI apps (mini-games)

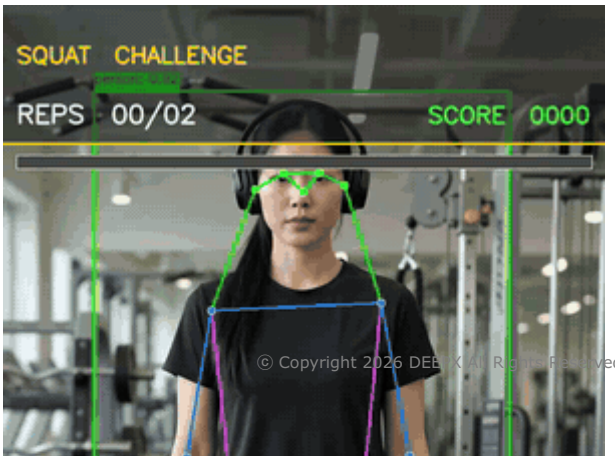
Build a fully autonomous DEEPX-NPU app from natural language — in ~20 minutes, for ~$10. Pose-driven mini-games with arcade HUDs, built end to end from a single prompt.

Showcase	Kind	Highlight
Squat-Counting Mini-Game	game	pose game + arcade HUD
Stretching Coach Mini-Game	game	coach avatar + 3 stages

Squat-Counting Mini-Game

Counts squat reps from knee/hip angles with an arcade HUD (reps / score / DOWN·UP·GOOD!).

Highlight: pose game + arcade HUD · **Claude Opus 4.8** · ≈ 12 min · ≈ $7.3 — details →



© Copyright 2026 DEEPX All Rights Reserved.

Stretching Coach Mini-Game



Guides 3 stretches with an animated coach avatar that demonstrates each target pose.

Highlight: coach avatar + 3 stages · **Claude Opus 4.8** · ≈ 15 min · ≈ $8.1 — details →

3.1.2 Ultralytics ecosystem integration

Take any Ultralytics YOLO to the DEEPX NPU in one command — or retrain it for your domain — all in natural language. `format=deepx` export + 4-way eval (base/retrained × fp32-GPU / INT8-NPU); INT8 ≈ fp32, and the domain model runs faster on the NPU.

Showcase	Kind	Highlight
Ultralytics YOLO → DeepX Export	export	1-cmd .pt → .dxnn
African Wildlife Monitoring	retrain	mAP ~0.0007→0.79, 59→80 FPS
Construction PPE Safety	retrain	mAP 0.0001→0.257, 58→80 FPS
Brain-Tumor Screening	retrain	mAP ~0.0005→0.40, 59→83 FPS
Pharmaceutical Pill Inspection	retrain	mAP ~0.001→0.75 (mAP50 0.97), 55→78 FPS

