DXNN All Suite (DX-AllSuite) User Manual

2026/10/23 (v2.5.0)

DEEPX.ai

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Table of contents

1. Introduction 5
1.1 Overview 5
1.2 DX-AS (DEEPX All Suite) Environments and Installation Guide 8
1.3 Documentation 9
2. DEEPX Agent-Driven Development - dx-agent-dev (Beta) 10
2.1 Introduction 10
2.2 Showcases 10
2.3 Prerequisites 15
2.4 Architecture Overview 15
2.5 Available Agents and Skills 16
2.6 Supported AI Tools 20
2.7 Quick Start by Tool 24
2.8 End-to-End Scenarios 26
2.9 Cross-Project Routing 30
2.10 Sub-Project Guides 30
2.11 Internal Reference Documents 31
2.12 Output Isolation 31
2.13 Session Sentinels 32
3. Agent-Driven Development Showcases 34
3.1 dx-agent-dev Showcases 34
4. DX-AllSuite Architecture Overview 42
4.1 Why DX-AllSuite? 42
4.2 System Architecture & Core Modules 42
4.3 Technical Compatibility (Support Matrix) 44
4.4 DEEPX ModelZoo & Supported Tasks 46
5. Setting Up Environment 48
5.1 Quick Install (one-line) 48
5.2 APT Repository Install (DX-Runtime) 50
5.3 Prerequisites 52
5.4 Local Installation 55
5.5 Docker Installation 61
6. Running Your First NPU Model 73
6.1 Overall Pipeline 73
6.2 DX-Compiler: AI Model Compilation Scripts Guide (Steps 0–4) 74
6.3 DX-Runtime: Application Execution Scripts Guide (Step 0, 5) 76

ⓒ Copyright 2026 DEEPX All Rights Reserved.

6.4 DX-ModelZoo: Model Evaluation Scripts Guide 79
7. Version Compatibility 82
7.1 DXNN SDK Version Compatibility Matrix 82
7.2 How to Check Your System Version 84
8. FAQ Troubleshooting Guide 86
8.1 Q1. Container 'Restarting' Error (dxrtd Conflict) 86
8.2 Q2. X11 Session Warnings & Mount Errors (Wayland Issues) 87
8.3 Q3. Firmware Version Mismatch Error 89
8.4 Q4. Device Driver Update Error 90
8.5 Q5. Model-Runtime Version Compatibility Error 91
8.6 Q6. Insufficient Shared Memory (shm) Error 92
8.7 Q7. DX-ModelZoo Environment Error (`dxmz` / `dx_engine` / `dx_com` Not Found) 93
8.8 Q8. DX-ModelZoo Model Download Timeout 95
8.9 Q9. DX-ModelZoo Dataset Not Found 96
8.10 Q10. Docker Mount Error After Recreating a Host Path 97
8.11 Q11. NPU Device Driver Version Is Too Old 98
9. DX AI Studio 100
9.1 Installation & Launch 100
9.2 The Hub 108
9.3 DX Model Zoo 112
9.4 DX Compiler 115
9.5 DX App 119
9.6 DX Stream 126
9.7 DX Monitor 130
9.8 DX Benchmark 133
9.9 DX EdgeGuide 136
9.10 DX Agent Dev 140
9.11 DX Cloud (AWS) 143
9.12 SDK Library & About 151
9.13 Third-Party License Notice 155
10. Kubernetes Quickstart 157
10.1 Architecture 157
10.2 Prerequisites 158
10.3 Install 159
10.4 Run a workload 159
10.5 Metrics 160
10.6 Troubleshooting 160

ⓒ Copyright 2026 DEEPX All Rights Reserved.

11. Kubernetes Beginner Guide — Learning with the DX-M1 NPU 162
11.1 Part 1. The picture 162
11.2 Part 2. Try it yourself 164
11.3 Part 3. Common snags 168
11.4 Glossary 169

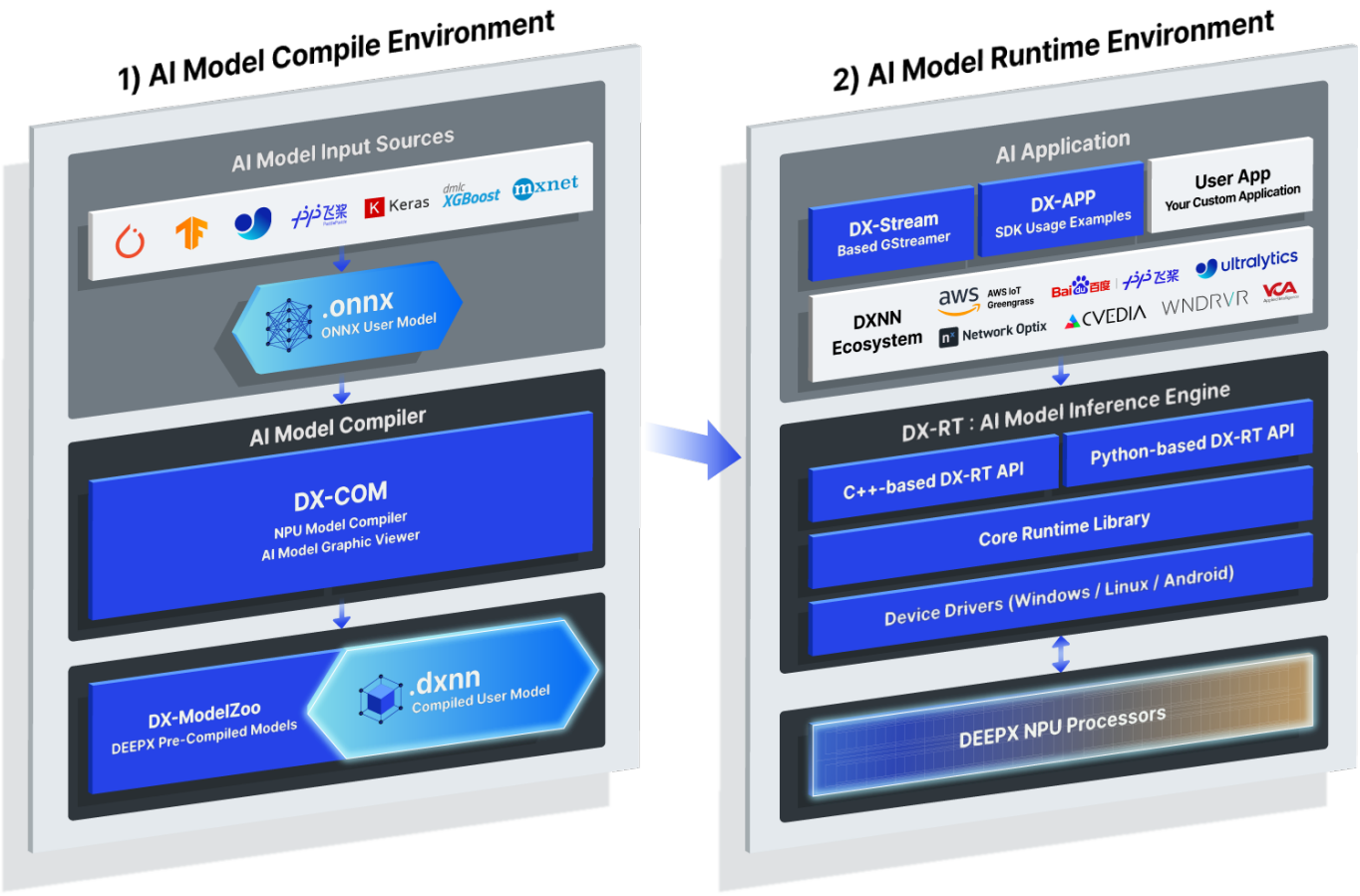
ⓒ Copyright 2026 DEEPX All Rights Reserved.

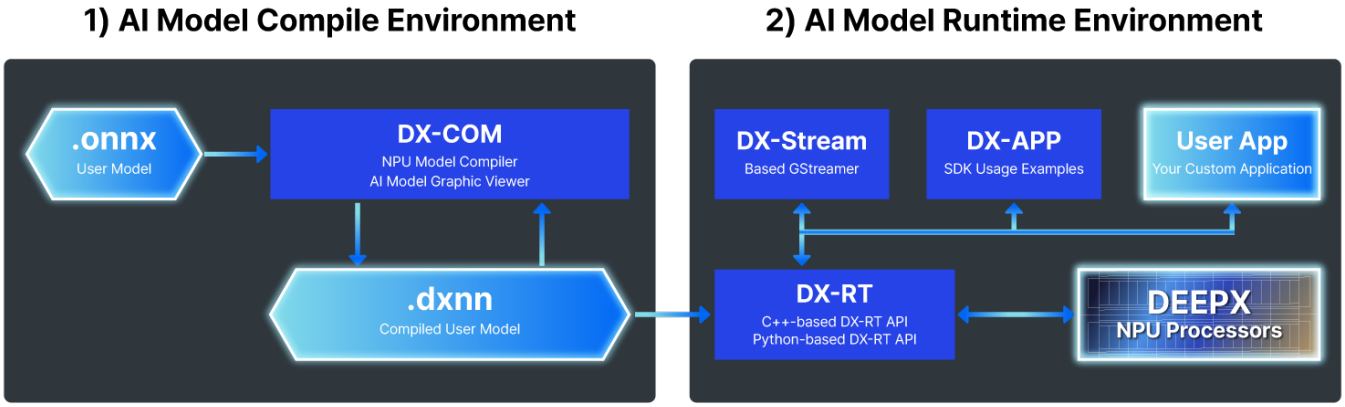
1. Introduction

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 DX-COM (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: Ubuntu 26.04/24.04/22.04/20.04, Fedora 42–45, RHEL 9–10, CentOS Stream 9–10
- Architecture: Supports x86_64 only
- Languages: Python 3.8 – 3.14

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: Ubuntu 26.04/24.04/22.04/20.04, Debian 13/12, Windows 11/10
- Architecture: Supports both x86_64 and aarch64
- Languages: Python 3.8 or higher, C++14 or higher (C++17 for MSVC/Windows)

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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

1.3 Documentation

1.3.1 Agent-Driven Development (Beta)

Link

1.3.2 Agent-Driven Development Showcases

Link

1.3.3 Architecture Overview

Link

1.3.4 Installation Guide

Link

1.3.5 Getting Started

Link

1.3.6 Version Compatibility

Link

1.3.7 FAQ & Troubleshooting

Link

1.3.8 Kubernetes Quickstart

Link

1.3.9 Kubernetes Beginner Guide

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 across multiple AI ecosystems (**Ultralytics**, **PaddlePaddle**, **RapidAI**, and more), 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 5 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,

© Copyright 2026 DEEPX All Rights Reserved.

measured results, a one-command reproduce, and the full recorded build-session transcript.

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 →	Turns an Ultralytics	≈ 12 min	108	≈ 84K	≈ $2.4

© Copyright 2026 DEEPX All Rights Reserved.

Showcase	What it is	Build time	Agent turns	Output tokens	~Cost
DeepX Export	YOLO `.pt` into a deployable DeepX NPU model (`.dxnn`) in a single `yolo export ... format=deepx` 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	≈ 9 min	91	≈ 103K	≈ $3.7

© Copyright 2026 DEEPX All Rights Reserved.

Showcase	What it is	Build time	Agent turns	Output tokens	~Cost
	medical edge device; 4-way eval base/ retrained × fp32/INT8.				
Pharmaceutical Pill Inspection	Retrains `yolo26n` on `medical-pills` for a pharma counting station; 4-way eval base/ retrained × fp32/INT8.	≈ 8 min	118	≈ 103K	≈ $5.1

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 +	≈ 18 min	175	≈ 184K	≈ $12.0

© Copyright 2026 DEEPX All Rights Reserved.

Showcase	What it is	Build time	Agent turns	Output tokens	~Cost
	recognized strings.				

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	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

ⓒ Copyright 2026 DEEPX All Rights Reserved.

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.

2.3 Prerequisites

Requirement	Details
DEEPX development environment	DX-RT installed (dx-runtime `install.sh`)
AI coding agent (one of)	Claude Code, GitHub Copilot (VS Code), Cursor, OpenCode, or Codex CLI
Python	3.8–3.14 with the dx-all-suite packages installed

2.4 Architecture Overview

The agent-driven knowledge base is organized into four 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

© Copyright 2026 DEEPX All Rights Reserved.

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.

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

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Level	Agent	Description
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
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-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

Level	Skill	Description
dx-runtime	`/dx-agent-runtime-validate`	Validate, collect feedback, apply fixes, verify

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Level	Skill	Description
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
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

© Copyright 2026 DEEPX All Rights Reserved.

Level	Skill	Description
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
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

ⓒ Copyright 2026 DEEPX All Rights Reserved.

> **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`	`@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)

© Copyright 2026 DEEPX All Rights Reserved.

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`
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
```


© Copyright 2026 DEEPX All Rights Reserved.

```
# 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.

> **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`

© Copyright 2026 DEEPX All Rights Reserved.

Agent	Copilot (@mention)	OpenCode (@mention)
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.

Skill Files (/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

© Copyright 2026 DEEPX All Rights Reserved.

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/`	Domain, process, and harness skills
`templates/`	`{en,ko}/*.tmpl` — instruction file templates
`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.

```
pipx install --force --editable .deepx/tools   # PEP 668-safe; 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)

ⓒ Copyright 2026 DEEPX All Rights Reserved.

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

Prompt:

```
"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"`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Submodule	Example Prompt
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 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

© Copyright 2026 DEEPX All Rights Reserved.

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

Prompt:

> "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:

ⓒ Copyright 2026 DEEPX All Rights Reserved.

> "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 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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Tool	How to Use
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.
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 (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.

© Copyright 2026 DEEPX All Rights Reserved.

Tool	How to Use
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-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	

© Copyright 2026 DEEPX All Rights Reserved.

Sub-Project	Guide
	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

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

© Copyright 2026 DEEPX All Rights Reserved.

Output Type	Path	When
Production	`src/`	Only when explicitly requested by the user

Session ID format: `YYYYMMDD-HHMMSS_<agent>_<coding_model>_<target_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.

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

© Copyright 2026 DEEPX All Rights Reserved.

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.

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.

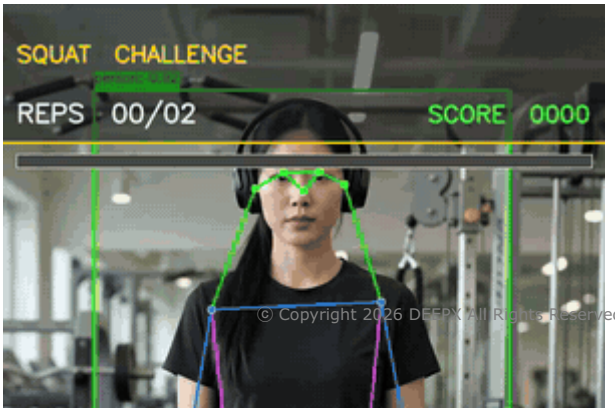
3.1.1 NPU-powered AI apps (mini-games)

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!).



© Copyright 2026 DEEPX All Rights Reserved.

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

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

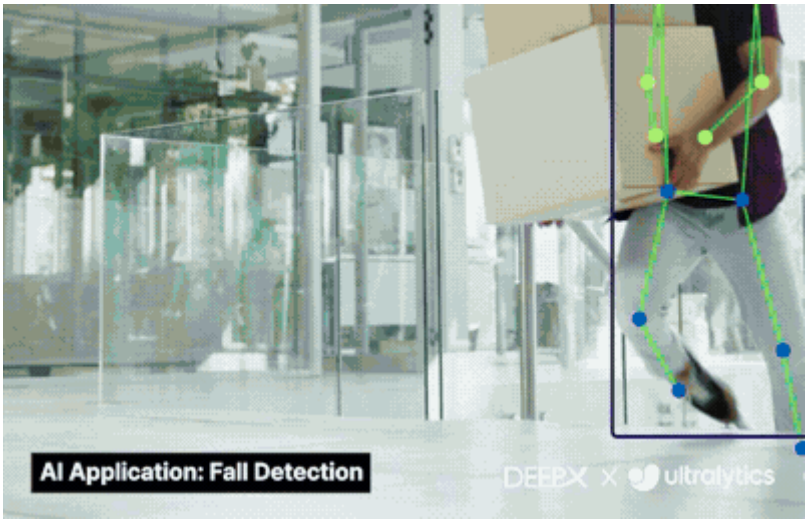
© Copyright 2026 DEEPX All Rights Reserved.