Ultralytics YOLO → DeepX Export

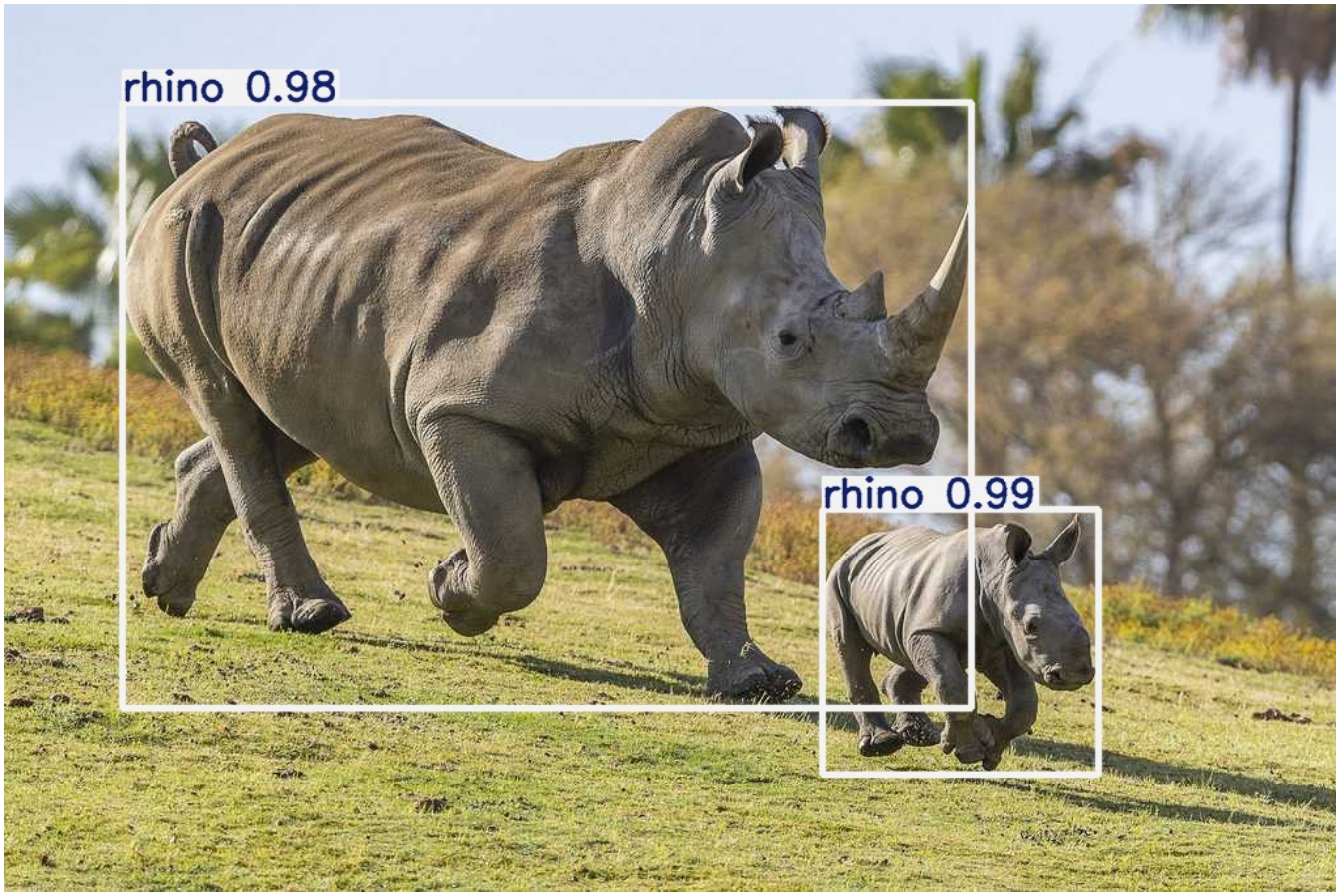


© Copyright 2026 DEEPX All Rights Reserved.

Turns an Ultralytics YOLO `.pt` into a deployable DeepX NPU model (`.dxnn`) in a single `yolo export ... format=deepx` command, then runs NPU inference + verify.

Highlight: 1-cmd .pt → .dxnn · **Claude Sonnet 4.6** · ≈ 12 min · ≈ $2.4 — details →

African Wildlife Monitoring



Retrains `yolo26n` on `african-wildlife` (buffalo/elephant/rhino/zebra) for a safari/conservation camera; 4-way eval base/retrained × fp32/INT8.

Highlight: mAP ~0.0007→0.79, 59→80 FPS · **Claude Opus 4.8** · ≈ 7 min · ≈ $3.2 — details →

Construction PPE Safety



© Copyright 2026 DEEPX All Rights Reserved.

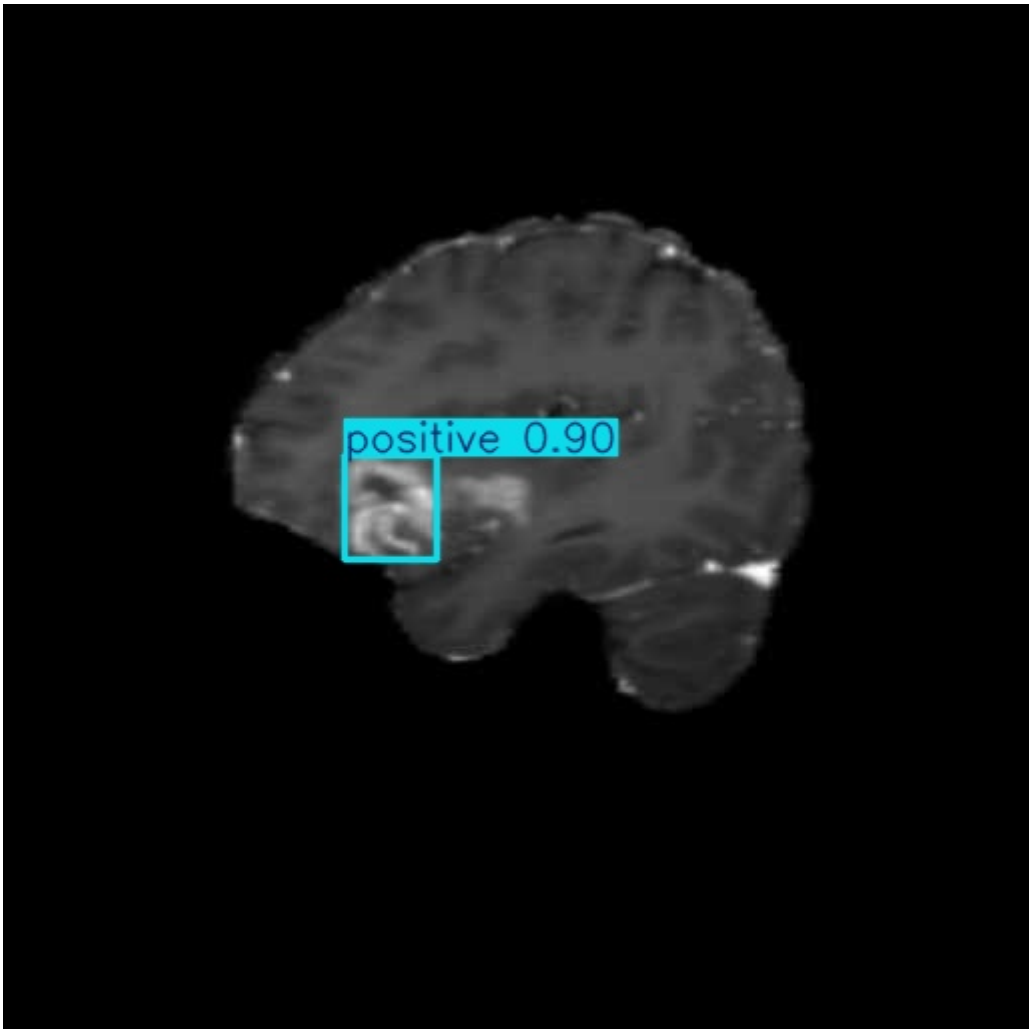
Retrains `yolo26n` on `construction-ppe` for a site-safety camera (helmet/vest/...); 4-way eval base/retrained × fp32/INT8.

Highlight: mAP 0.0001→0.257, 58→80 FPS · **Claude Opus 4.8** · ≈ 17 min · ≈ $4.0 — details →

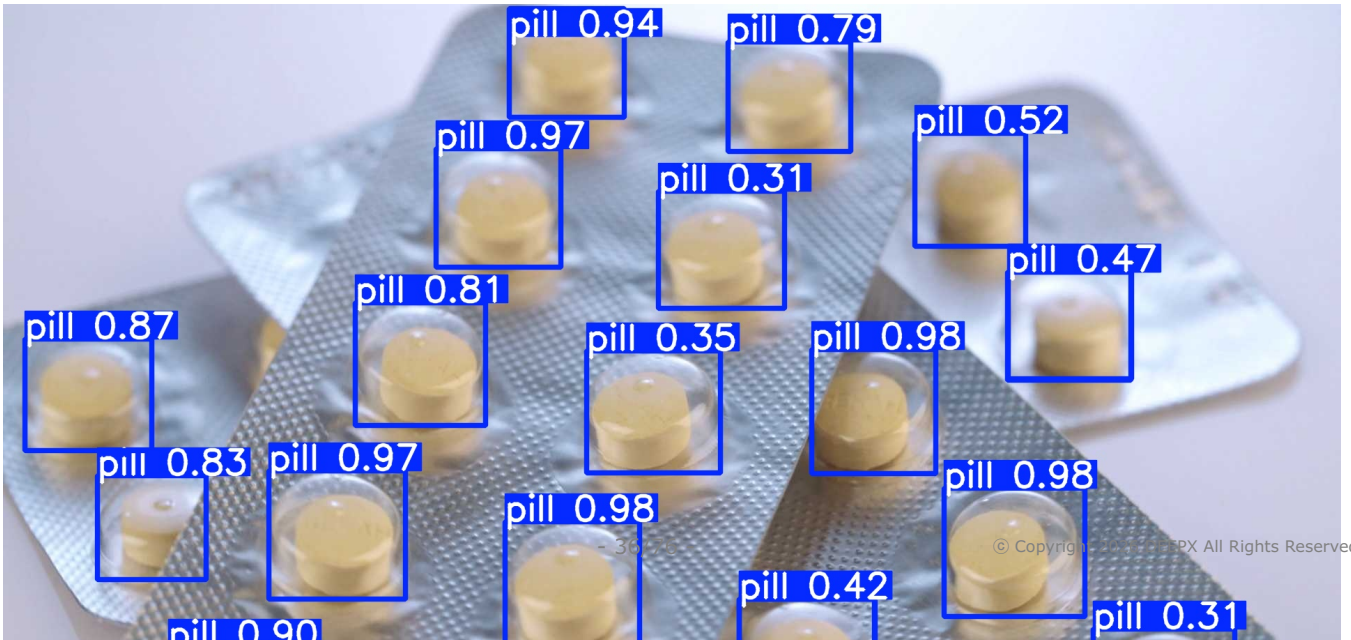
Brain-Tumor Screening

Retrains `yolo26n` on `brain-tumor` (MRI/CT) for a medical edge device; 4-way eval base/retrained × fp32/INT8.

Highlight: mAP ~0.0005→0.40, 59→83 FPS · **Claude Opus 4.8** · ≈ 9 min · ≈ $3.7 — details →



Pharmaceutical Pill Inspection



© Copyright 2026 FPX All Rights Reserved.

Retrains `yolo26n` on `medical-pills` for a pharma counting station; 4-way eval base/retrained × fp32/INT8.