Showcase	Kind	Highlight
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

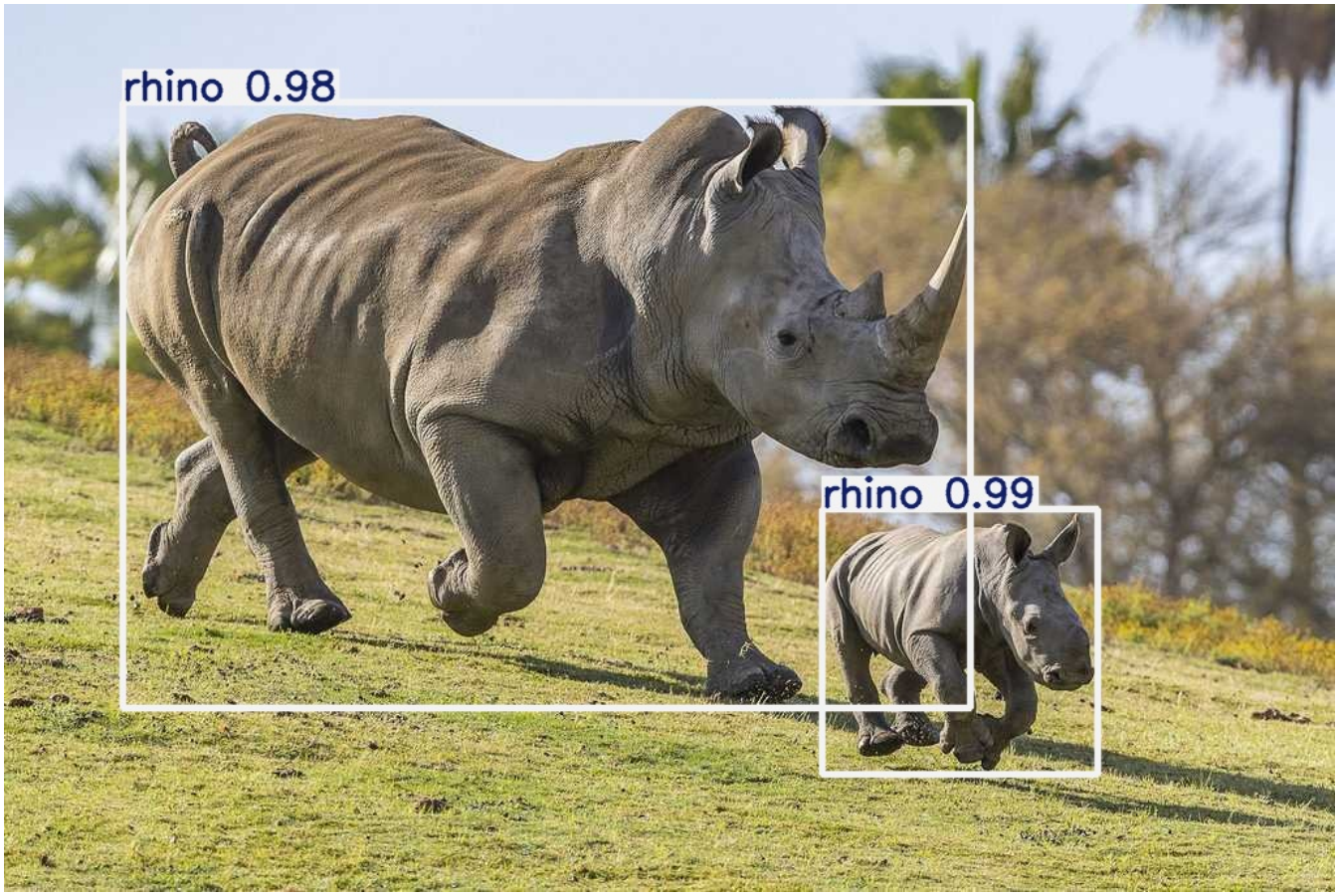
Ultralytics YOLO → DeepX Export

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

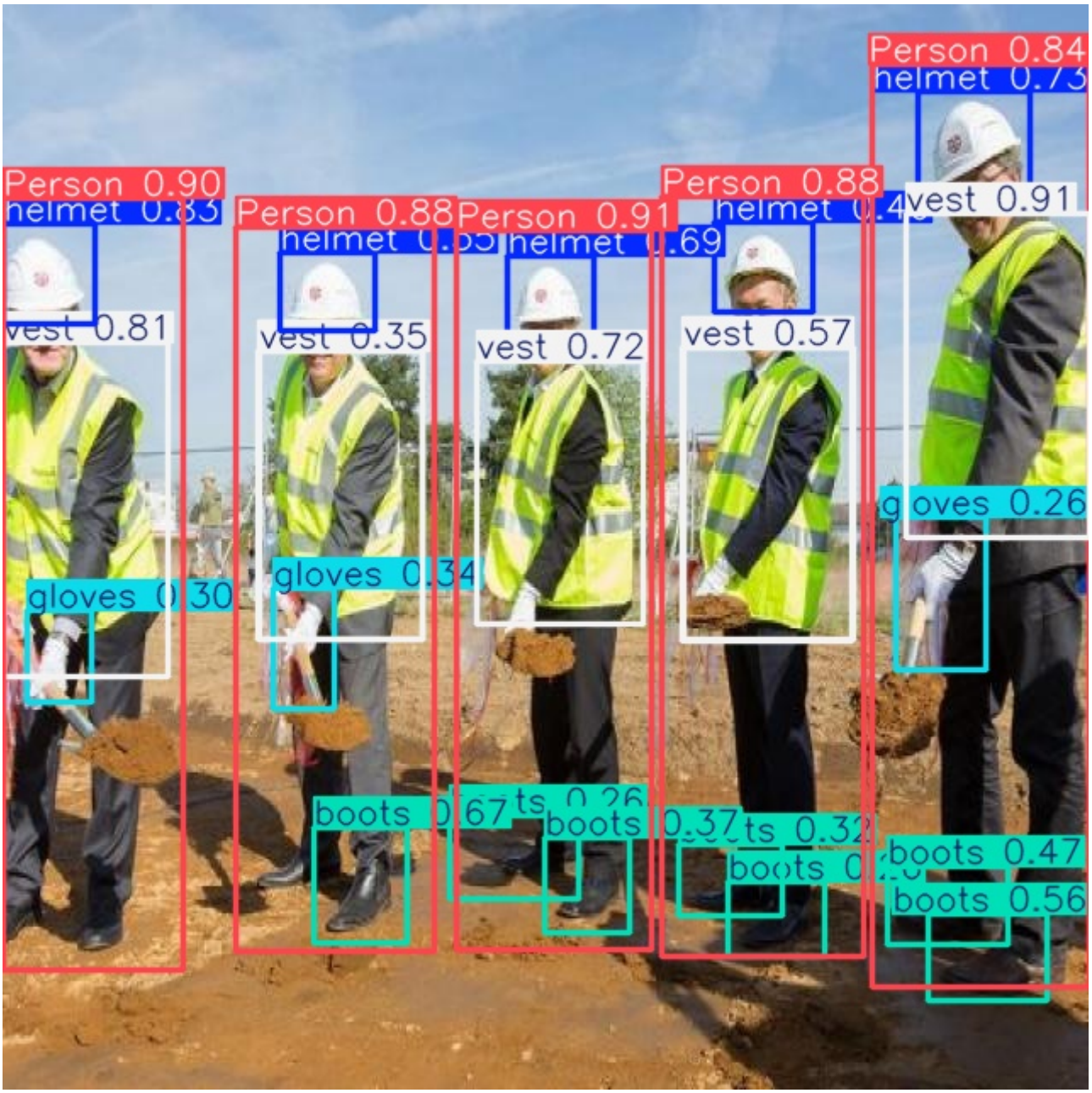


ⓒ Copyright 2026 DEEPX All Rights Reserved.

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



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

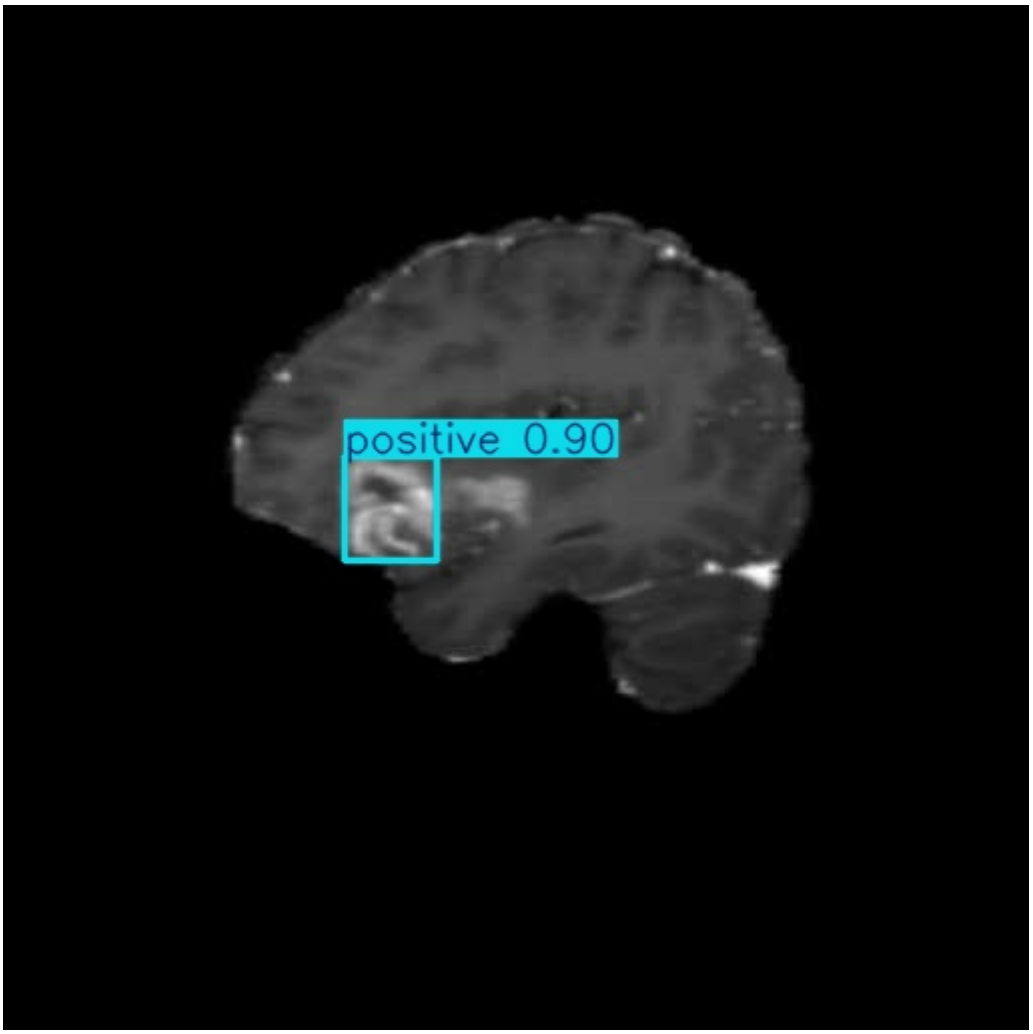
Highlight: mAP 0.0001→0.257, 58→80 FPS · **Claude Opus 4.8** · ≈ 17 min · ≈ $4.0 — details →

© Copyright 2026 DEEPX All Rights Reserved.

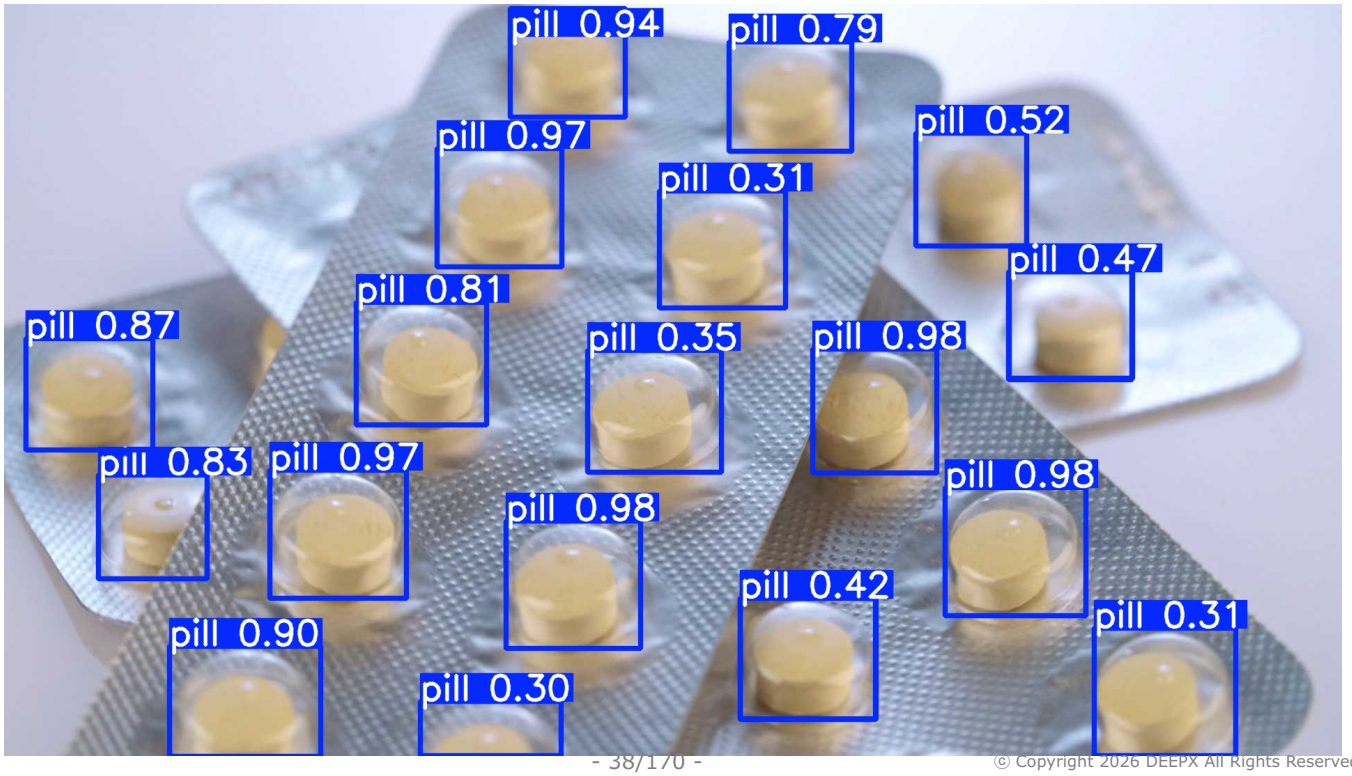
Brain-Tumor Screening

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 DEEPX 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.

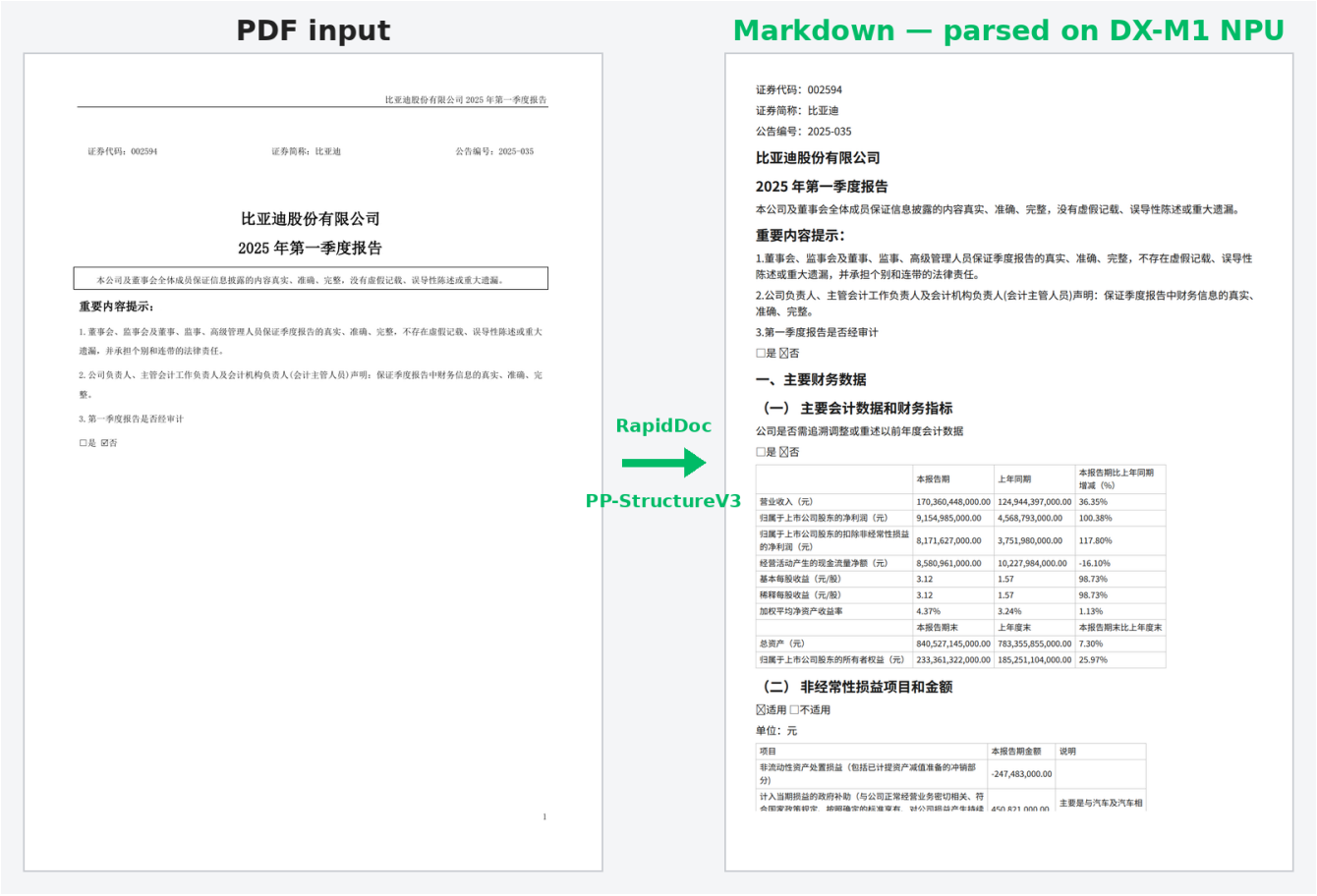
Highlight: PP-OCRv5 det→cls→rec on-device (~2.8 FPS, 341 ms/frame); 14 text regions/frame; video + webcam from one --source flag · **Claude Opus 4.8** · ≈ 18 min · ≈ $12.0 — details →

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.

4.2 System Architecture & Core Modules

ⓒ Copyright 2026 DEEPX All Rights Reserved.

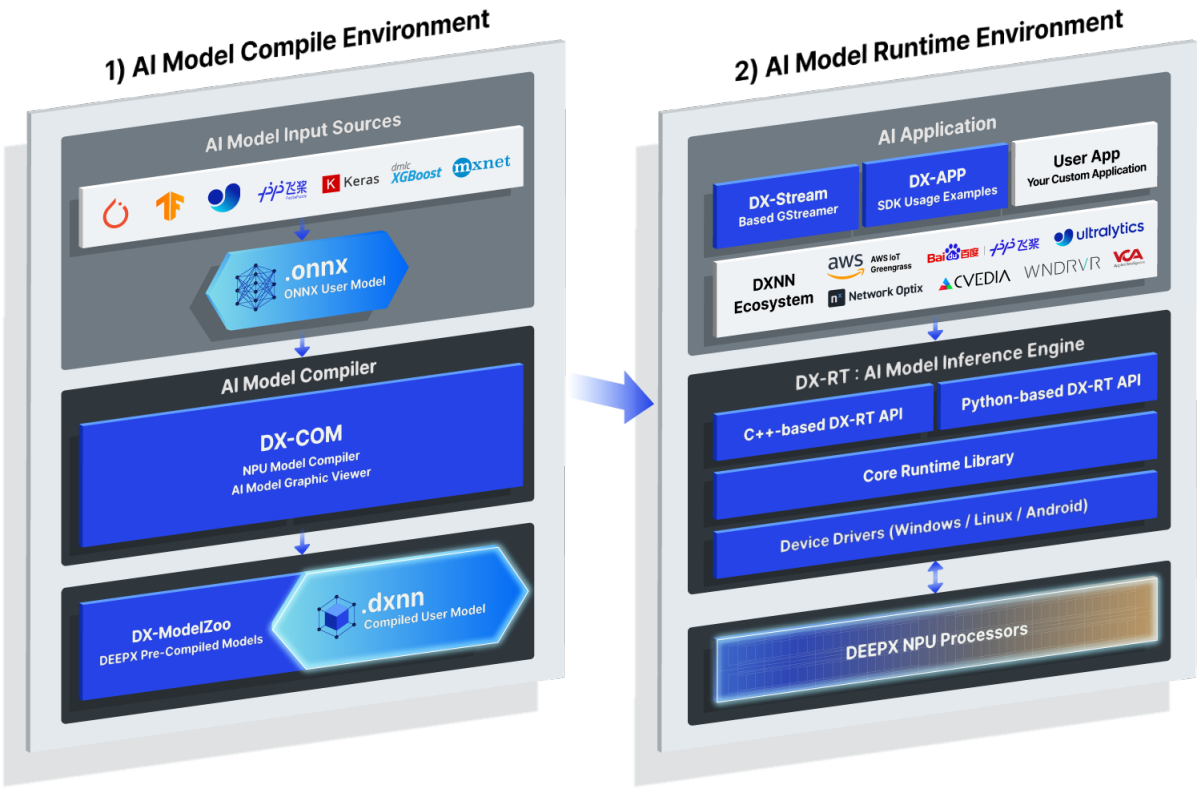


Figure. DXNN SDK Full Architecture Overview.