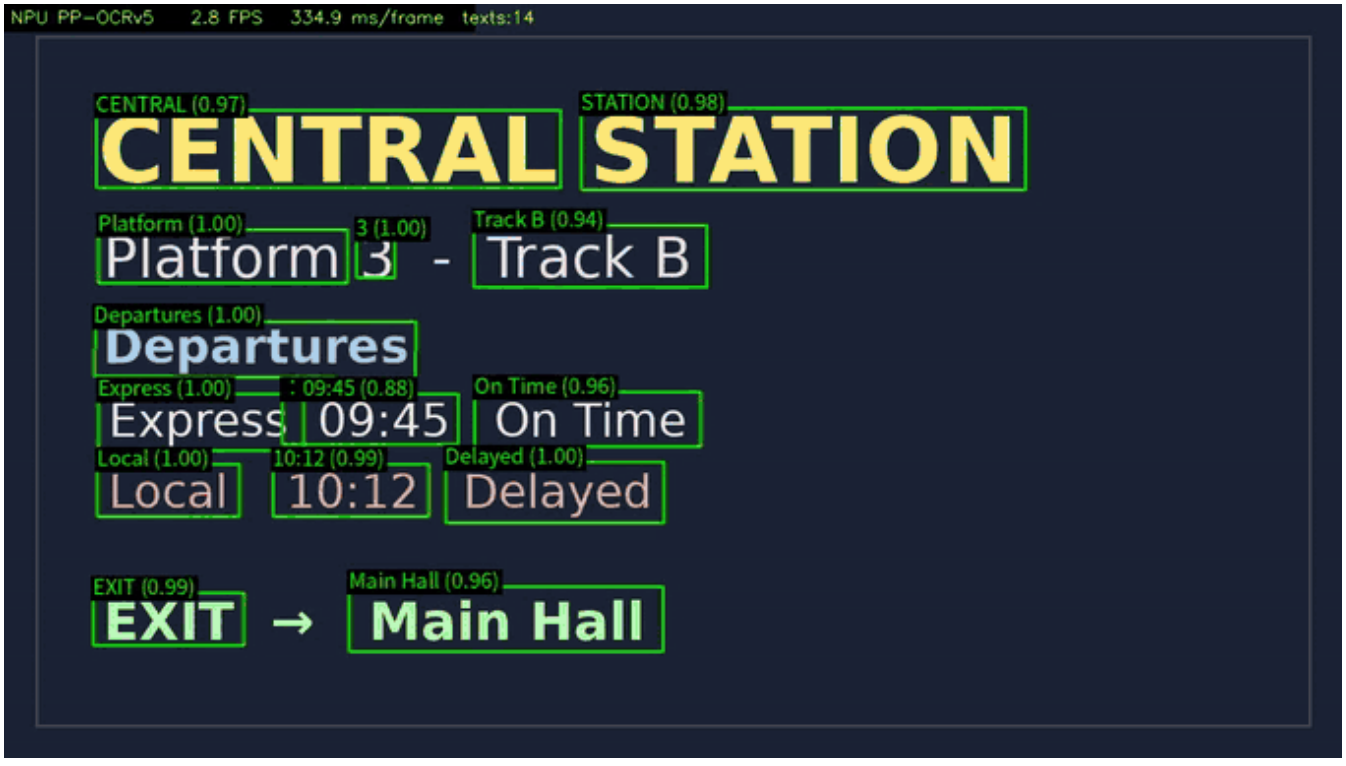
Highlight: mAP ~0.001→0.75 (mAP50 0.97), 55→78 FPS · **Claude Opus 4.8** · ≈ 8 min · ≈ $5.1 — details →

3.1.3 PaddlePaddle ecosystem integration

PaddleOCR (PP-OCRv5) on the DEEPX NPU — real-time video & webcam OCR from a single, concise prompt. Baidu's PaddlePaddle OCR (text detection → orientation → recognition) running on the DX-M1 NPU.

Showcase	Kind	Highlight
Video / Webcam OCR (PP-OCRv5)	app	PP-OCRv5 det→cls→rec on-device (~2.8 FPS, 341 ms/frame); 14 text regions/frame; video + webcam from one --source flag

Video / Webcam OCR (PP-OCRv5)



Real-time text detection + recognition on the DX-M1 NPU — one code path for a video file and a live webcam, overlaying detected boxes + recognized strings.

© Copyright 2026 DEEPX All Rights Reserved.

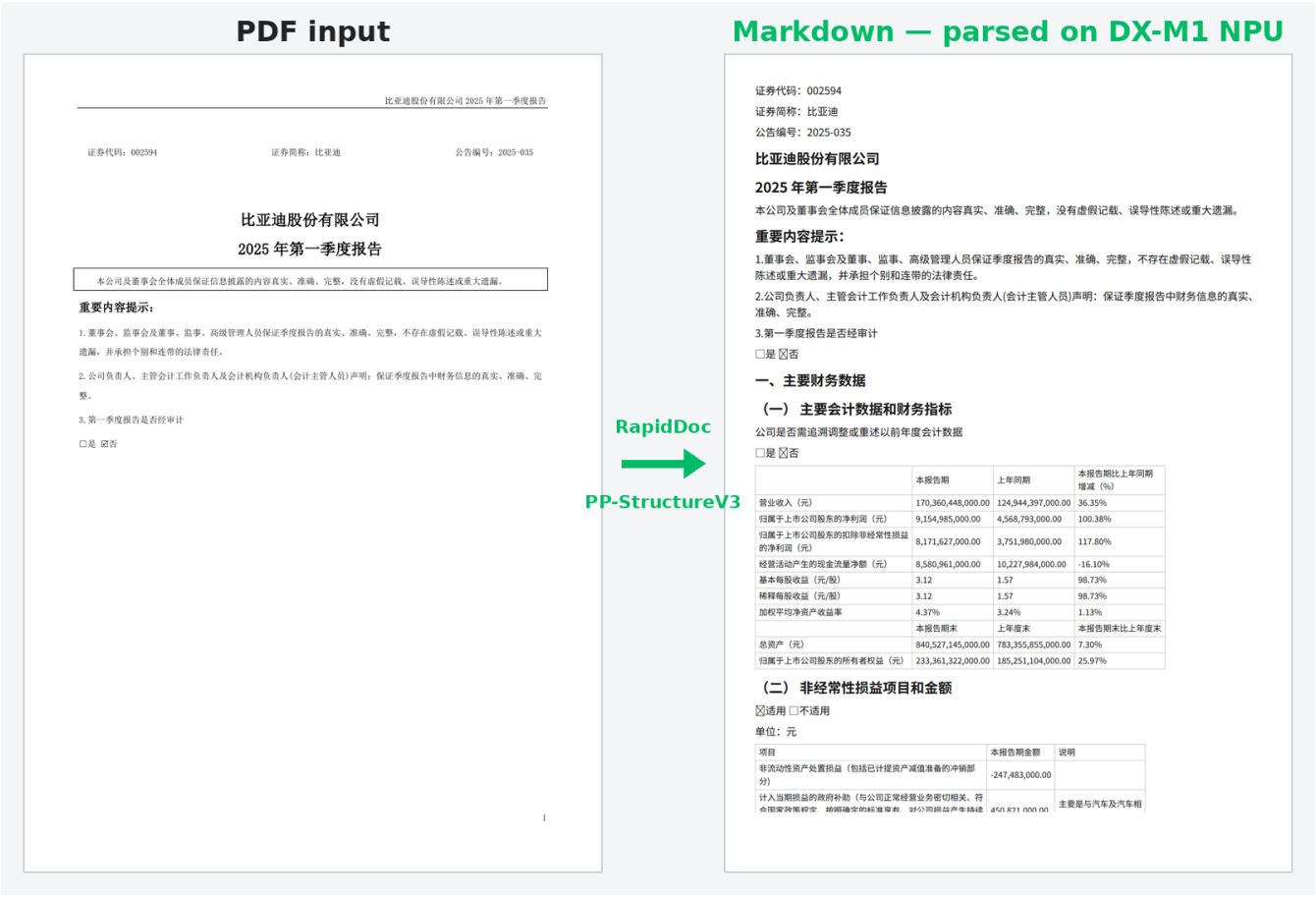
Highlight: PP-OCRv5 det→cls→rec on-device (~2.8 FPS, 341 ms/frame); 14 text regions/frame; video + webcam from one --source flag · **Claude Opus 4.8** · ≈ 18 min · ≈ $12.0 — details →

3.1.4 RapidAI ecosystem integration

A PDF → Markdown document-conversion app on the DEEPX NPU — from a single, concise natural-language prompt. RapidAI's RapidDoc (PP-StructureV3): layout, OCR, tables, formulas — running PaddlePaddle-trained models on the DX-M1 NPU. A standalone, self-contained app generated from the fork's pipeline.

Showcase	Kind	Highlight
PDF → Markdown (document conversion app)	app	9-page financial report parsed on-device; standalone app — vendored rapid_doc, own entry, no fork clone; 21 headings + 9 HTML tables preserved (auto 12.6s / ocr 14.7s)

PDF → Markdown (document conversion app)



ⓒ Copyright 2026 DEEPX All Rights Reserved.

Converts a PDF (digital or scanned) to structured Markdown + JSON — layout analysis, OCR, tables and formulas — on the DEEPX DX-M1 NPU via the RapidDoc fork. Supports `--parse-method auto|txt|ocr`.

Highlight: 9-page financial report parsed on-device; standalone app — vendored rapid_doc, own entry, no fork clone; 21 headings + 9 HTML tables preserved (auto 12.6s / ocr 14.7s) · **Claude Opus 4.8** · ≈ 12 min · ≈ $6.2 — details →

3.1.5 Reproduce any showcase

```
cd dx-agent-dev-showcase/<showcase>
bash setup.sh && bash run.sh        # retrain/export showcases
# games: ./setup.sh then ./run.sh (or ./run.sh --camera 0)
```

Requires x86-64 Linux + the DeepX runtime (`dx_engine`). Per-showcase prerequisites and the exact prompt are in each showcase's README.

> Korean: `README-ko.md`.

© Copyright 2026 DEEPX All Rights Reserved.

4. DX-AllSuite Architecture Overview

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

4.1 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.