[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.
- **Compilation Summary Report**: An interactive HTML report generated by DX-COM (`--export_html`). It visualizes the compiled model graph with CPU/NPU partitioning and partition reasons, 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.

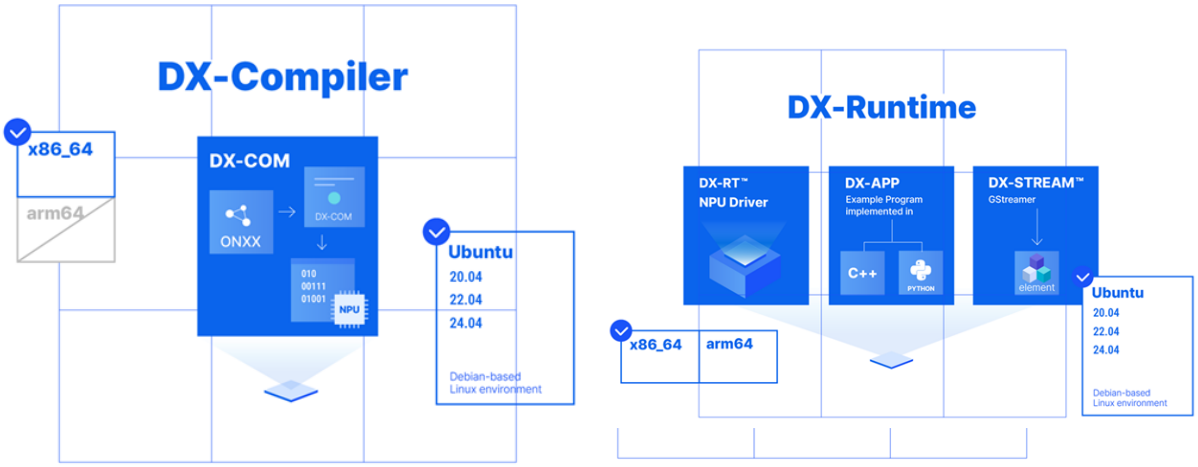


Figure. DX-Compiler & DX-Runtime Support Ecosystem.

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 42–45, RHEL 9–10, CentOS Stream 9–10	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 **300+ 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, FastViT, CasViT, RegNet, ShuffleNet, VGG
Object Detection	YOLO families (YOLOv3–YOLOv13, YOLOX, YOLO26), RT-DETR (v1/v2/v3), 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, ULFGFD, YOLOv5-Face, YOLOv7-Face

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Task	Representative Models
Face Recognition	ArcFace (IResNet50/100, MobileFaceNet, R50)
Face Landmark	3DDFA_V2 (MobileNet variants)
Face Attribute	FaceAttrResNetV1-18
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	RepVGG
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.

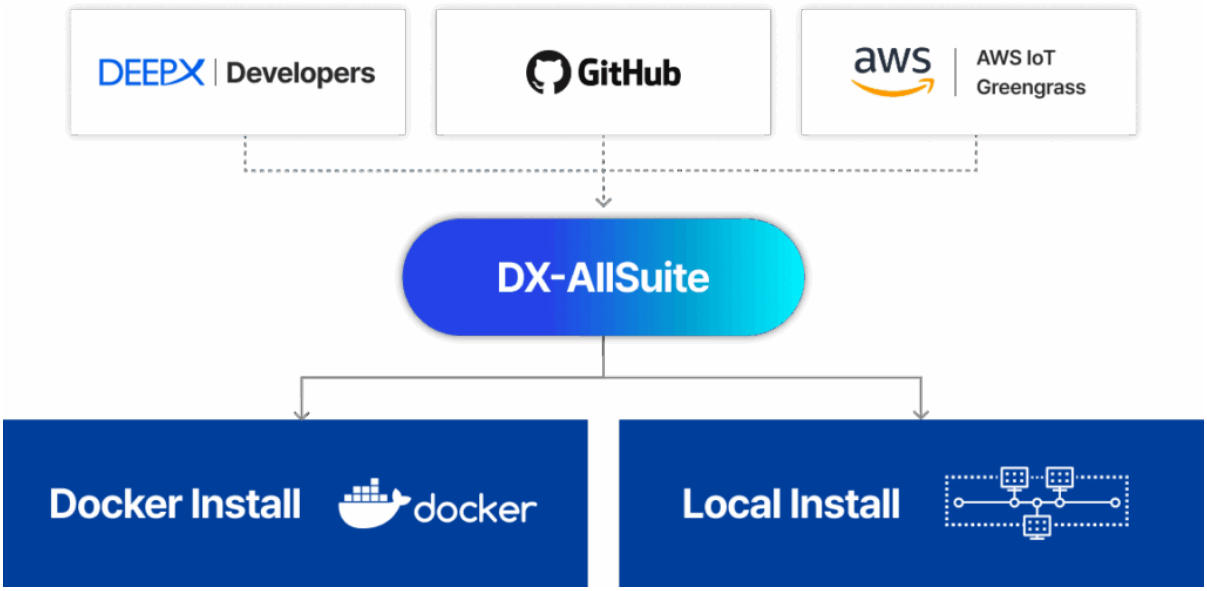


Figure. DX-AllSuite Supported Environments & Integrations.

Installation Summary

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

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

5.1 Quick Install (one-line)

If you only need a single component, install it directly — no cloning, no submodule synchronization:

```
# DX-Compiler (x86_64 Host PC)
curl -fsSL https://raw.githubusercontent.com/DEEPX-AI/dx-compiler/main/oneline-install.sh | sh
```


© Copyright 2026 DEEPX All Rights Reserved.

```
# DX-Runtime (target device with a DEEPX NPU: NPU driver + dx_rt + firmware)
curl -fsSL https://raw.githubusercontent.com/DEEPX-AI/dx-runtime/main/oneline-install.sh | sh
```

Each installer resolves the newest version on its own — DX-Compiler from its latest release on PyPI, DX-Runtime from the `latest` pointer on each component's `main` branch. Override with `DX_VERSION=2.4.1` (DX-Compiler) or, for DX-Runtime, per component with `DX_RT_VERSION=3.4.2`, `DX_DRIVER_VERSION=2.6.0`, `DX_FW_VERSION=2.7.4`. DX-Compiler also accepts `DX_INSTALL_DIR=<dir>` (default `~/deepx`) to move the install root; DX-Runtime has no equivalent, because it installs system packages through `dpkg` rather than into a directory you choose.

DX-Runtime tracks `main`, so runs are not reproducible by default

Because versions are resolved at run time, the downloaded artifacts are not checksum-verified, and the three components follow their own branches independently — a run made between dx-runtime releases can install a combination that has not been validated together. Pin the versions above, or use the repository's `install.sh`, whenever you need a known-good set.

DX-Compiler needs root for its system packages

`dx-com` is installed from PyPI into `$DX_INSTALL_DIR/venv-dx-compiler` (default `~/deepx/venv-dx-compiler`), and the `dxcom` launcher is linked into `DX_BIN_DIR` (default `~/.local/bin`) so it runs without activating the venv. The installer also apt-installs `libgl1-mesa-dev` and `libglib2.0-0`: `dx-com` depends on `opencv-python`, whose `cv2` extension links against those libraries and fails to import without them. Run as root or with `sudo` available, on Debian or Ubuntu. This route does not install the sample data — clone the repository for that.

DX-Runtime: reboot required

A reboot is mandatory after installation so the kernel loads the NPU driver. When no NPU device is detected the firmware update step is skipped — rerun the same command after rebooting to finish it.

© Copyright 2026 DEEPX All Rights Reserved.

Scope of one-line install

One-line install covers `dx_fw`, `dx_rt`, and `dx_rt_npu_linux_driver` for DX-Runtime. `dx_app`, `dx_stream`, source builds, and the Docker route all use the full setup described below.

5.1.1 Uninstalling a one-line install

```
# DX-Compiler
curl -fsSL https://raw.githubusercontent.com/DEEPX-AI/dx-compiler/main/oneline-uninstall.sh | sh

# DX-Runtime
curl -fsSL https://raw.githubusercontent.com/DEEPX-AI/dx-runtime/main/oneline-uninstall.sh | sh
```

DX-Compiler must be given the same `DX_INSTALL_DIR` / `DX_BIN_DIR` used at install time, otherwise it looks in the default location and finds nothing. DX-Runtime takes no such argument: both of its install routes produce the same Debian packages, so this command removes an `install.sh` install as well.

What uninstalling cannot undo

Firmware already flashed to the NPU is not reverted — there is no uninstall path for it. The `dx_engine` Python wheel is also left in place: `libdxrt-bin` cannot know which interpreter or virtualenv it went into, so it prints removal instructions during its own purge instead. `dx_app` and `dx_stream` are untouched, since the one-liner never installs them; use the repository's `uninstall.sh` for those.

5.2 APT Repository Install (DX-Runtime)

The NPU driver and the DX-RT runtime are also published as Debian packages in the DEEPX APT repository, for Ubuntu and Debian on `amd64` and `arm64`. Use this route when you want them managed by `apt` like any other system package — installed, upgraded and removed with the usual commands.

```
# 1. Download the DEEPX signing key and check its fingerprint
wget -O deepx-archive-keyring.asc https://apt.releases.deepx.ai/gpg
gpg --show-keys deepx-archive-keyring.asc

# 2. Only if the fingerprint matches, install the key and remove the download
sudo gpg --dearmor -o /usr/share/keyrings/deepx-archive-keyring.gpg deepx-archive-keyring.asc
rm deepx-archive-keyring.asc
```


© Copyright 2026 DEEPX All Rights Reserved.

```
# 3. Add the repository for this Ubuntu / Debian release
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/deepx-archive-keyring.gpg] https://apt.releases.deepx.ai $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") main" | sudo tee /etc/apt/sources.list.d/deepx.list

# 4. Install the driver and the runtime
sudo apt update
sudo apt install dxrt-driver-dkms
sudo apt install libdxrt-bin
```

Verify the key fingerprint before step 2

To guard against a tampered key (e.g. a man-in-the-middle attack), check the output of `gpg --show-keys` before installing the key. It must show exactly this 40-character fingerprint — `3A20CC853C64AE328D0F58CFD816AAC6689DBDEA`:

```
pub   rsa4096 2026-08-11 [SCEA]
      3A20CC853C64AE328D0F58CFD816AAC6689DBDEA
uid                 DeepX APT Repo <jhsa@deepx.ai>
```

If it differs, stop, delete `deepx-archive-keyring.asc`, and contact DEEPX support.

Package	Contents
`dxrt-driver-dkms`	NPU kernel driver, built for the running kernel through DKMS (pulls in `dkms` and `build-essential`) and rebuilt automatically on kernel updates
`libdxrt-bin`	Pre-built DX-RT library and headers under `/usr/local`, `dxcli` and the other runtime tools, and the `dxrt` service

CLI command names

Since DX-RT v3.4.0, `dxrt-cli`, `parse_model`, and `run_model` have been renamed to `dxcli`, `dxparse`, and `dxrun`. The old names remain available as backward-compatible aliases.

© Copyright 2026 DEEPX All Rights Reserved.

Reboot and firmware

Reboot after installing `dxrt-driver-dkms` so the kernel loads the NPU driver. Firmware is not shipped in the APT repository — update it with the one-line installer above or `./dx-runtime/install.sh --target=dx_fw`, then cold boot as described in **DX-Runtime Installation**.

Install the `dx_engine` Python package

`libdxrt-bin` does not install the `dx_engine` Python binding, since it cannot know which interpreter or virtualenv you use. Wheels matching the installed `libdxrt-bin` (one per CPython ABI) are staged in `/usr/share/libdxrt-bin/python`; install the one for your Python version into your virtualenv. The package prints the exact `pip install` command after installation.

Upgrade to later releases with `sudo apt update && sudo apt upgrade`, and remove the packages with `sudo apt purge libdxrt-bin dxrt-driver-dkms`. This route covers the driver and runtime only; `dx_app` and `dx_stream` still use the full setup below.

5.3 Prerequisites

Follow these steps first to ensure a stable installation. This is the full-suite route — skip it if the one-line install above already covers what you need.

5.3.1 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

© Copyright 2026 DEEPX All Rights Reserved.

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
```

5.3.2 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`
- **ModelZoo Environment**: DX-ModelZoo is installed into the runtime environment (`dx-runtime/venv-dx-runtime`) so that `dxmz` can use `dx_engine` for NPU evaluation. In the Docker image, it is pre-installed in `/venv-dxnn`.

> ⚠ **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.

5.3.3 SDK Workflow Guide

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

© Copyright 2026 DEEPX All Rights Reserved.

[Path A] Custom Model Inference Path

The standard workflow for converting and deploying user-trained models.

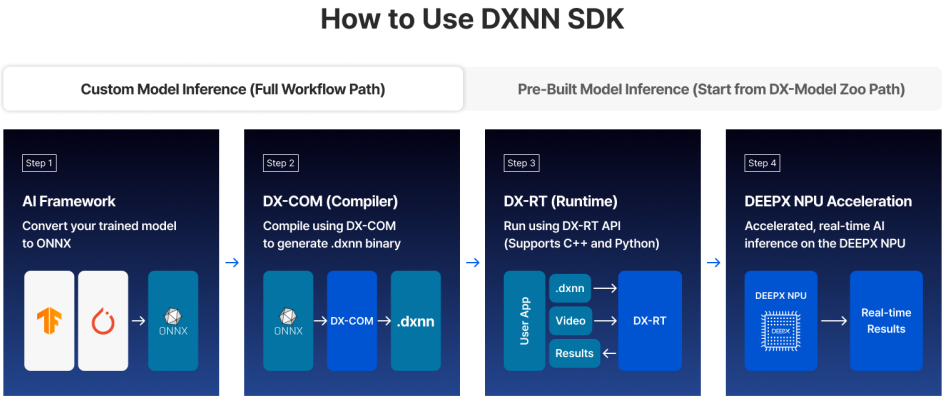


Figure. Custom Model Inference.

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 11–21 (17 recommended)** 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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

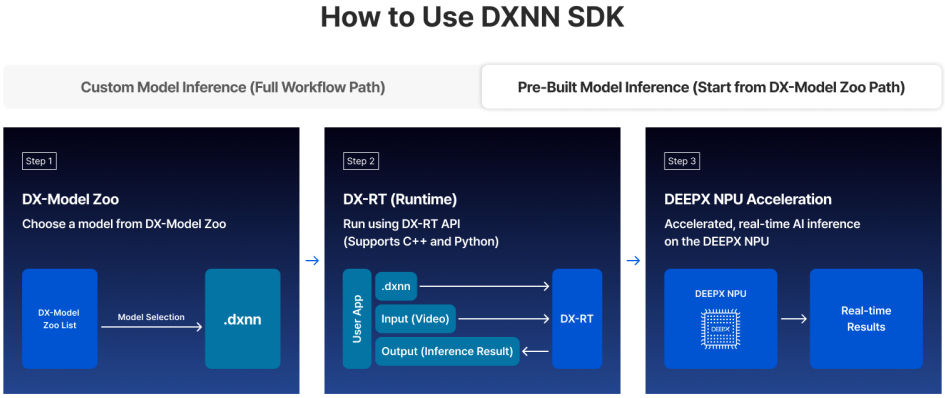


Figure. Pre-Built Model Inference.

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**.

[Path C] Model Evaluation Path (DX-ModelZoo)
The path for measuring accuracy and performance of pre-configured models with a single CLI (`dxmz`).

- **Step 1 (Select)**: Choose a model from **DX-ModelZoo** (`dxmz list` , `dxmz info`). Each model is defined by one YAML file (preprocessing, postprocessing, dataset, profiles).
- **Step 2 (ONNX Baseline)**: Evaluate the ONNX model on CPU/GPU (`--profile onnx`). Missing models are downloaded automatically from `DXMZ_MODEL_URL` .
- **Step 3 (DX-COM)**: Compile the ONNX model to `.dxnn` with a DXNN profile (`c-lite` , `q-lite` , `q-pro`).
- **Step 4 (DX-RT)**: Evaluate the `.dxnn` model on the NPU with the same dataset and compare the metrics with the ONNX baseline.

5.4 Local Installation

ⓒ Copyright 2026 DEEPX All Rights Reserved.

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.

5.4.1 DX-Compiler Installation (DX-COM)

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

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

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

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

© Copyright 2026 DEEPX All Rights Reserved.

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
```

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. Model Visualization

DX-TRON has been removed as of DX-Compiler v2.5.0. To inspect the compiled model structure and CPU/NPU workload distribution, generate the interactive HTML **Compilation Summary Report** with DX-COM:

© Copyright 2026 DEEPX All Rights Reserved.

```
dxcom -m model.onnx -c config.json -o output/ --export_html
```

Then open `output/<model_name>_summary.html` in your browser.

5.4.2 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 dxcli
dxcli -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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

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

5.4.3 DX-ModelZoo Installation

DX-ModelZoo is a YAML-driven framework for model evaluation and NPU compilation, operated through the `dxmz` CLI. It is located in the `./dx-modelzoo` submodule.

A. Pre-Installation Requirements

- **DX-Runtime**: DX-ModelZoo is installed into the runtime environment (`dx-runtime/venv-dx-runtime`), which provides `dx_engine` for NPU evaluation. Install DX-Runtime first (`./getting-started/runtime-0_install_dx-runtime.sh`).
- **DX-Compiler** (for `dxmz compile` only): The `dx_com` wheel must be installed in the same environment.
- **Python**: 3.8 or higher

B. Installation Method

The provided scripts install DX-ModelZoo and prepare its workspace.

```
# 1. Install dx-modelzoo into dx-runtime/venv-dx-runtime (CPU ONNX Runtime by default)
./getting-started/modelzoo-0_install_dx-modelzoo.sh

# (Optional) GPU ONNX Runtime
./getting-started/modelzoo-0_install_dx-modelzoo.sh --extra gpu

# 2. Create getting-started/modelzoo (-> workspace/modelzoo) and dx-modelzoo/.env
./getting-started/modelzoo-1_prepare_assets.sh
```

GPU Inference (ONNX Runtime GPU)

`--extra gpu` installs `.[gpu]` (`onnxruntime-gpu` and CUDA libraries) and removes `onnxruntime` , because the two packages cannot coexist. Do not combine `gpu` with `all` , which includes `cpu` . Requirements: NVIDIA GPU (Pascal/Turing/Ampere), NVIDIA Driver ≥ 535.230.02, CUDA 12.2, cuDNN 9.1. After installation, add the NVIDIA library paths printed by the script to `LD_LIBRARY_PATH` . These requirements apply to the local (pip) install; Docker GPU images have their own driver requirement (see GPU Acceleration Options below).

ⓒ Copyright 2026 DEEPX All Rights Reserved.