4.2 System Architecture & Core Modules

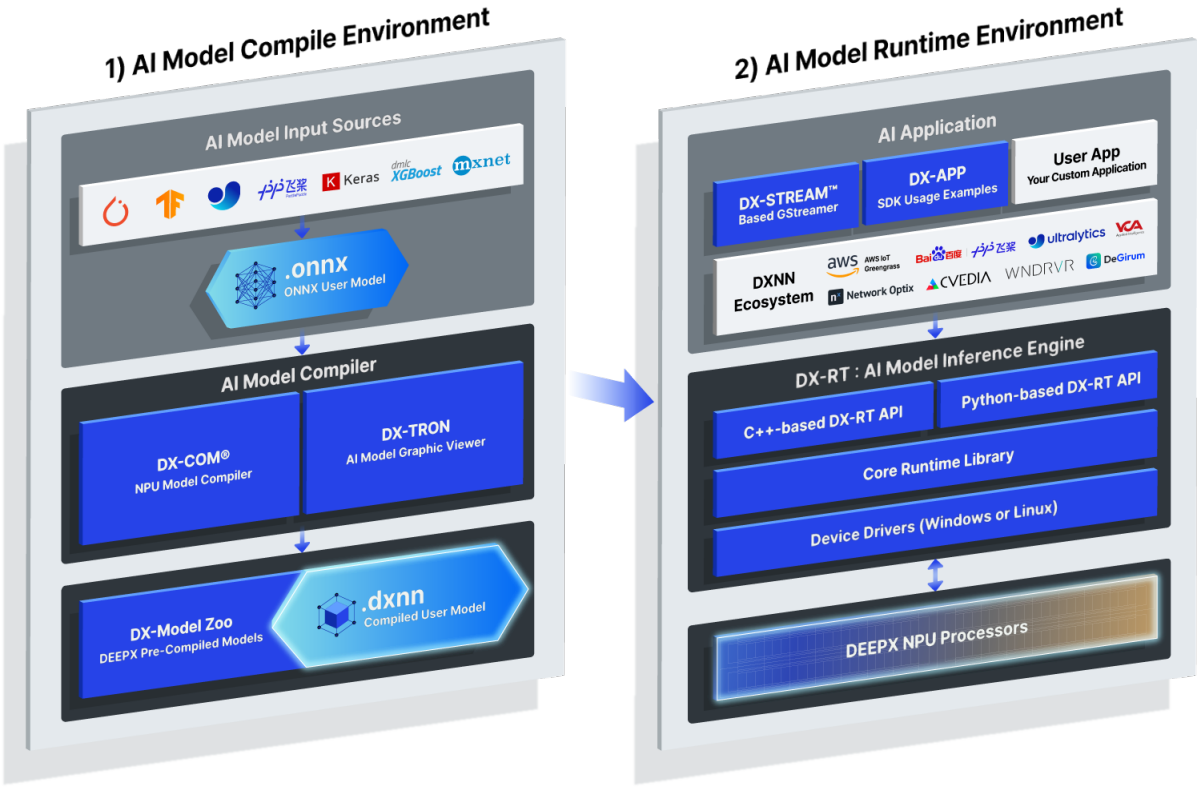


Figure. DXNN SDK Full Architecture Overview.

[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**.

4.3 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.

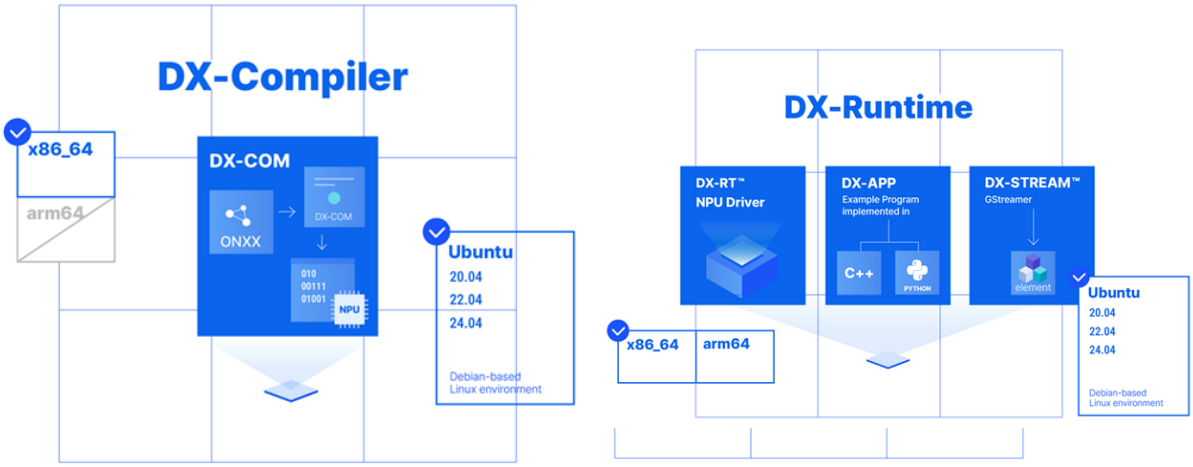


Figure. DX-Compiler & DX-Runtime Support Ecosystem.

4.3.1 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 26.04/24.04/22.04/20.04, Fedora, Redhat, CentOS	Ubuntu 26.04/24.04/22.04/20.04, Debian 13/12, Windows 11/10
Languages	Python 3.8, 3.9, 3.10, 3.11, 3.12, 3.13, 3.14	Python 3.8 or higher, C++14 or higher (C++17 for MSVC/ Windows)

4.3.2 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)

4.4 DEEPX ModelZoo & Supported Tasks

DEEPX ModelZoo is a comprehensive repository providing **345 pre-validated models**. It allows users to immediately verify performance and accuracy across various hardware profiles without the need for a manual compilation process.

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
3D Object Detection	3D detection models
Instance Segmentation	YOLACT, YOLOv5-Seg, YOLOv8-Seg, YOLO26-Seg
Semantic Segmentation	DeepLabV3/DeepLabV3+, SegFormer, BiSeNet, UNet
Panoptic Driving Perception	Panoptic perception models
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

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Task	Representative Models
Hand Detection	Hand detection models
Hand Landmark	MediaPipeHandsLite
Pose Estimation (Human)	CenterPose, YOLO26-Pose, YOLOv8-Pose
Object Pose Estimation	Object pose estimation models
Keypoint Detection	Keypoint detection models
Person Attribute	DeepMAR (ResNet18/50)
Person Re-identification	CasViT
Depth Estimation	FastDepth, SCDepthV3
Image Denoising	DnCNN variants
Low-light Enhancement	ZeroDCE
Super-resolution	ESPCN (x2/x3/x4)

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 2026 DEEPX All Rights Reserved.

5. 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.

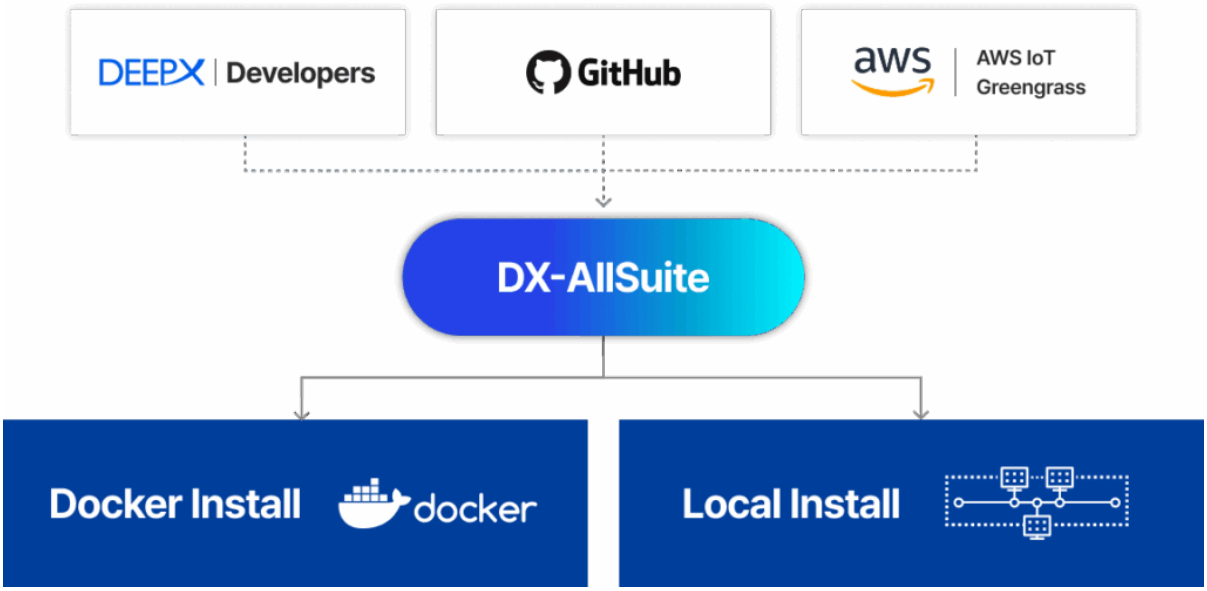


Figure. DX-AllSuite Supported Environments & Integrations.

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.

5.1 Prerequisites

Follow these steps first to ensure a stable installation.

© Copyright 2026 DEEPX All Rights Reserved.

5.1.1 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
```

5.1.2 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.

5.1.3 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.

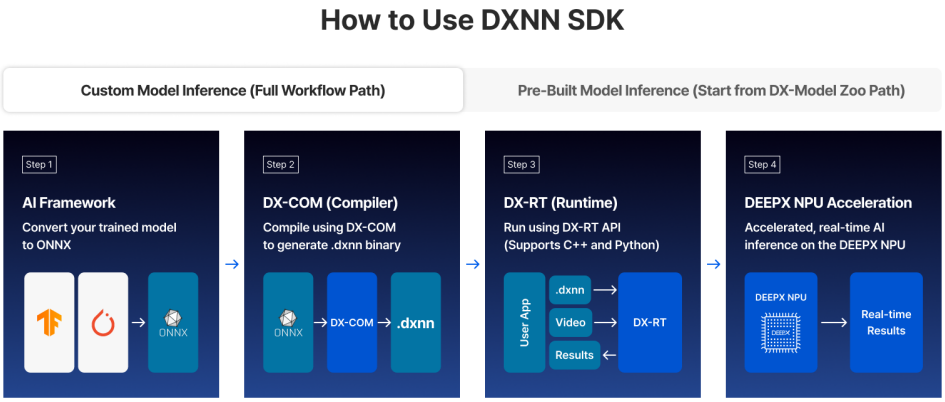


Figure. Custom Model Inference.

ⓒ 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.

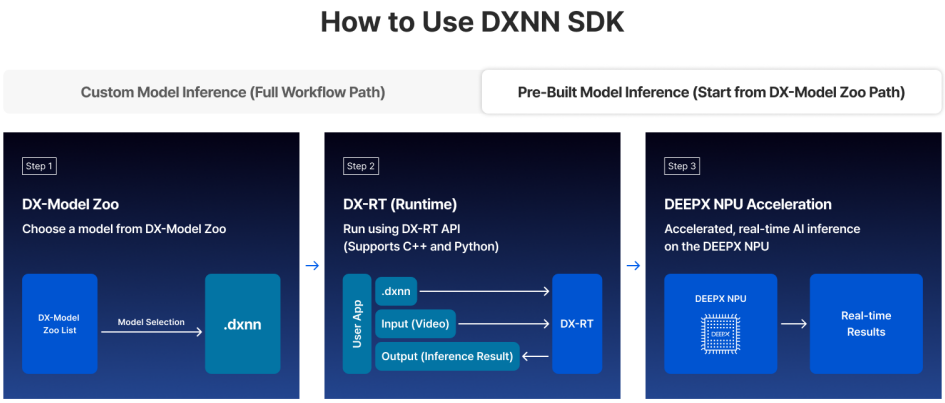


Figure. Pre-Built Model Inference.

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.

5.2 Docker Installation

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

5.2.1 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
```

5.2.2 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
```

> ⚠ **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**.

© Copyright 2026 DEEPX All Rights Reserved.

5.2.3 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`.

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
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

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
```

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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

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.

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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

5.2.4 [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"]
```

> **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.

5.2.5 [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
```


© Copyright 2026 DEEPX All Rights Reserved.

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)
```

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.

5.3 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.

5.3.1 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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

- **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.

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, 3.13, 3.14 (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
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

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/`

> **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**.

© Copyright 2026 DEEPX All Rights Reserved.

5.3.2 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)
./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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

5.3.3 [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
```

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 2026 DEEPX All Rights Reserved.