`modelzoo-1_prepare_assets.sh` copies `dx-modelzoo/.env.example` to `dx-modelzoo/.env` only when `.env` does not exist. Edit it for your environment:

Variable	Description
`DXMZ_MODEL_URL`	Base URL for automatic model download. Required for auto-download (no built-in default); `.env.example` sets `https://sdk.deepx.ai/modelzoo`
`MODEL_ROOT`	Model file root directory (overridden by `--model-root`)
`DATA_ROOT`	Dataset root directory (overridden by `--data-root`)

C. Verify and Usage

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

# 2. Verify the CLI and the NPU Python API
dxmz --help
python3 -c "import dx_engine; print('dx_engine OK')"

# 3. Inspect a model
dxmz info resnet50_224x224
```

- **Next Step**: Run the first model with `./getting-started/modelzoo-2_run_first_npu_model.sh`. Refer to **Running Your First NPU Model**.

5.4.4 [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.

```
dxcli -s
```


© Copyright 2026 DEEPX All Rights Reserved.

Success Checklist

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

B. System Integrity Check

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

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

> **Tip**
>
> If any item returns a **FAIL** or **Not Found**, please revisit the **DX-Runtime Installation** section (Building and Installing Modules) to ensure all components were compiled correctly.

5.5 Docker Installation

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

5.5.1 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.

© Copyright 2026 DEEPX All Rights Reserved.

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

5.5.2 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 **FAQ Troubleshooting Guide**.

5.5.3 GPU Acceleration Options (Optional)

Docker images can optionally be built with GPU acceleration. The two options serve **different purposes and different containers**, and are mutually exclusive (Ubuntu only):

Option	Accelerates	Target containers	Typical use case
`--nvidia_gpu`	Compute (CUDA)	`dx-compiler`, `dx-modelzoo`	Faster ONNX raw accuracy evaluation (dx-modelzoo) and `q-pro` quantization calibration (dx-compiler)
`--intel_gpu_hw_acc`	Media (VA-API)		Offload video decode/ scale to the Intel iGPU

© Copyright 2026 DEEPX All Rights Reserved.

Option	Accelerates	Target containers	Typical use case
		`dx-runtime` — **dx_stream pipelines only**	so the CPU stays free for pre/post-processing in multi-channel GStreamer pipelines

A. NVIDIA GPU (CUDA) — dx-compiler / dx-modelzoo

Host requirements: NVIDIA driver and nvidia-container-toolkit (`sudo nvidia-ctk runtime configure --runtime=docker && sudo systemctl restart docker`).

The NVIDIA driver must support the image's CUDA version, which the container toolkit checks at container start:

- Default CUDA 12.8.1 image: driver **R570 or newer** (`nvidia-smi` shows CUDA Version ≥ 12.8).
- Datacenter/professional GPUs on the R535, R550, R560, or R565 branches can also run it through CUDA forward compatibility. Other GPUs (e.g. GeForce) on these branches fail with `unsatisfied condition: cuda>=12.8` .
- With `--cuda_version=<ver>` , the driver must support that CUDA version. Ubuntu 24.04 cuDNN base images start at CUDA 12.6.3 (no lower tag exists), which needs driver R560 or newer on GeForce and other non-datacenter GPUs.

```
# Build on a CUDA base image (default CUDA version: 12.8.1, override with --cuda_version=<ver>)
./docker_build.sh --target=dx-modelzoo --ubuntu_version=24.04 --nvidia_gpu
./docker_build.sh --target=dx-compiler --ubuntu_version=24.04 --nvidia_gpu

# Run / stop — GPU images are tagged cuda<ver>-ubuntu-<os> and coexist with CPU images
./docker_run.sh  --target=dx-modelzoo --ubuntu_version=24.04 --nvidia_gpu
./docker_down.sh --target=dx-modelzoo --ubuntu_version=24.04 --nvidia_gpu
```


© Copyright 2026 DEEPX All Rights Reserved.

The dx-modelzoo GPU image already includes an `onnxruntime-gpu` build that matches the image's CUDA version, so ONNX evaluation runs on the GPU without extra setup.

- CUDA 12 images: `onnxruntime-gpu<1.27`; CUDA 13 images: `onnxruntime-gpu>=1.27`. CUDA/cuDNN come from the `nvidia/cuda` base image.
- **Do not upgrade `onnxruntime-gpu` inside the container** (e.g. `pip install -U onnxruntime-gpu`): the newest release may target a different CUDA major version, and evaluation then falls back to the CPU. The `[gpu]` extra is not needed either; it only adds pip CUDA 12 libraries that duplicate the base image.
- To check: `python -c "import onnxruntime as ort; print(ort.__version__)"`, and watch `nvidia-smi` during evaluation.

For dx-compiler, `q-pro` calibration runs on the GPU automatically when one is available (`--use_gpu`, default `True`). Use `--use_gpu False` to force CPU, or set `CUDA_VISIBLE_DEVICES` to select a specific GPU.

B. Intel GPU (VA-API Media Acceleration) — dx-runtime (dx_stream)

Host requirements: Intel GPU with the `i915` / `xe` driver loaded (`/dev/dri/renderD*` present on the host).

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

No dx_stream code changes are required — GStreamer `decodebin` automatically selects the VA-API HW decoder (e.g., `vah264dec`), and adding `vapostproc` to a pipeline additionally offloads resize/color conversion to the iGPU video-enhance engine.

> **Intel GPU scope**
>
> `--intel_gpu_hw_acc` accelerates **media processing for dx_stream (GStreamer) pipelines only**. It does not accelerate dx-compiler or dx-modelzoo workloads (compute acceleration is CUDA-only via `--nvidia_gpu`), and dx_app uses software decoding by default.

5.5.4 Container Access and Task Guide

A. DX-Compiler Environment (Model Conversion)

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

ⓒ Copyright 2026 DEEPX All Rights Reserved.

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
```

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)

© Copyright 2026 DEEPX All Rights Reserved.

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 bash

# 2. Inside Container: Verify NPU hardware recognition
dxcli -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 (interactive menu covers both C++ and Python examples)
./run_demo.sh
```

> **Demo Selection**
>
> `run_demo.sh` is a text-based menu with three steps:
>
> 1. **AI task**
> 2. **Language and execution mode**: C++ sync/async, Python sync/async, or Python with C++ post-processing
> 3. **Input**: video or image
>
> Enter the number at each prompt, or press **Enter** to accept the default. Use `./run_demo.sh --help` to pre-select options without the menu.

© 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.

C. DX-ModelZoo Environment (Model Evaluation)

The **DX-ModelZoo** environment evaluates pre-configured models with ONNX Runtime and the DEEPX NPU, and compiles them to `.dxnn`. The image pre-installs DX-ModelZoo (`dxmz`) in `/venv-dxnn` .

> **Rebuild Required for Older Images**
>
> `dxmz` is installed during the image build. An image built before DX-ModelZoo was added to `Dockerfile.dx-modelzoo` does not contain it — verify with `docker run --rm --entrypoint /venv-dxnn/bin/dxmz dx-modelzoo:ubuntu-24.04 --help` and rebuild if it fails:
>
> ```
> ./docker_build.sh --target=dx-modelzoo --ubuntu_version=24.04
> ```

C-1. Accessing the Container

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

# 2. Inside Container: Verify the CLI
dxmz --help
```


© Copyright 2026 DEEPX All Rights Reserved.

NPU Access

The `dx-modelzoo` container does not start its own `dxrtd`. It uses the DX-RT service running in the `dx-runtime` container (or on the host). Start the `dx-runtime` container before NPU (`dxnn`) evaluation.

GPU Image (ONNX Runtime GPU)

The default image installs `.[cpu]`. To evaluate ONNX models on an NVIDIA GPU, build and run with `--nvidia_gpu`. It uses the `nvidia/cuda` cuDNN runtime base image (CUDA/cuDNN included) and installs an `onnxruntime-gpu` build matching the image's CUDA version (see GPU Acceleration Options above), so no `LD_LIBRARY_PATH` setup is needed. The host requires the NVIDIA driver (R570 or newer for the default CUDA 12.8.1 image) and NVIDIA Container Toolkit.

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

The GPU container is named `dx-modelzoo-12.8.1-cudnn-runtime-ubuntu24.04`.

```
docker exec -it dx-modelzoo-12.8.1-cudnn-runtime-ubuntu24.04 bash
```

C-2. Preparing the Workspace and `.env`

`modelzoo-1_prepare_assets.sh` links `getting-started/modelzoo` to `${DOCKER_VOLUME_PATH}/modelzoo` (mounted from the host `workspace/` directory) and creates `/deepx/dx-modelzoo/.env` from `.env.example` if it does not exist.

```
cd /deepx/getting-started
./modelzoo-1_prepare_assets.sh
```

Review `/deepx/dx-modelzoo/.env` and set `DXMZ_MODEL_URL` if your network uses a model mirror. Place evaluation datasets under `getting-started/modelzoo/datasets` (e.g., `ILSVRC2012/val`).

© Copyright 2026 DEEPX All Rights Reserved.

Mounting an Existing Dataset Directory

If the datasets already live on the host (e.g., `/mnt/datasets/ILSVRC2012`), mount them read-only instead of copying. Set `DX_DATASETS_PATH` when starting the container — it is mounted at `/deepx/datasets` (override with `DX_DATASETS_TARGET`).

```
# Host terminal
DX_DATASETS_PATH=/mnt/datasets ./docker_run.sh --target=dx-modelzoo --
ubuntu_version=24.04
```

Inside the container, point the script or the CLI at the mount:

```
# Script: override DATA_ROOT
DATA_ROOT=/deepx/datasets ./getting-started/modelzoo-2_run_first_npu_model.sh

# CLI: override per run
dxmz eval resnet50_224x224 --profile onnx --data-root /deepx/datasets --save
```

C-3. Running the First Model

```
./modelzoo-2_run_first_npu_model.sh
```

Compilation

`dxmz compile` requires `dx_com`. The script installs it automatically with `pip` (the version pinned in `dx-compiler/compiler.properties`, falling back to the PyPI release); pass `--dx-com-wheel /path/to/dx_com-*.whl` to use a local wheel instead.

NPU Evaluation Without Compiling

`dxmz eval --profile q-lite` downloads a prebuilt `.dxnn` from `DXMZ_MODEL_URL`, so you can evaluate on the NPU without `dx_com`:

```
./modelzoo-2_run_first_npu_model.sh --skip-onnx --skip-compile
```

For the detailed script flow, refer to **DX-ModelZoo: Model Evaluation Scripts Guide** in **Running Your First NPU Model**.

© Copyright 2026 DEEPX All Rights Reserved.

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

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

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

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

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

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

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

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

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

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


ⓒ Copyright 2026 DEEPX All Rights Reserved.

Manual Startup

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

5.5.6 [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 (`dxcli`)

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

```
dxcli -s
```

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

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

[Example of Normal Output]

```
DXRT v3.5.0 (build: x.xxxxxxx)
============================================================
 * Device 0: M1, Accelerator type
---------------------  Version  ---------------------
 * RT Driver version   : v2.7.0
 * PCIe Driver version : v2.7.0
------------------------------------------------------
 * FW version         : v2.7.6
------------------------------------------------------
... (Continued)
```

B. System Consistency Check

ⓒ Copyright 2026 DEEPX All Rights Reserved.

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.

ⓒ 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

ⓒ Copyright 2026 DEEPX All Rights Reserved.

DX-ModelZoo Path

To evaluate pre-configured models (ONNX baseline → NPU compilation → NPU evaluation) instead of the sample models, use the `modelzoo-*.sh` scripts. Refer to **DX-ModelZoo: Model Evaluation Scripts Guide** below.

Directory Structure

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

```
getting-started/
├── calibration_dataset          # dx-compiler/dx_com/calibration_dataset
├── dxnn                         # dx-compiler/dx_com/output
├── forked_dx_app_example        # Example execution target (forked)
│   ├── bin
│   │   ├── efficientnet_async
│   │   └── yolov5_async
│   │   └── yolov5face_async
│   └── sample
│       └── ILSVRC2012
├── modelzoo                     # workspace/modelzoo (models, datasets, result)
└── 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

© Copyright 2026 DEEPX All Rights Reserved.

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
```

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.

© Copyright 2026 DEEPX All Rights Reserved.

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

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

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

[Step 4] `compiler-4_model_compile.sh` (Execute Model Compilation)

The final stage where the prepared model and data are combined to generate the NPU-optimized binary.

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

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

- **Output**: 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
```


© Copyright 2026 DEEPX All Rights Reserved.

```
./getting-started/runtime-2_setup_assets.sh # Prepare resources
./getting-started/runtime-3_run_example_using_dxrt.sh # Run example
```

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.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

> ⚠ **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.

6.4 DX-ModelZoo: Model Evaluation Scripts Guide

This section covers evaluating a pre-configured **DX-ModelZoo** model end to end — ONNX baseline, NPU compilation, and NPU evaluation — using the `modelzoo-*.sh` scripts.

6.4.1 A. Execution Order

The scripts **must** be executed in the following order. DX-ModelZoo reuses the DX-Runtime environment, so complete the runtime installation first.

```
./getting-started/runtime-0_install_dx-runtime.sh      # Prerequisite: DX-Runtime environment
./getting-started/modelzoo-0_install_dx-modelzoo.sh      # Environment Setup
./getting-started/modelzoo-1_prepare_assets.sh         # Workspace & .env Configuration
./getting-started/modelzoo-2_run_first_npu_model.sh      # ONNX Eval -> Compile -> NPU Eval
```

6.4.2 B. Common Principles & Tips

- **Shared Runtime Environment**: DX-ModelZoo is installed into `dx-runtime/venv-dx-runtime` (`/venv-dxnn` in Docker), so `dxmz` uses the `dx_engine` installed by `runtime-0_install_dx-runtime.sh` for NPU evaluation.
- **Auto-Environment Detection**: Like `compiler-3_setup_output_path.sh`, the scripts detect **Host** or **Docker** and link `getting-started/modelzoo` to the matching workspace path.
- **Evaluate Before Compile**: `dxmz eval` downloads missing models from `DXMZ_MODEL_URL`, but `dxmz compile` does not. The runner evaluates the ONNX model first, so the ONNX file exists before compilation.
- **NPU Evaluation Without Compiling**: `dxmz eval --profile <c-lite|q-lite|q-pro>` also downloads a prebuilt `.dxnn` from `DXMZ_MODEL_URL`. Use `--skip-onnx --skip-compile` to run NPU evaluation alone — no `dx_com` and no ONNX baseline required.
- **`dx_com` Is Installed Automatically**: When compilation is requested and `dx_com` is missing, the runner installs it with `pip` (version pinned by `dx-compiler/compiler.properties`,

© Copyright 2026 DEEPX All Rights Reserved.

falling back to the PyPI release). Use `--dx-com-wheel <path>` only when installing from a local wheel.

- **Datasets Are Not Downloaded**: Evaluation and calibration read the dataset defined in the model YAML. Place it under `getting-started/modelzoo/datasets` before running. In Docker, an existing host dataset directory can be mounted read-only instead: `DX_DATASETS_PATH=/mnt/datasets ./docker_run.sh --target=dx-modelzoo --ubuntu_version=24.04` mounts it at `/deepx/datasets`. Then run the script with `DATA_ROOT=/deepx/datasets` so it reads the mounted dataset instead of `getting-started/modelzoo/datasets`.

> ⚠ **Prerequisite**
>
> NPU evaluation requires a working DX-RT stack (`dxcli -s`). Compilation requires `dx_com` in the same environment (installed automatically by the runner).

6.4.3 C. Detailed Script Guide

[Step 0] `modelzoo-0_install_dx-modelzoo.sh` (Package Installation)
Installs DX-ModelZoo (`dxmz` CLI) from the `dx-modelzoo` submodule.

- **Core Function**: Runs `pip install -e "dx-modelzoo[cpu]"` in the runtime environment and verifies `dxmz --help`.
- **Smart Check**: If `dxmz` already works, installation is skipped (unless you use the `--force` flag).
- **Options**: `--extra gpu` (switches to ONNX Runtime GPU; `onnxruntime` is removed because it conflicts with `onnxruntime-gpu`), `--venv <path>`, `--source <path>`.

[Step 1] `modelzoo-1_prepare_assets.sh` (Workspace Configuration)

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

- **Structure**: Creates `models/`, `datasets/`, and `result/` in the workspace.
- **`.env` Setup**: Copies `dx-modelzoo/.env.example` to `dx-modelzoo/.env` only when `.env` does not exist. Set `DXMZ_MODEL_URL` for your network.

© Copyright 2026 DEEPX All Rights Reserved.

[Step 2] `modelzoo-2_run_first_npu_model.sh` (Evaluate, Compile, and Run on NPU)
Runs the full DX-ModelZoo flow for one model (default: `resnet50_224x224`, profile `q-lite`).

Stage	Command	Output
[1] Model Info	`dxmz info`	Inputs, profiles, dataset path
[2] ONNX Evaluation	`dxmz eval --profile onnx --save`	`result/eval_<model>_onnx.json`
[3] NPU Compilation	`dxmz compile --profile q-lite`	`models/<model>/<model>_q-lite.dxnn`
[4] NPU Evaluation	`dxmz eval --profile q-lite --save`	`result/eval_<model>_q-lite.json`

- **Options**: `--model <name>`, `--profile <c-lite|q-lite|q-pro>`, `--venv <path>`, `--dx-com-wheel <path>` (local wheel instead of the pip package), `--skip-onnx`, `--skip-compile`.
- **Shortcut**: `--skip-onnx --skip-compile` runs stage [4] only, on a `.dxnn` downloaded from `DXMZ_MODEL_URL` — the fastest way to check the NPU path.
- **Result Report**: Compare the metrics (e.g., Top-1/Top-5) and **FPS** in the ONNX and DXNN result files to check the quantization impact.

> ⚠ **Essential Check**
>
> The default model `resnet50_224x224` reads the full ImageNet validation set (1,000 class directories, 50,000 JPEG images) from `getting-started/modelzoo/datasets/ILSVRC2012/val` for both evaluation and calibration. ONNX evaluation of all 50,000 images on CPU takes tens of minutes (about 35 minutes at ~27 FPS in our test); use `--skip-onnx` to skip the baseline when the ONNX file is already in place. If the dataset or the model download is missing, refer to **Q7–Q9** in **FAQ Troubleshooting Guide**.

ⓒ 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-ModelZoo
	DX-COM	DX-TRON*	DX-FW	NPU Driver	DX-RT	DX-STREAM	DX-APP	
2026-10-23	v2.5.0							
	v2.5.0		v2.5.0					v1.0.0
	v2.5.0	—	v2.7.6	v2.7.0	v3.5.0	v3.2.0	v3.3.0	
2026-09-03	v2.4.3							
	v2.4.2		v2.4.2					-
	v2.4.1	v2.0.1	v2.7.4	v2.6.0	v3.4.2	v3.1.2	v3.2.2	
2026-08-14	v2.4.2							
	v2.4.1		v2.4.2					-
	v2.4.0	v2.0.1	v2.7.4	v2.6.0	v3.4.2	v3.1.2	v3.2.2	
2026-08-03	v2.4.1							
	v2.4.1		v2.4.1					-
	v2.4.0	v2.0.1	v2.7.4	v2.6.0	v3.4.1	v3.1.1	v3.2.1	
2026-07-22	v2.4.0							
	v2.4.0		v2.4.0					-

© Copyright 2026 DEEPX All Rights Reserved.

Release Date	DX-AllSuite							
	DX-Compiler		DX-Runtime					DX-ModelZoo
	DX-COM	DX-TRON*	DX-FW	NPU Driver	DX-RT	DX-STREAM	DX-APP	
	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	
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					-

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Release Date	DX-AllSuite							
	DX-Compiler		DX-Runtime					DX-ModelZoo
	DX-COM	DX-TRON*	DX-FW	NPU Driver	DX-RT	DX-STREAM	DX-APP	
	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	

DX-TRON Removal

DX-TRON was removed as of DX-Compiler v2.5.0. Releases from that version onward show `—` in the DX-TRON column. Use the DX-COM Compilation Summary Report (`--export_html`) for model visualization.

* DX-TRON is no longer included in DX-AllSuite starting from v2.5.0.
`—` : Not included in that release. `-` : Not listed in this matrix for that release.

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

ⓒ Copyright 2026 DEEPX All Rights Reserved.

status utility.