6. 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.

6.1 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

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
```


© Copyright 2026 DEEPX All Rights Reserved.

```
├── 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).

6.2 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.

6.2.1 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.

6.2.2 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.

6.2.3 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 skips reinstallation (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.

[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/`

© Copyright 2026 DEEPX All Rights Reserved.

> **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**: The generated `.dxnn` files are stored in `dx-compiler/dx_com/output/` and are immediately accessible via the `getting-started/dxnn` link.

6.3 DX-Runtime: 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.

6.3.1 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
```


© Copyright 2026 DEEPX All Rights Reserved.

6.3.2 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.

6.3.3 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).

© Copyright 2026 DEEPX All Rights Reserved.

[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.

[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 2026 DEEPX All Rights Reserved.

7. 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.

7.1 DXNN SDK Version Compatibility Matrix

Release Date	**DX-AllSuite**						
	DX-Compiler		**DX-Runtime**				
	DX-COM	**DX-TRON**	**DX-FW**	**NPU Driver**	**DX-RT**	**DX-Stream**	**DX-APP**
2026-07-28	v2.4.1						
	v2.4.1		v2.4.1				
	v2.4.0	v2.0.1	v2.7.3	v2.5.1	v3.4.1	v3.1.1	v3.2.1
2026-07-22	v2.4.0						
	v2.4.0		v2.4.0				
	v2.4.0	v2.0.1	v2.7.3	v2.5.1	v3.4.0	v3.1.0	v3.2.0
2026-05-14	v2.3.3						
	v2.3.1		v2.3.3				
	v2.3.0	v2.0.1	v2.5.6	v2.4.1	v3.3.2	v3.0.1	v3.1.1
2026-05-11	v2.3.2						
	v2.3.1		v2.3.2				
	v2.3.0	v2.0.1	v2.5.6	v2.4.1	v3.3.2	v3.0.1	v3.1.1
2026-05-06	v2.3.1						
	v2.3.1		v2.3.1				
	v2.3.0	v2.0.1	v2.5.6	v2.4.1	v3.3.1	v3.0.1	v3.1.1

ⓒ 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	v2.4.1	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
2025-07-23	v1.0.0						
	v1.0.0		v1.0.0				
	v1.60.1	v0.0.8	v2.1.0	v1.5.0	v2.9.5	v1.7.0	v1.11.0

ⓒ Copyright 2026 DEEPX All Rights Reserved.

7.2 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 2026 DEEPX All Rights Reserved.

8. 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
- Q6. Insufficient Shared Memory (shm) Error

8.1 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.

8.1.1 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 a
go  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.

8.1.2 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.

8.1.3 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** section 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 (Multi-Runtime Containers)** section in **02. Setting Up Environment**.

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

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

8.2.1 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.

8.2.2 Cause

- **Q2.1:** GUI tools such as **DX-TRON** are optimized for X11. Wayland sessions can cause `xauth` processes to be unstable.

© Copyright 2026 DEEPX All Rights Reserved.

- **Q2.2:** Ending a Wayland session may purge X authentication data or change directory paths, causing Docker mount points to become invalid.

8.2.3 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**.

8.3 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.

© Copyright 2026 DEEPX All Rights Reserved.

8.3.1 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.
```

8.3.2 Cause

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

8.3.3 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.

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

ⓒ Copyright 2026 DEEPX All Rights Reserved.

updated successfully.

```
dxrt-cli -s
```

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

8.4 Q4. Device Driver Update Error

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

8.4.1 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.
```

8.4.2 Cause

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

8.4.3 Solution: Update the Device Driver on the Host OS

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

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
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

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 **Building and Installing Modules** section in **02. Setting Up Environment**.

8.5 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.

8.5.1 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.
```

8.5.2 Cause

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

8.5.3 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.

© Copyright 2026 DEEPX All Rights Reserved.

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 Version Compatibility Matrix** in **04. Version Compatibility**.

Copyright © DEEPX. All rights reserved.

8.6 Q6. Insufficient Shared Memory (shm) Error

This error occurs when Docker's default `/dev/shm` size is too small for **DX-COM** (the DEEPX compiler) during model compilation.

8.6.1 Diagnostic Steps

Check the terminal for the following error message.

```
[ResourceError]: Insufficient shared memory (shm). Increase the available /dev/shm
size (e.g. --shm-size for Docker).
```

8.6.2 Cause

Docker containers are allocated a default `/dev/shm` of **64 MB**. **DX-COM** uses shared memory during the model compilation process. When compiling large or complex models, this default limit can be exceeded.

8.6.3 Solution: Increase the Docker Shared Memory Size

Method 1: `docker run` with `--shm-size` (Recommended)
Pass the `--shm-size` flag when starting the container. Set the value based on your model size — `256m` is a common starting point; increase to `1g` or more for large models.

```
docker run --shm-size=256m ...
# For larger models
docker run --shm-size=1g ...
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

Method 2: `docker-compose.yml`

If you manage containers with Docker Compose (e.g., via `docker_run.sh`), add the `shm_size` field to the `dx-compiler` service definition.

```
services:
  dx-compiler:
    shm_size: '1gb'
```

Choosing the Right Size

A value of `256m` resolves most cases. If the error persists with larger or batch-processed models, increase to `1g` or higher.

© Copyright 2026 DEEPX All Rights Reserved.