```
# Display NPU status and driver information
dxcli -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__)"
```

Step 3. Model Zoo

Verify the installed DX-ModelZoo version. Run it in the environment where `dx-modelzoo` is installed.

```
python3 -c "from importlib.metadata import version; print(version('dx-modelzoo'))"
```


ⓒ 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
- Q7. DX-ModelZoo Environment Error (`dxmz` / `dx_engine` / `dx_com` Not Found)
- Q8. DX-ModelZoo Model Download Timeout
- Q9. DX-ModelZoo Dataset Not Found
- Q10. Docker Mount Error After Recreating a Host Path
- Q11. NPU Device Driver Version Is Too Old

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)'
```


© Copyright 2026 DEEPX All Rights Reserved.

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

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

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 **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 **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.

© Copyright 2026 DEEPX All Rights Reserved.

8.2.1 Diagnostic Steps

- **Q2.1 Warning**:
  `[WARNING] it is recommended to use an **X11 session (with .Xauthority support)** when working with the 'dx-all-suite' container.` 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 applications (e.g., X11-forwarded windows from Docker containers) are optimized for X11. Wayland sessions can cause `xauth` processes to be unstable.
- **Q2.2:** Ending a Wayland session may purge X authentication data or change directory paths, causing Docker mount points to become invalid.

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 the existing container before re-running.

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


ⓒ Copyright 2026 DEEPX All Rights Reserved.

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

For more details, refer to the **Docker Installation** section in **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.

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 (`dxcli`)
Update the firmware manually by pointing to the specific binary file.

```
dxcli -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

© Copyright 2026 DEEPX All Rights Reserved.

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 updated successfully.

```
dxcli -s
```

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

© Copyright 2026 DEEPX All Rights Reserved.

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

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

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

```
sudo reboot
```

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

```
dxcli -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 **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 with 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

© Copyright 2026 DEEPX All Rights Reserved.

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.

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 **Version Compatibility**.

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

© Copyright 2026 DEEPX All Rights Reserved.

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 ...
```

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.

8.7 Q7. DX-ModelZoo Environment Error (`dxmz` / `dx_engine` / `dx_com` Not Found)

This error occurs when DX-ModelZoo commands run in a Python environment that does not contain all required packages.

8.7.1 Diagnostic Steps

Check the terminal for one of the following messages.

```
bash: dxmz: command not found
ImportError: dx_engine is required for DxRuntimeSession. Please install the DeepX runtime package.
```


© Copyright 2026 DEEPX All Rights Reserved.

```
ImportError: dx_com is not installed. Install it to use compile: pip install dx_com
TypeError: compile() got an unexpected keyword argument 'calibration'
```

8.7.2 Cause

`dxmz`, `dx_engine` (DX-RT Python API, for NPU evaluation), and `dx_com` (DX-COM, for compilation) **must** be installed in the same Python environment. DX-ModelZoo is installed into the runtime environment (`dx-runtime/venv-dx-runtime`, or `/venv-dxnn` in Docker), and `dx_com` is not installed there by default. A `TypeError` on a `compile()` keyword argument means the installed `dx_com` is older than the API DX-ModelZoo uses — reinstall `dx_com` with the version pinned in `dx-compiler/compiler.properties` (`COM_VERSION`).

8.7.3 Solution: Install All Packages into the Runtime Environment

Step 1. Install DX-Runtime (provides `dx_engine`)

```
./getting-started/runtime-0_install_dx-runtime.sh
```

Step 2. Install DX-ModelZoo into the same environment

```
./getting-started/modelzoo-0_install_dx-modelzoo.sh
```

Step 3. Provide `dx_com` for compilation
The runner installs `dx_com` automatically with `pip` (version pinned in `dx-compiler/compiler.properties`, falling back to the PyPI release) while keeping DX-ModelZoo's `numpy>=1.20,<2.0` requirement.

```
./getting-started/modelzoo-2_run_first_npu_model.sh
```

To install from a local wheel instead, pass one whose Python tag matches the environment (e.g., `cp310` for Python 3.10).

```
./getting-started/modelzoo-2_run_first_npu_model.sh --dx-com-wheel /path/to/dx_com-*.whl
```

If you only need NPU evaluation, `dx_com` is not required at all — a prebuilt `.dxnn` is downloaded from `DXMZ_MODEL_URL`.

```
./getting-started/modelzoo-2_run_first_npu_model.sh --skip-onnx --skip-compile
```

Step 4. Verify

© Copyright 2026 DEEPX All Rights Reserved.

```
source ./dx-runtime/venv-dx-runtime/bin/activate
dxmz --help
python3 -c "import dx_engine, dx_com; print('OK')"
```

For more details, refer to the **DX-ModelZoo Installation** section in **Setting Up Environment**.

8.8 Q8. DX-ModelZoo Model Download Timeout

This issue occurs when `dxmz eval` tries to download a missing model and the connection never completes.

8.8.1 Diagnostic Steps

- **Step 1. Check Logs**: The download log shows no progress, and the command fails after several minutes (60-second timeout per server address).

```
WARNING | Model file not found locally: .../models/resnet50_224x224/resnet50_224x224.onnx
INFO    | Downloading model: https://sdk.deepx.ai/modelzoo/onnx/resnet50_224x224.onnx → ...
SessionCreationError: Failed to download model from https://sdk.deepx.ai/modelzoo/onnx/
resnet50_224x224.onnx: <urlopen error ...>
```

- **Step 2. Check Connectivity**: DNS resolves, but the TCP connection to port 443 times out.

```
getent ahosts sdk.deepx.ai
timeout 15 nc -4 -vz -w 10 sdk.deepx.ai 443
env | grep -i proxy
```

If the log instead shows the following, `DXMZ_MODEL_URL` is not set (for example, `dx-modelzoo/.env` is missing):

```
Model file not found: .../models/resnet50_224x224/resnet50_224x224.onnx
To enable auto-download, set DXMZ_MODEL_URL environment variable.
```

8.8.2 Cause

Outbound HTTPS (TCP 443) to `DXMZ_MODEL_URL` is blocked by a firewall, NAT, or a missing proxy setting. This is not a certificate problem: the connection fails **before** TLS negotiation, so `--cacert` or CA updates have no effect. `dxmz compile` never downloads models, so it fails with `ONNX file not found` when the download did not complete.

© Copyright 2026 DEEPX All Rights Reserved.

8.8.3 Solution: Allow Network Access or Place Models Manually

Method 1: Allow Outbound Access or Configure a Proxy (Recommended)
Allow outbound TCP 443 to `sdk.deepx.ai`, or set the proxy before running the scripts.

```
export HTTPS_PROXY=http://<proxy_host>:<port>
export HTTP_PROXY=http://<proxy_host>:<port>
```

If your network provides a model mirror, set `DXMZ_MODEL_URL` in `dx-modelzoo/.env`.

Method 2: Place Model Files Manually (Offline)
Download the files on a host with internet access and copy them to the expected paths.

```
getting-started/modelzoo/models/<model>/<model>.onnx
getting-started/modelzoo/models/<model>/<model>_<profile>.dxnn
```

8.9 Q9. DX-ModelZoo Dataset Not Found

This error occurs during evaluation or compilation (calibration) when the dataset defined in the model YAML is missing or incomplete.

8.9.1 Diagnostic Steps

Check the terminal for messages such as the following (`resnet50_224x224`).

```
DatasetValidationError:
  Dataset not found: ILSVRC2012
  Expected path: .../ILSVRC2012/val
DatasetValidationError: Dataset validation failed for ILSVRC2012: Expected 50000 images in .../
ILSVRC2012/val, but found 0.
```

8.9.2 Cause

DX-ModelZoo does not download datasets. Each model reads `dataset.eval_path` (for example, `${DATA_ROOT}/ILSVRC2012/val`) for both evaluation and calibration, and the dataset loader validates its structure.

8.9.3 Solution: Prepare the Dataset under `DATA_ROOT`

Step 1. Check the expected path
The `[1] Model info` step of `modelzoo-2_run_first_npu_model.sh` shows the `Dataset` path, and the

ⓒ Copyright 2026 DEEPX All Rights Reserved.

`PATH INFO` header shows `DATA_ROOT` (default: `getting-started/modelzoo/datasets`). When you run `dxmz` directly, `DATA_ROOT` comes from `--data-root` or `dx-modelzoo/.env`.

Step 2. Place the dataset

For `resnet50_224x224`, place the full ImageNet validation set (1,000 class directories, 50,000 JPEG images) under `getting-started/modelzoo/datasets/ILSVRC2012/val`. The runner always passes `--data-root getting-started/modelzoo/datasets`, so to reuse an existing dataset, link it instead of editing `.env`:

```
ln -s /path/to/ILSVRC2012 getting-started/modelzoo/datasets/ILSVRC2012
```

When you run `dxmz` directly, you can instead set `DATA_ROOT` in `dx-modelzoo/.env` or pass `--data-root`.

8.10 Q10. Docker Mount Error After Recreating a Host Path

This issue occurs when a host path that is bind-mounted into a running container is deleted and recreated.

8.10.1 Symptom

`docker exec` fails even though the container is running.

```
Error response from daemon: mount /home/user/dx-all-suite/dummy_xauthority:/var/lib/docker/
rootfs/.../dummy_xauthority, flags: 0x5000: not a directory
```

8.10.2 Cause

A bind mount is bound to the inode that existed when the container started. Deleting and recreating the host path (for example, removing `workspace/` or `dummy_xauthority`) leaves the container attached to the old inode, so the mount no longer matches. When the source is missing, the Docker daemon creates a **directory** in its place — which fails if the container expects a file (`dummy_xauthority` is a file created by `docker_run.sh`).

8.10.3 Solution: Restore the Path Type and Restart the Container

Step 1. Restore a file source that became a directory

```
rmdir dummy_xauthority && touch dummy_xauthority
```


© Copyright 2026 DEEPX All Rights Reserved.

Step 2. Restart the container so the mounts are re-created

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

Stale Workspace Mount

The same applies to `workspace/` : if it was recreated after the container started, files added on the host are not visible inside the container. Restart the container to rebind the mount.

8.11 Q11. NPU Device Driver Version Is Too Old

8.11.1 Symptom

NPU evaluation fails immediately after the model is loaded.

```
RuntimeError: Failed to create InferenceEngine for model '.../resnet50_224x224_q-lite.dxnn':
[dxrt-exception] Invalid operation exception {"The current device driver version is 2.6.0.
Please update your device driver to version 2.7.0 or higher.":lib/device_version.cpp:
217:CheckVersion}
```

8.11.2 Cause

DX-RT inside the container and the NPU device driver on the **host** are separate components. The driver is a host kernel module, so rebuilding a Docker image upgrades DX-RT but never the driver. When the rebuilt image contains a newer DX-RT than the installed driver supports, DX-RT refuses to create an inference engine.

Check the installed versions on the host:

```
dxcli --status | grep -A2 Version
```

8.11.3 Solution: Update the Host NPU Driver

Step 1. Check the driver version pinned by DX-AllSuite

```
git submodule status dx-runtime/dx_rt_npu_linux_driver
```

Step 2. Install the pinned driver on the host

© Copyright 2026 DEEPX All Rights Reserved.

```
./getting-started/runtime-0_install_dx-runtime.sh
```

Step 3. Verify the driver version and re-run the evaluation

```
dxcli --status | grep "RT Driver version"
```

Host and Container Must Be Upgraded Together

Always update the host driver whenever you rebuild the runtime or DX-ModelZoo image. The container reuses the host driver through `/dev`, so an old driver blocks every NPU workload regardless of the image contents.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9. DX AI Studio

9.1 Installation & Launch

> ⚠ **Beta release (v0.1.1)**
>
> DX-AI-Studio is a **beta** release. Features, APIs, and the UI may change before the general-availability release.

9.1.1 Prerequisites

- **Python 3.8+**. `./launcher.sh` installs the package automatically on first run — no manual install step. The base Studio package uses the Python standard library (`dependencies = []`); the optional requirements below apply only to their features.
- **Local compilation** needs the DX Compiler SDK; **real NPU inference** needs DX-RT, a compatible driver, a DEEPX NPU and prepared models / examples (from `dx-runtime` / `dx_app`). Without these the Studio still launches; tools report missing requirements.
- **DX Stream** additionally needs the system **GStreamer + PyGObject** (`python3-gi` , `gstreamer1.0-*`) — standard on desktop Linux — to run pipelines.
- **Live DX App runs** (camera, RTSP, Continuous, Run Demo video, and Graph Live) draw the example's window on a virtual display and stream it to the browser. They need the OS packages **`xvfb`** , **`libxcb1`** , and **Pillow** (typically `python3-pil` on Debian/Ubuntu). These display dependencies are not required for ordinary image or batch-video inference. A missing dependency is reported rather than replaced with a successful mock run.
- **Graph Composer**, a feature of DX App, needs a dx_app build containing **`multi_model_graph_sync`** / **`multi_model_graph_async`** and the models used by the graph. Build the graph runners with dx_app's `./build.sh` ; Studio detects the installed build's capabilities. **Edit on the frame...** needs Pillow, and reading a video's first frame or using **Layers for this frame** also needs **`ffmpeg`** . See Graph Composer.
- **DX Cloud** requires the optional **`boto3`** AWS extra for real AWS operations, plus host-side credentials and the relevant AWS subscriptions / permissions. Browsing the page and its guide does not require this extra. See DX Cloud.

© Copyright 2026 DEEPX All Rights Reserved.

How the studio finds these

`./launcher.sh` creates its virtualenv with **`--system-site-packages`**, so it inherits the platform-provided `dx_engine` (DEEPX runtime) and `gi` (GStreamer bindings). The base Studio stays stdlib-only; feature-specific packages are not base dependencies. If `dx_engine` is missing, **DX Monitor** shows *NPU telemetry unavailable* and live inference is unavailable; if `gi` is missing, **DX Stream** can build but not run pipelines.

Camera discovery and preview decoding use a separate media Python with compatible NumPy and OpenCV; the Studio web server does not import `cv2`. If no compatible interpreter is available, Setup diagnostics and camera selection report the missing media runtime rather than treating it as an empty camera list. These are feature-specific requirements, not base Studio dependencies.

Check your NPU

Once the DEEPX runtime is installed, `dxrt-cli --status` lists each NPU device with its driver and firmware versions. **DX Monitor** shows the same info live in the browser.

9.1.2 Launch

From the `dx-ai-studio` directory:

```
./launcher.sh
```

It starts the launcher hub, boots every module server, and opens your browser at the hub URL. On first load a brief splash appears while servers start — give it a moment if a tool isn't ready yet.

Common options

Option	Effect
`--port <PORT>` / `-p <PORT>`	Preferred hub port (default **8890**). If another service holds it, the next free port is used; the last port is remembered in `.launcher-port`. Open the URL the launcher prints.
`--no-browser`	Start the servers but do not open a browser.

© Copyright 2026 DEEPX All Rights Reserved.

Option	Effect
`--no-kill`	Keep a studio that is already running (by default a new launch replaces your previous one).
`--fast`	Skip the boot animation in the terminal.
`--debug[=PATH]`	Log user actions to `var/log/studio-debug.log` (or to `PATH`).

Run `./launcher.sh --help` for the full flag list. The hub proxies all modules, so you only ever open the hub port.

9.1.3 Typical Workflows

Once the hub is running, here are common workflows:

🎯 Workflow 1: Model to Inference

The complete path from model to running inference:

1. **Model Zoo** — Browse and download AI models (ONNX + .dxnn variants)
2. **Compiler** — Convert ONNX to optimized `.dxnn` for DEEPX NPU
3. **App** or **Stream** — Run inference on images, video, or camera
4. **Monitor** — Watch real-time NPU performance and utilization

🎯 Workflow 2: Hardware Selection

Find the right DEEPX hardware for your workload:

1. **Benchmark** — Review benchmark results across NPU platforms
2. **EdgeGuide** — Get hardware recommendations based on your requirements

🎯 Workflow 3: Quick Development

Build NPU applications from natural language:

1. **Agent Dev** — Describe what you want; the agent generates and runs it

See **The Hub** for details on all nine tools and navigation.

© Copyright 2026 DEEPX All Rights Reserved.

9.1.4 How to Use This Manual

📖 Getting Started:

1. **Installation & Launch** (this page) — Get the studio running
2. **The Hub** — Understand the interface and navigation

🔧 Workflow:

- **Model Zoo** — Browse and download AI models
- **Compiler** — Convert ONNX to `.dxnn` for DEEPX NPU
- **App** — Run image/video inference
- **Stream** — Build real-time GStreamer pipelines

📊 Monitoring:

- **Monitor** — Watch NPU performance in real time
- **Benchmark** — Review benchmark results across NPU platforms
- **EdgeGuide** — Get hardware recommendations

🚀 Development:

- **Agent Dev** — Generate NPU apps from natural language
- **DX Cloud** — Compile on AWS and deploy to edge devices

📚 Reference:

- **SDK Library & About** — In-app documentation and company info

📋 Appendix:

- **Change Log** — Version history and updates
- **Third Party License** — Open source licenses

9.1.5 Stopping

`Ctrl+C` in the terminal running `./launcher.sh`. Running `./launcher.sh` again replaces your previous studio; pass `--no-kill` to leave it running (the new one then takes the next free port).

9.1.6 Remote access & security

© Copyright 2026 DEEPX All Rights Reserved.

By default the hub listens on **all network interfaces**, so a studio running on a headless NPU board can be opened from another machine's browser. The launcher prints that address under **other computers:** in its URL banner (for example `http://192.168.0.42:8890`), and the **Connected browsers** button in the top bar shows it too. Remote browsers must be **paired** first — nobody on the network can use the studio without the code shown on the board. The module servers behind the hub (App, Stream, Compiler …) only listen on the board's `127.0.0.1`; everything goes through the hub.

Who	What they need
You, on the board itself (or through an SSH / VS Code tunnel)	Nothing — works as before.
A browser on another machine	The **6-digit code** printed in the board's `./launcher.sh` terminal — once per browser.
A script / API client	`DX_API_TOKEN` (header `Authorization: Bearer <token>` or `X-DX-Api-Token`).

Pair a browser (open LAN access)

1. On the board, run `./launcher.sh`. Next to the URL banner it prints **`Remote access code: 123456`**.
2. On your laptop, open the address printed under **other computers:** (or shown in **Connected browsers** on the board). You see **Connect this browser** — type the code.
3. That browser is remembered for 30 days (`DX_SESSION_DAYS`). The code works once; a new one is printed after each use, and after 5 wrong tries the studio pauses for a minute and prints a new code.

Connected browsers (top bar) lists every paired browser — browser and OS, IP, last active, paired date — with **Disconnect**. On the board you see and disconnect all of them; a paired browser sees only itself and can **Disconnect this browser**. Scripts can use `GET /api/auth/sessions`, `POST /api/auth/sessions/revoke` (board only) and `POST /api/auth/logout`.

> **Watching DX Stream from another computer**
>
> Choose **Playback → Remote (MJPEG)** in DX Stream. **Local (WebRTC)** is for the board itself or the same LAN segment and does not cross an SSH tunnel.

© Copyright 2026 DEEPX All Rights Reserved.

Worked example — laptop → board at `192.168.0.42`

- **On the board:** `./launcher.sh` → note `Remote access code: 482915`.
- **In your laptop browser:** open `http://192.168.0.42:8890`, enter `482915`.

Private access via SSH tunnel (no network exposure)

Keeps the studio invisible to the network — only someone who can SSH into the board can reach it, and no code is needed (the tunnel arrives on the board's `localhost`).

1. On the board, bind to localhost only:

```
DX_BIND_LOCAL=1 ./launcher.sh
```

2. From your laptop, forward the port over SSH:

```
ssh -L 8890:localhost:8890 <user>@<board-ip>
```

   (On Windows, PuTTY/MobaXterm can save this as a stored port-forward.)

3. Open `http://localhost:8890` in your laptop browser.

What the studio refuses

- Requests whose `Host` is a name other than `localhost`, this machine's hostname or an address listed in `DX_ALLOWED_HOSTS` (an IP address is always fine) — this blocks DNS rebinding.
- Cross-site requests: no `Access-Control-Allow-Origin: *`, and a state-changing request (POST/PUT/PATCH/DELETE) carrying another site's `Origin` / `Referer` is refused, even on the board itself.
- Compile paths outside the allowed folders — the suite folder, the studio's `var/`, your home folder, `/media` and `/mnt` (add more with `DX_COMPILER_ALLOWED_ROOTS`).
- For HTTPS or access from outside the LAN, put the hub behind an approved reverse proxy that terminates TLS; requests that come through a proxy (`X-Forwarded-For`) are treated as remote and must be paired.

Multiple people

- **Different machines** — fully independent studios. On an open LAN, remember each is reachable by anyone via its board IP (use SSH tunnels to keep them private).

© Copyright 2026 DEEPX All Rights Reserved.

- **Same board, same Linux account** — relaunching `./launcher.sh` **stops the previous instance** (it clears stale studio processes owned by your user on start). Two people sharing one login will interrupt each other; pass `--no-kill` to leave a running instance alone, but they'll then contend for the port and shared files.
- **Same board, separate Linux accounts** — instances don't kill each other (the cleanup is per-user) and the port auto-bumps (8890 → 8891 …). Give each user their **own copy** of `dx-ai-studio` so they don't share `outputs/`, sessions, and downloads.

Environment variables

Variable	Effect
`DX_BIND_LOCAL=1`	Bind `127.0.0.1` only — no network exposure (use with an SSH tunnel).
`DX_BIND_HOST=<host>`	Bind the hub to an explicit interface/address.
`DX_API_TOKEN=<secret>`	Lets scripts call the studio from another machine (`Authorization: Bearer` / `X-DX-Api-Token`). Required to run a module server on its own beyond `127.0.0.1`.
`DX_PAIRING=off`	Disable browser pairing — remote access only with `DX_API_TOKEN`.
`DX_SESSION_DAYS=<n>`	How long a paired browser stays connected (default 30).
`DX_ALLOWED_HOSTS=<a,b>`	Extra host names the studio answers to (e.g. a DNS alias).
`DX_COMPILER_ALLOWED_ROOTS=<dir:dir>`	Extra folders the compiler may read from and write to.

9.1.7 Troubleshooting

- **DX Monitor says *NPU telemetry unavailable* / tools say the SDK is missing** — the NPU or SDK isn't detected. Confirm the driver is loaded with `lsmod | grep dx` (expect `dxrt_driver` and `dx_dma`) and the device with `dxrt-cli --status`; **DX Monitor**'s version panel shows what the studio sees.
- **A tool stays on the splash / "not ready"** — module servers may still be starting; wait a moment and reload. If it persists, check the terminal running `./launcher.sh` for errors.

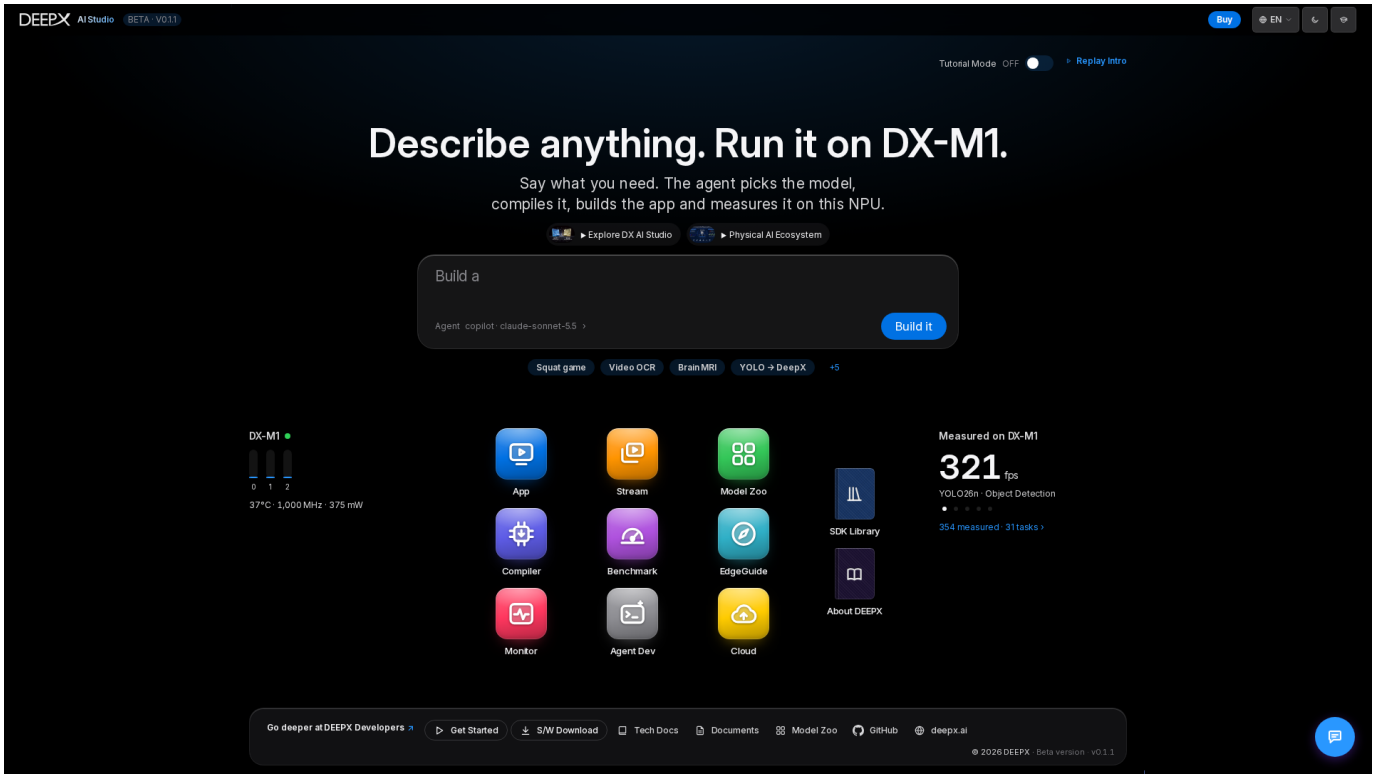
© Copyright 2026 DEEPX All Rights Reserved.

- **"Port already in use"** — 8890 auto-bumps to the next free port; pin one with `-p`, or pass `--no-kill` to leave an existing instance alone.
- **UI works but compile / inference fails** — inspect the error rather than assuming the UI proves readiness. Use **Setup** in DX Compiler for its SDK, or DX App / DX Stream for the runtime, driver, example binaries and models required by the action.
- **Stream refuses to launch after a runtime change** — open **Setup** → **DX-Runtime Dependencies** and **NPU Linux Driver**, inspect the failed check and follow its remediation. After an authorized installation or a requested reboot, run the checks again before retrying the pipeline. A browsable Dashboard is not proof that the runtime and driver passed validation.
- **Graph validation / Run once is unavailable** — check that Studio is using a dx_app checkout with the graph runners, build them there, and prepare the graph's models. A missing runner or model is not an automatic mock mode.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.2 The Hub

The **hub** is the home page that ties the studio together. Every tool opens from here and runs under the hub's single address; the hub proxies each module, so one URL is all you need (and all you can reach from another computer — module ports listen on the board only).



9.2.1 Layout

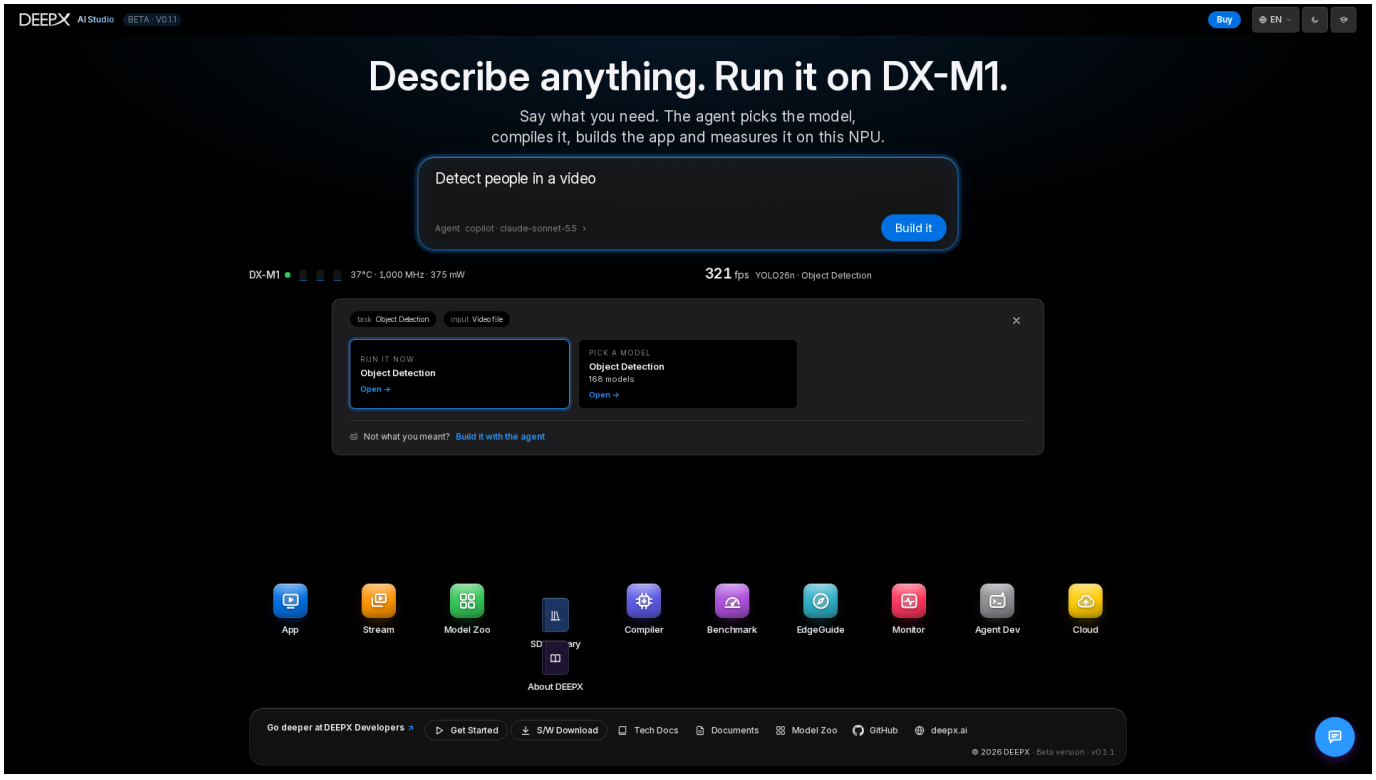
- **Prompt box** — *Describe anything. Run it on DX-M1.* Write what you want in plain words and click **Build it**. The hub first shows recommended tool routes or an agent plan — it does **not** immediately compile, run inference, or start an agent. Select a route card to open its tool, or choose the agent execution button to start the proposed work on the home page, where you follow activity and answer questions. Example chips fill in a request for you to review; the **Agent** row picks the coding agent, model, effort and mode. The agent and model are saved as the Studio agent and shared with Agent Dev and the chat.
- **Tool cards** — the nine tools plus **SDK Library** and **About DEEPX**. Cards show live facts where a module can count them (for example *26 demos · 12 groups* for DX App, the catalogue size for Model Zoo). Click a card to open the tool; the hub chrome stays in place.
- **DX-M1 widget** (left) — the NPU in this computer: cores, temperature, clock and power. Click it to open DX Monitor.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

- **Measured on DX-M1** (right) — a model's FPS measured on this device, and how many models and tasks were measured. Click it to open DX Benchmark.
- **Go deeper at DEEPX Developers** — the bar at the bottom links to Get Started, S/W Download, Tech Docs, Documents, Model Zoo, GitHub and deepx.ai.
- **Explore DX AI Studio** and **Physical AI ecosystem** — the two chips under the heading open the platform overview (every module at a glance) and the DEEPX ecosystem page.

If a module is not up yet, opening it shows a loading state; the hub keeps retrying for about 30 seconds, then shows **Module unavailable** with **Retry**.

For example, enter **"compile yolo26n to DXNN"**, press **Build it**, then select the Compiler route card. Review the model and config in DX Compiler before compiling. If the suggested route is not what you intended, edit the request or choose the agent option; an agent needs an installed, authenticated coding CLI.



9.2.2 Top bar

ⓒ Copyright 2026 DEEPX All Rights Reserved.

From left to right on the right side:

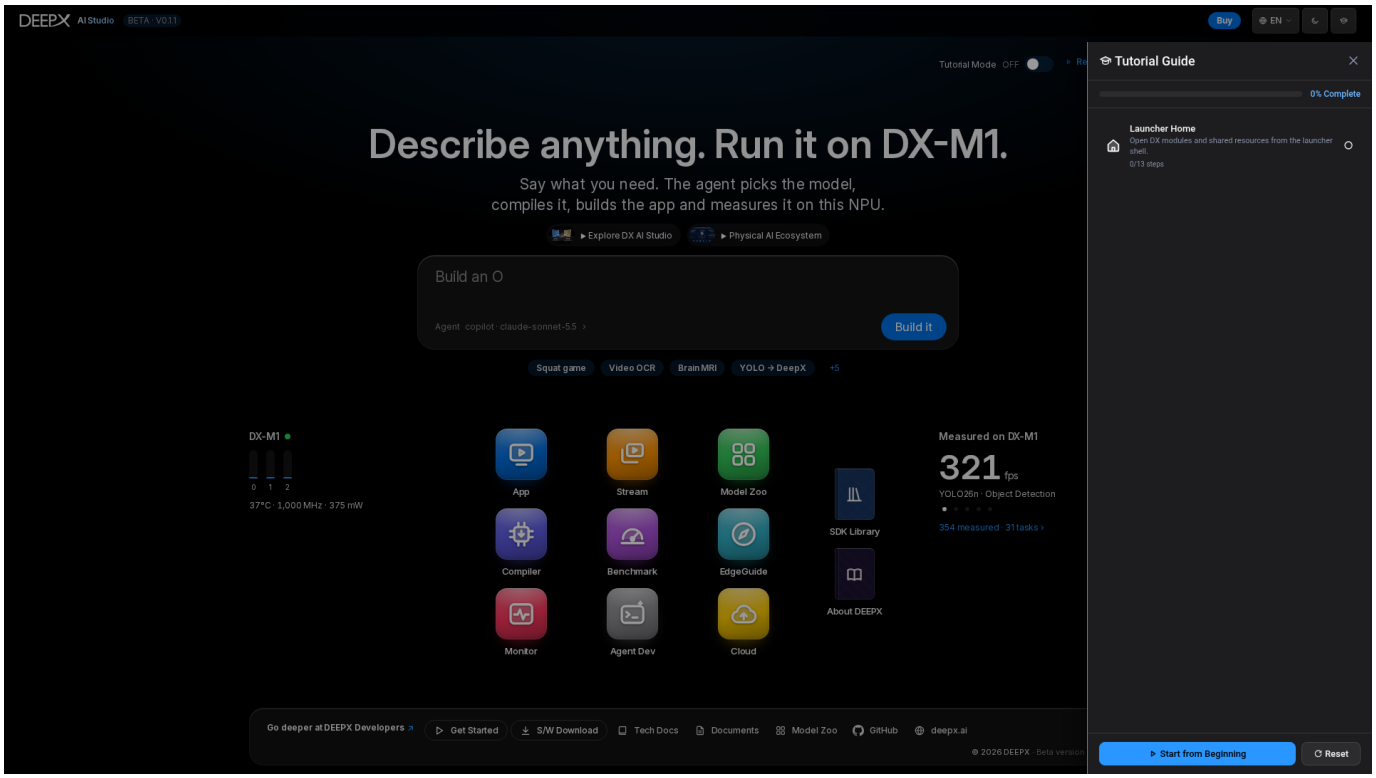
- **Buy** — the DEEPX store.
- **Language** — the six languages (see below).
- **Theme** — cycles dark → light → system.
- **Connected browsers** — appears when the studio is reachable from the network. It lists the browsers paired from other computers (browser, IP, last active, paired date) with **Disconnect**; on the board it also shows the address other computers open. See Remote access & security.
- **Tutorial** — opens the tutorial contents for the current view.

The global **DX AI Studio Help** assistant is the round chat button at the bottom right of every view. Its top line says which AI answers. With nothing set, it answers through the coding agent you are signed in to (claude, copilot, cursor …), the same one Home and Agent Dev use. **Change** asks one question — a coding agent on this PC, your local LLM server, or a cloud API key (OpenAI, Anthropic, Google, GitHub Models or a custom endpoint) — and tests the connection before saving; **Back to automatic** returns to the signed-in agent. See SDK Library & About. Its header also links to the web **DEEPX Agent**.

9.2.3 Guided tutorials

Tutorial Mode (the switch at the top right of the home, next to **Replay Intro**) is on by default. With it on, the first visit to the home runs a short walkthrough, and opening a tool opens that tool's tutorial contents — pick a section or **Start from Beginning**. Steps have **Prev / Skip / Next** (arrow keys work too); **Esc** ends the tour. The **Tutorial** button in the top bar opens the contents at any time.

ⓒ Copyright 2026 DEEPX All Rights Reserved.



9.2.4 Navigating

- Click a card or an icon in the **left module rail** to enter a tool. Use **Hub** at the bottom of the rail to return home; the **top page tabs** select views within the current tool (for example Models or Graph Composer inside DX App).
- The current tool and view are reflected in the **URL**, so links are shareable and reload-safe (for example, a DX EdgeGuide recommendation or an SDK Library document).
- **Keyboard shortcuts** — `Alt` + `1` … `9` open a tool directly. `Esc` closes the topmost thing first — the chat, the tutorial, a search, a dialog — and only then leaves the tool.

9.2.5 Language

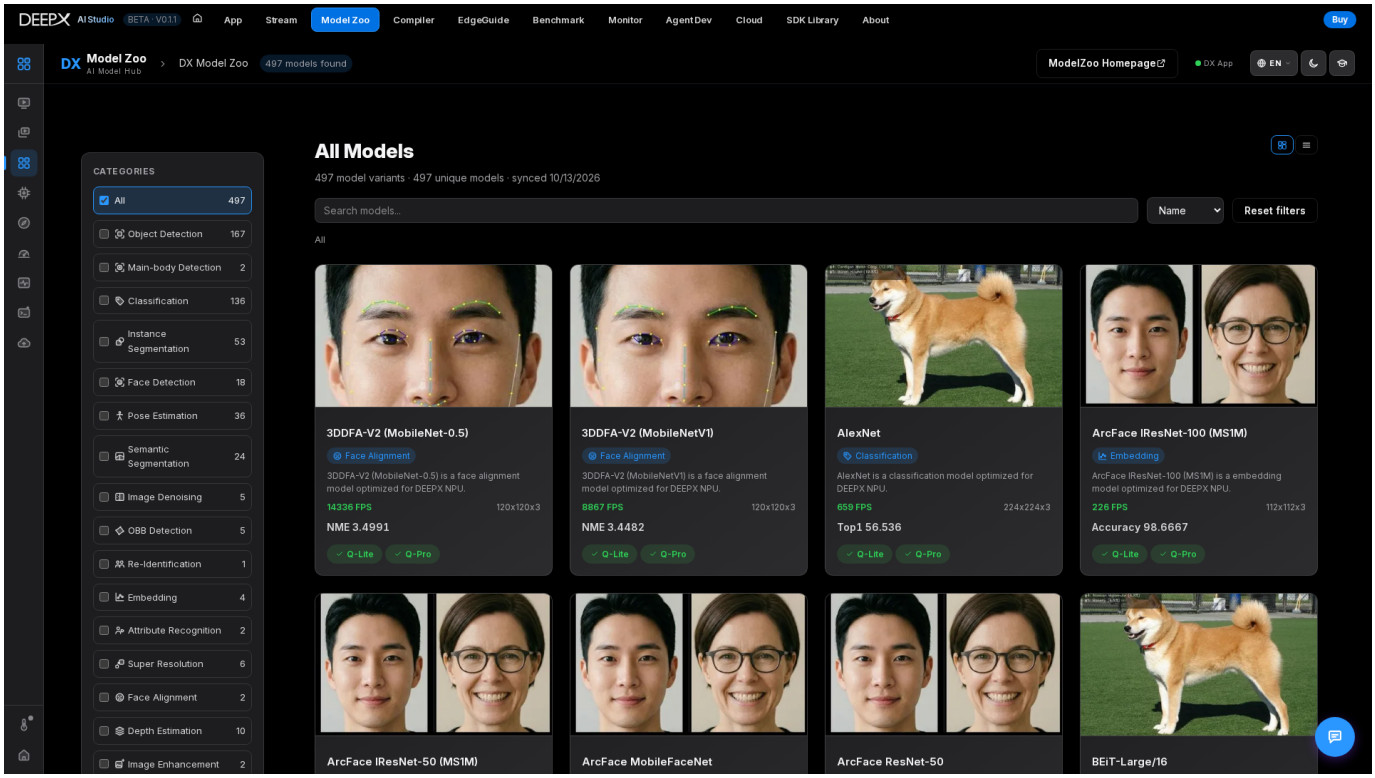
The entire studio is available in **6 languages** — English, 한국어, 日本語, 简体中文, 繁體中文, Español. A first visit opens in your browser's language (English when it is none of these). Switch from the top bar at any time; every open tool follows, and the choice persists and wins over the browser.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.3 DX Model Zoo

Browse the DEEPX model catalog — search and filter by AI task, open a model to see its details (accuracy, input, license / source), and use it in the other studio tools.

The October 2026 reference is **497 canonical model variants across 29 tasks**, aligned with dx_app dev `2e20108a5`. Variants include different sizes, inputs and training variants; this does **not** mean 497 distinct architectures. The Studio internal snapshot dated **2026-10-13** grew from 348 to 497 entries (+149). The explorer's variants / unique-model counts and sync date describe the loaded catalog; a different source or revision can differ. The Hub and Zoo UI group these entries under **31 category labels**, while the Graph palette uses **29 canonical tasks**; these are different groupings of the same 497 variants.



9.3.1 Using it

Step 1. **Browse or search** the catalog; filter by **category** (image classification, object detection, segmentation, pose, …), toggle **card / list** view, and sort. Cards show the available **Q-Lite / Q-Pro** artifacts; **Q-Master** appears only where provided.

Step 2. **Open a model** to see its detail view: task, input shape, accuracy metrics, legal info (license / source / copyright), and example images. The desktop detail view groups Key Facts, Example, artifacts / actions and metrics in a 2×2 layout. A before/after slider is

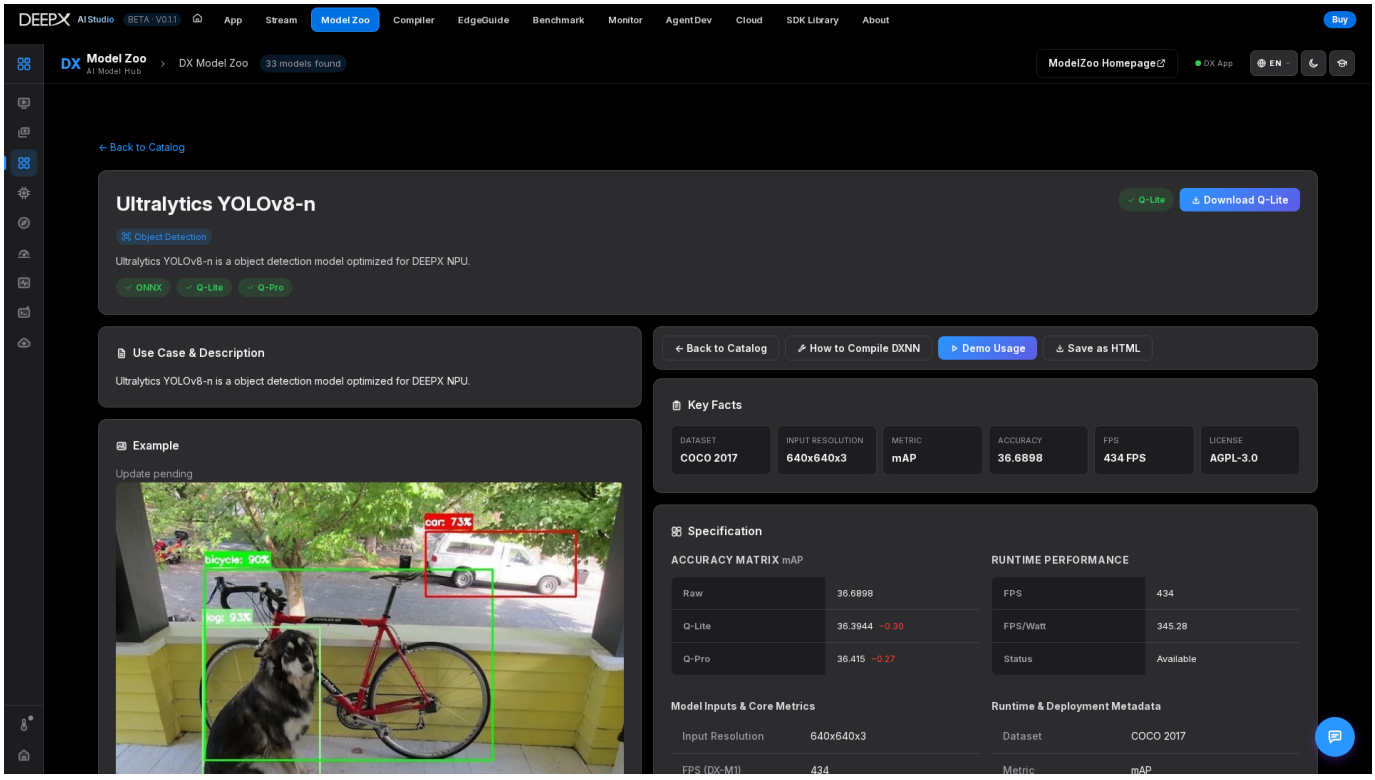
ⓒ Copyright 2026 DEEPX All Rights Reserved.

available for suitable examples, including super-resolution: open a Real-ESRGAN variant and compare its existing input / result images.

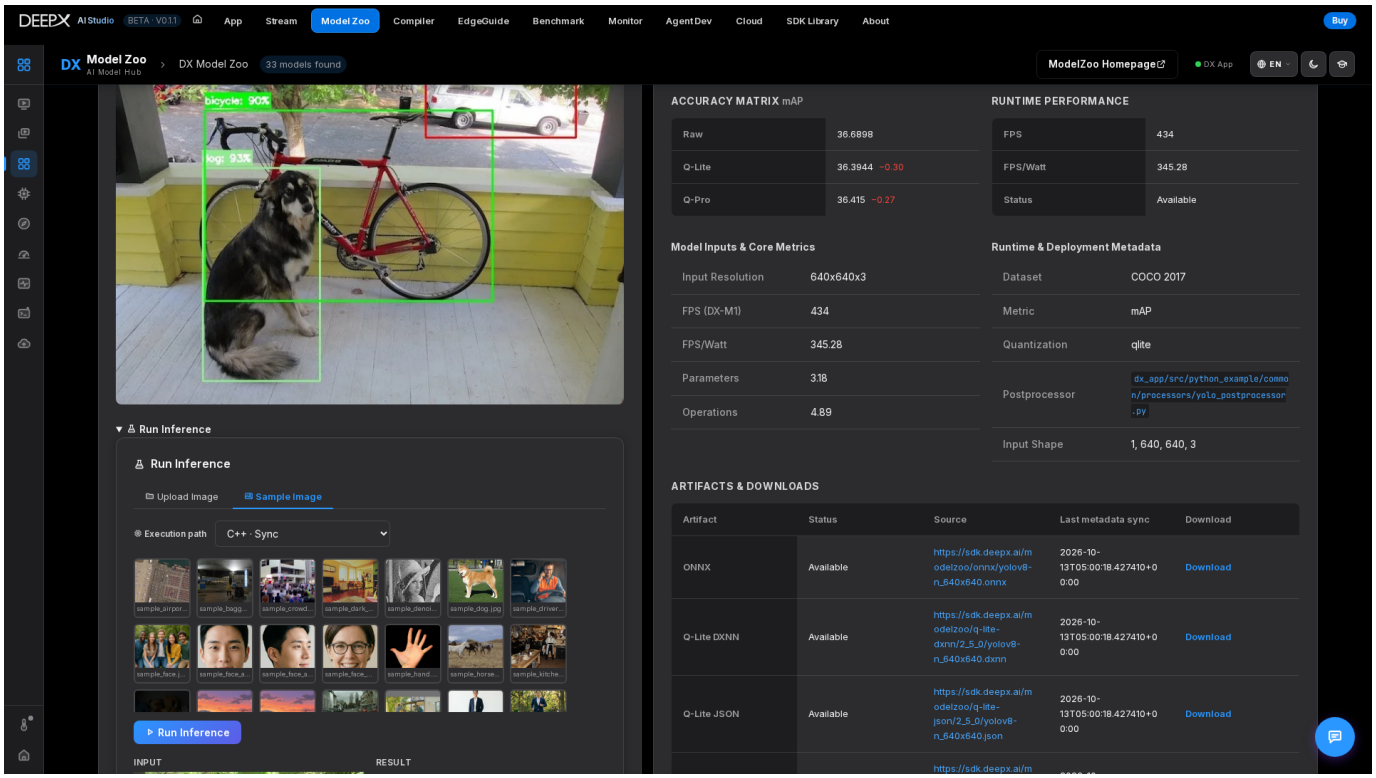
Step 3. **Download** an available artifact — a Q-Lite / Q-Pro `.dxnn` (with progress / cancel), ONNX or JSON. Not every model provides every artifact or quantization tier. In **DX App** → **ModelZoo**, an **Unverified** badge means the local file has no current Studio completion record, not that it is absent. Add it to the cart and run the download to verify it; matching content is kept. This checks transfer completeness, not whether the model has successfully run on this NPU.

Step 4. **Try it in-browser** — run inference right from the detail view (upload or sample image) against your local DX App: see the result image, predictions, FPS, and a per-stage latency table, and switch the execution path (C++/Python × Sync/Async). DX App must be reachable, the selected model downloaded, its example prepared / built, and the NPU runtime ready. Non-image tasks do not necessarily support this image demo.

Step 5. Or pull the model into **DX App** or **DX Compiler** (the detail view also links straight to "View Model Graph" and a compile guide).



ⓒ Copyright 2026 DEEPX All Rights Reserved.



9.3.2 Key features

- **497 canonical variants / 29 tasks** in the reference catalog, with per-category filtering, search, sort, and card/list views.
- **In-browser inference demo** and one-click artifact **downloads** from the detail view.
- **Rich detail view** — accuracy, license / source / copyright, copyable demo code (C++/Python/CLI), and save-as-HTML model card.

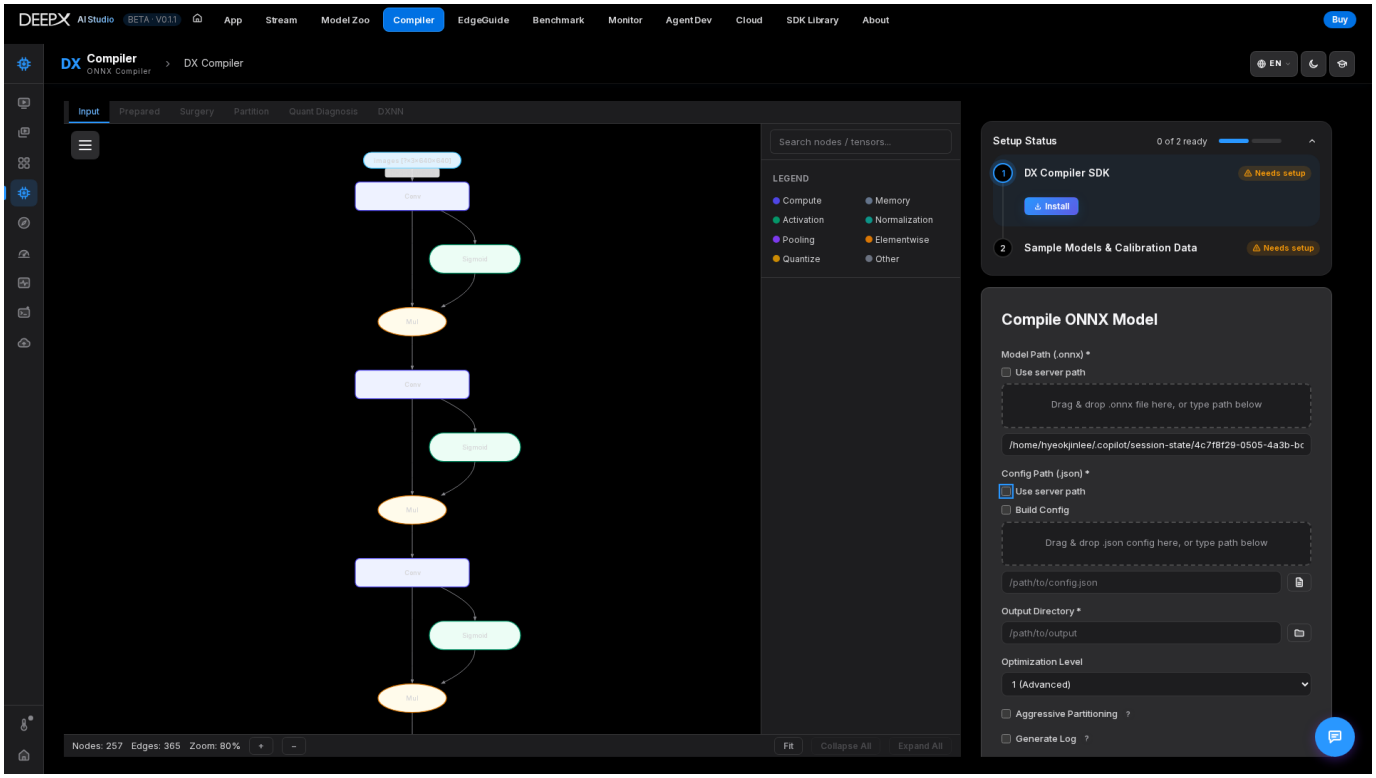
9.3.3 Reading results and recovering

- Card FPS and accuracy are **published catalog measurements**, not a guarantee for your board. Keep the metric name (for example mAP, top-1 or NME), dataset and quantization tier with the number; local inference timings describe a different execution.
- If inference is disabled or fails, check DX App **Setup**, download the selected tier and build its example before retrying. Missing example media alone does not mean the model needs to be re-inferred: reuse the supplied model images where available.
- The full catalog is not the Graph execution list. **492** variants are runnable in DX App's Graph Composer at the reference revision; the four anomaly models and the LiDAR model are excluded. See DX App.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.4 DX Compiler

Compile a trained **ONNX model into a `.dxnn` binary** that runs on the DEEPX NPU — with a guided config wizard, live graph visualization, quantization tuning and diagnosis, subgraph (range) compile, re-quantization, and an optional AI agent that does the whole thing for you.



9.4.1 What you can do

- One-click **ONNX → DXNN** compilation with a live progress bar and log.
- Build the compile **config** with a 4-step wizard — no hand-editing JSON.
- **Visualize** the model graph at each stage (Input → Prepared → Surgery → Partition → DXNN).
- Tune **quantization** (DXQ presets or automatic Q-Pro) and read a **Quantization Diagnosis** report that tells you whether the quantized model is healthy.
- Compile only **part of a model** by picking start/end nodes on the graph.
- **Re-quantize** a `.qxnn` artifact into a new `.dxnn` without re-running everything.
- Let an **AI agent** compile from just a model name / path / URL.

9.4.2 Using it

ⓒ Copyright 2026 DEEPX All Rights Reserved.

1. Setup (first run)

In the **Setup** panel, check the SDK / sample status. If the compiler SDK is missing, click **Install**; optionally **Download** sample models + calibration data. The Compile button stays disabled until the SDK is ready. Read the incomplete / locked step's reason, finish that step, and re-check its status before retrying; a loaded model graph alone does not mean the compiler is installed.

2. Choose the model

In **Compile ONNX Model**, drag & drop an `.onnx` file (or type its path). Toggle **Use server path** to point at a file already on the server.

3. Choose the config

Drag & drop a config `.json`, or tick **Build Config** to open the wizard:

Step 1. **Input Shapes** — **Auto Detect from Model**, or add input rows manually.

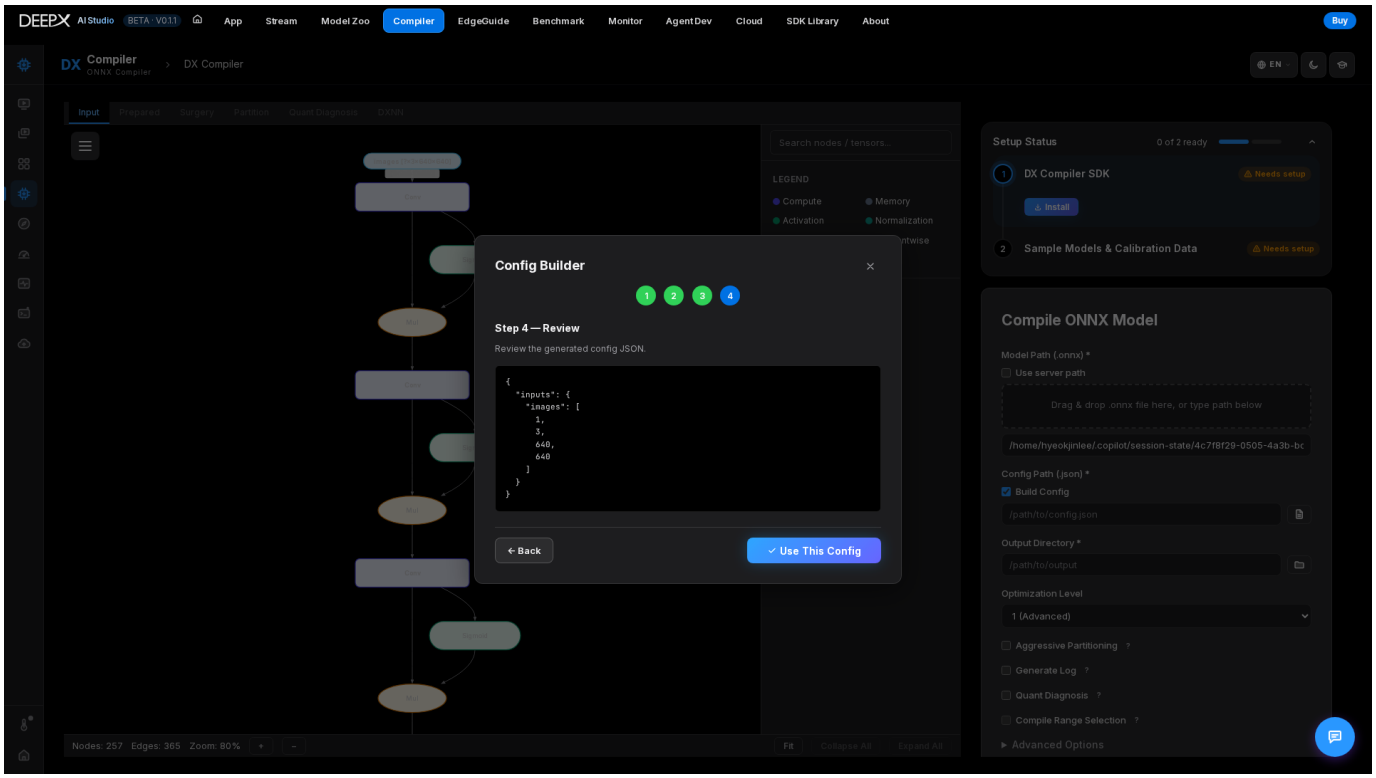
Step 2. **Data Loader** — **Default Loader** (real calibration data) or **Dummy Loader** (random tensors, quick check).

Step 3. **Dataset & Preprocessing** — dataset path, file extensions, a preprocessing pipeline (resize / normalize / …), calibration samples & method. *(skipped for Dummy)*

Step 4. **Review** — review the generated JSON, then **Use This Config**.

For example, use **Default Loader** with a dataset folder accessible on the Studio host, check the model's resize / normalization, and review the JSON before applying it. The wizard requires a dataset path for this loader. Use **Dummy Loader** only for a quick compilation check, not as evidence of calibrated accuracy.

ⓒ Copyright 2026 DEEPX All Rights Reserved.



4. Options

Set **Output Directory** and **Optimization Level**. Optional toggles: **Aggressive Partitioning** (more nodes on the NPU), **Generate Log**, **Quant Diagnosis**, **Compile Range Selection**. Under **Advanced Options** → **DXQ Enhancement Scheme**, enable quantization presets **DXQ-P0…P5** (each with its own parameters) or **Auto (Q-PRO)** for automatic tuning (Auto is mutually exclusive with the manual presets).

5. Compile

Click **Compile** — watch the **Current Phase** progress card and the live **Compiler Log**; graph tabs unlock as each phase finishes. On success, **Save Model Summary** downloads an HTML report of the compile.

9.4.3 Key features

- **Quant Diagnosis** — tick it before compiling to generate a Quantization Diagnosis HTML report (opens in the **Quant Diagnosis** tab). It shows the model **Status**, plus **Action Items** and **Evidence** (omitted when Status is **GOOD**). A diagnosis failure does not fail the compile.
- **QXNN Resume (re-quantization)** — the **Resume from QXNN** card re-quantizes an existing `.qxnn` into a new `.dxnn`: change the **Recalibration Method** (minmax / ema / iqr) and/or apply **Q-Pro / DXQ** presets, without re-running the full pipeline. **Dataset Path**

ⓒ Copyright 2026 DEEPX All Rights Reserved.

(optional) overrides the calibration dataset when needed. For example, open a completed diagnosis, choose **Resume from this QXNN** →, review the prefilled artifact, choose the new method / presets and optional dataset, then start the resume and inspect its log and new output. A diagnosis report is guidance, not an accuracy guarantee.

- **Compile Range Selection** — after the Prepare phase, use **Set Input Nodes** / **Set Output Nodes** on the graph, **Calculate Range**, then **Resume Compilation** to compile only the chosen subgraph.
- **Graph Viewer** — pan / zoom, **Fit**, **Collapse / Expand All**, `Ctrl+F` node search, click a node / tensor for details; a color-coded **Legend** by node category.
- **Agentic Auto Compile** — give a model name / path / URL, pick an agent (with LLM model and reasoning-effort; only the agents installed on this machine are listed), and it downloads, converts, configures, compiles, and verifies automatically — either fully hands-off or **interactive** (you can reply mid-run).
- **Sample-model picker** — the **Select Sample Model** button fills the Model Path from an already-downloaded sample.
- **AI Assistant** chat (floating button) explains errors and settings.

If compilation is refused or fails, read the Setup reason or the last failing phase in **Compiler Log**, correct the SDK / input / dataset / config, and retry. A diagnosis failure is separate from compilation success; do not treat a graph preview or an incomplete job as a produced `.dxnn`.

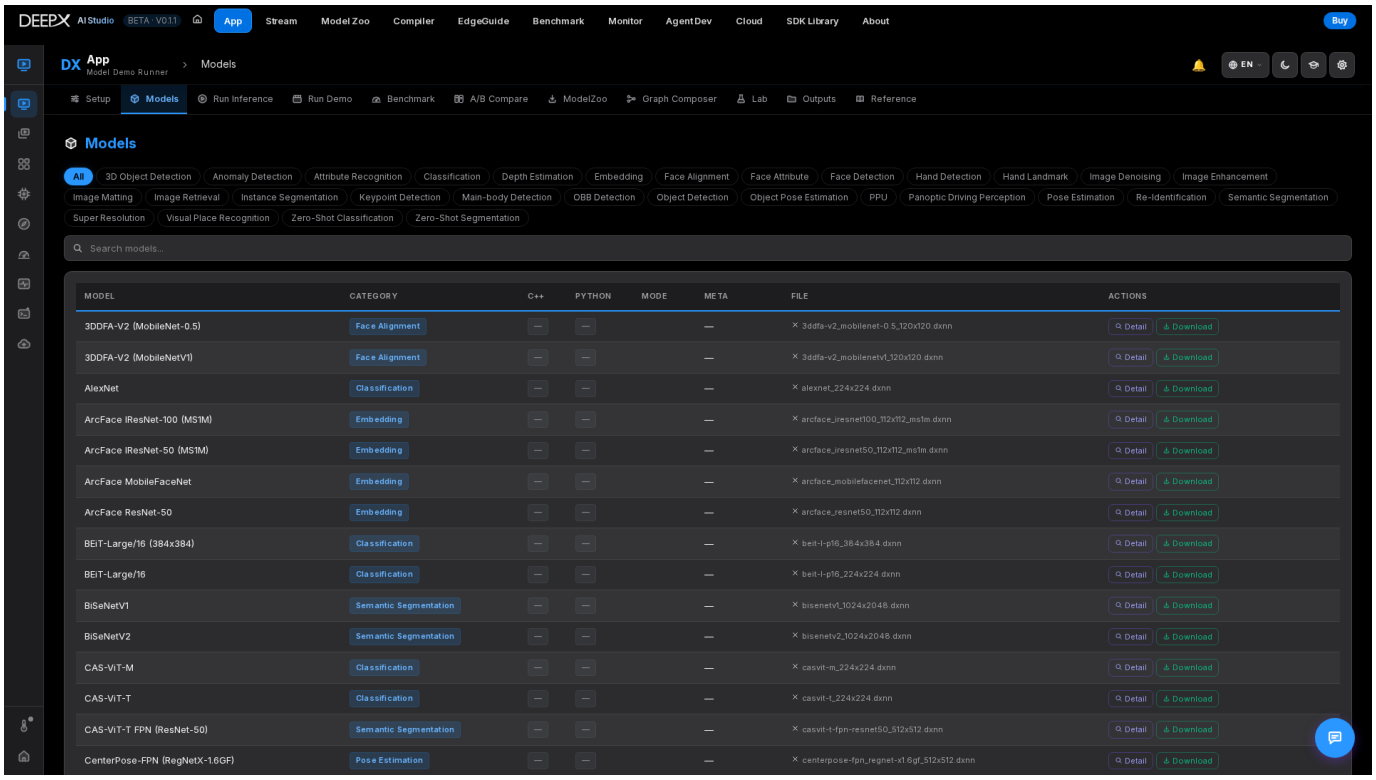
Related

Compiler outputs (`.dxnn`) run in **DX App** and **DX Stream**. Source models come from **DX Model Zoo**.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.5 DX App

Run AI inference on the DEEPX NPU from your browser — pick a model, run it on images, video, a camera or an RTSP stream, watch live results, and benchmark or compare models.



9.5.1 Using it

ⓒ Copyright 2026 DEEPX All Rights Reserved.

The module opens on a set of pages (top tabs):

ⓒ Copyright 2026 DEEPX All Rights Reserved.

- **Setup** — guided environment check; run it first so the NPU / runtime is ready.
- **Models** — the model list across 29 AI tasks in the October 2026 reference (detection, classification, segmentation, pose, depth, super-resolution, 3D object detection, PPU and more), drawn from the Model Zoo catalogue; open a model for details or its graph. The reference contains **497 canonical variants**, not 497 distinct architectures; counts depend on the loaded catalog. The Hub / Zoo's **31 UI category groups** differ from the Graph palette's **29 canonical tasks**; this does not change the variant count.
- **Run Inference** — pick a **category** → **model** → **input**, then Run. Two tabs:
  - **Single** — one image or video. Inputs adapt to the category: a sample image, your **own uploaded image**,
    or a video (some tasks are image-only; special inputs like 3D LiDAR `.bin` appear where they apply). Image runs show a **before/after compare slider**.
  - **Continuous** — video, camera or RTSP streamed live, with several slots side by side. For RTSP, paste the camera's full URL (`rtsp://user:pass@cam:554/…`); an `IP:Port` with a stream pick still works. A model that takes still images only is refused with an explanation.
- **Run Demo** — 26 ready-made demos in 12 groups on one stage: choose a demo, the input (image or video), C++ or Python, Sync or Async, and run. Video runs stream live with FPS.
- **Benchmark** / **A/B Compare** — measure a model's throughput and compare models side by side; Benchmark can run several models in a batch and **export a report**.
- **ModelZoo** — browse and download additional models into the app (public or air-gapped source, Q-Lite / Q-Pro variants, batch cart).
- **Graph Composer** — a DX App feature for connecting sources, models, CPU operations and logic on a canvas, then validating, running and exporting the pipeline. Follow the Graph Composer workflow below.
- **Lab** — advanced tools, each with a change preview before anything is written:
  - **Add Model** — copies an existing example of a category (**Start from**) under a new name, in C++, Python or both. Run the Python copy with your own model: `python3 <name>_sync.py --model <your>.dxnn` . Build the C++ copy with dx_app's `build.sh` .
  - **Create Task** — skeleton code for a new task type.
  - **Experiment** — checks a compiled `.dxnn` on the board: reads the file, runs one inference, then measures throughput (FPS).
  - **Generated Files** / **Safety Center** — what each change wrote. A change that only created new files can be rolled back after typing `rollback:<name>` ; one that overwrote files has to be restored by hand.
- **Outputs** — browse and manage saved inference results (grid / table, filters, preview).
- **Reference** — searchable in-app feature and parameter guides.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

First inference or demo

1. Open **Setup** and complete the environment checks. For a first demo, use **Demo Quick Start** → **Start Demo Setup**; review the installation progress and any failed step.
2. When setup is ready, choose **Try demo**, select a demo and a supported sample input, then run it. Alternatively, use **Run Inference** → **Single**, select a category, model and sample / uploaded image, and run.
3. Inspect the result and reported timings; saved results appear in **Outputs**. If a model, binary or runtime is missing, return to the named Setup step or download the model and retry. An error is not a completed inference.

Quick Start prepares the demo runners, their models and sample videos, not every per-model build target. The full Setup checks the declared model set and the runners needed by the demo and Graph workflows. Xvfb is an optional Live requirement, not a reason to repeat the base dependency installation.

A local asset without a current Studio completion record is **unverified**, not missing. Use **Verify** or rerun the corresponding setup step to check existing content and finish preparation; matching files are kept. Video folders without a completed Studio archive inventory remain unverified even if individual files have completion records. An installed file or a ready demo set is not proof of a complete, verified archive or a successful inference.

> **Live runs**
>
> Continuous, Run Demo video and Graph Live draw the example's window on a virtual display and stream it to the browser. Their optional display requirements are listed under Prerequisites.

From the Run page you can also **Export Model Package** — bundle a model's source, config, and file (C++ / Python / both) for reuse.

9.5.2 Graph Composer

Use **DX App** → **Graph Composer** to build a multi-model workflow without writing the runner yourself. It is part of DX App, not a separate Studio tool.

Prepare and validate

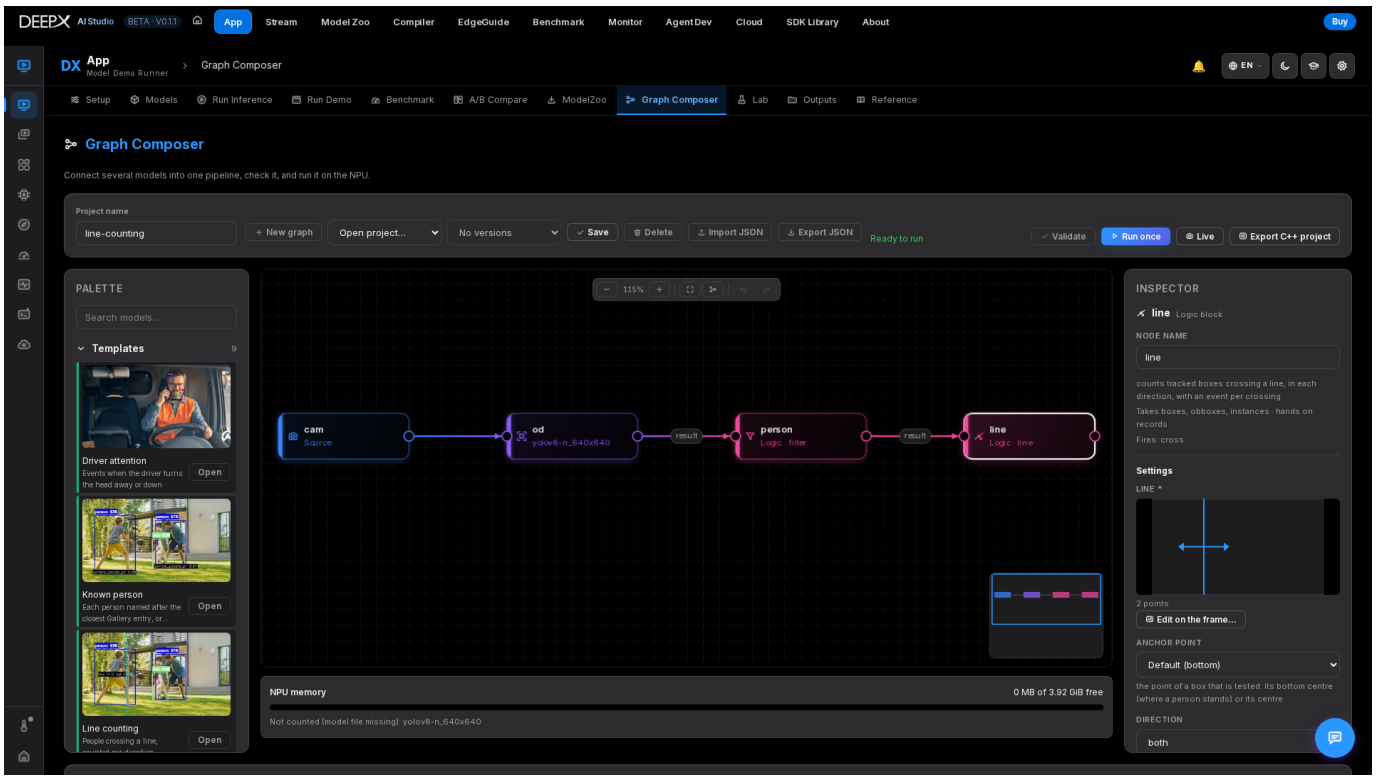
1. Open **Templates** and choose a scenario, such as restricted-zone intrusion or line counting. Other scenarios include PPE, dwell, zone counts, gallery matching, driver attention, multi-

© Copyright 2026 DEEPX All Rights Reserved.

camera counts and volume alerts. Engine examples are also available; names and descriptions follow the UI language.

2. Check the template's models in the palette. Installed models are marked; use **Download** for missing files. The connected dx_app must provide the graph runners and the required capabilities. At reference revision `2e20108a5`, **492 of 497 canonical variants** are graph-runnable: four anomaly-detection variants are not graph-ready, and the LiDAR model (`sfa3d_608x608`) needs an unsupported source.
3. Select each **Source** and set its input. Start with a local sample image or video for **Run once**; camera and RTSP sources are for **Live**. For several sources, supply an input for each. File paths refer to the Studio host, not the remote browser.
4. Select nodes / connections and use the **Inspector** to change properties. The graph is checked after edits; use **Validate** to inspect current errors pinned to their node or edge. Check the NPU memory bar as well. Fix missing models, incompatible connections or excess memory before running; engine sharing or fewer buffers can reduce memory where the runner supports them.

Keyboard editing includes Tab between nodes, arrows to move, Delete, and Ctrl+Z/Y.

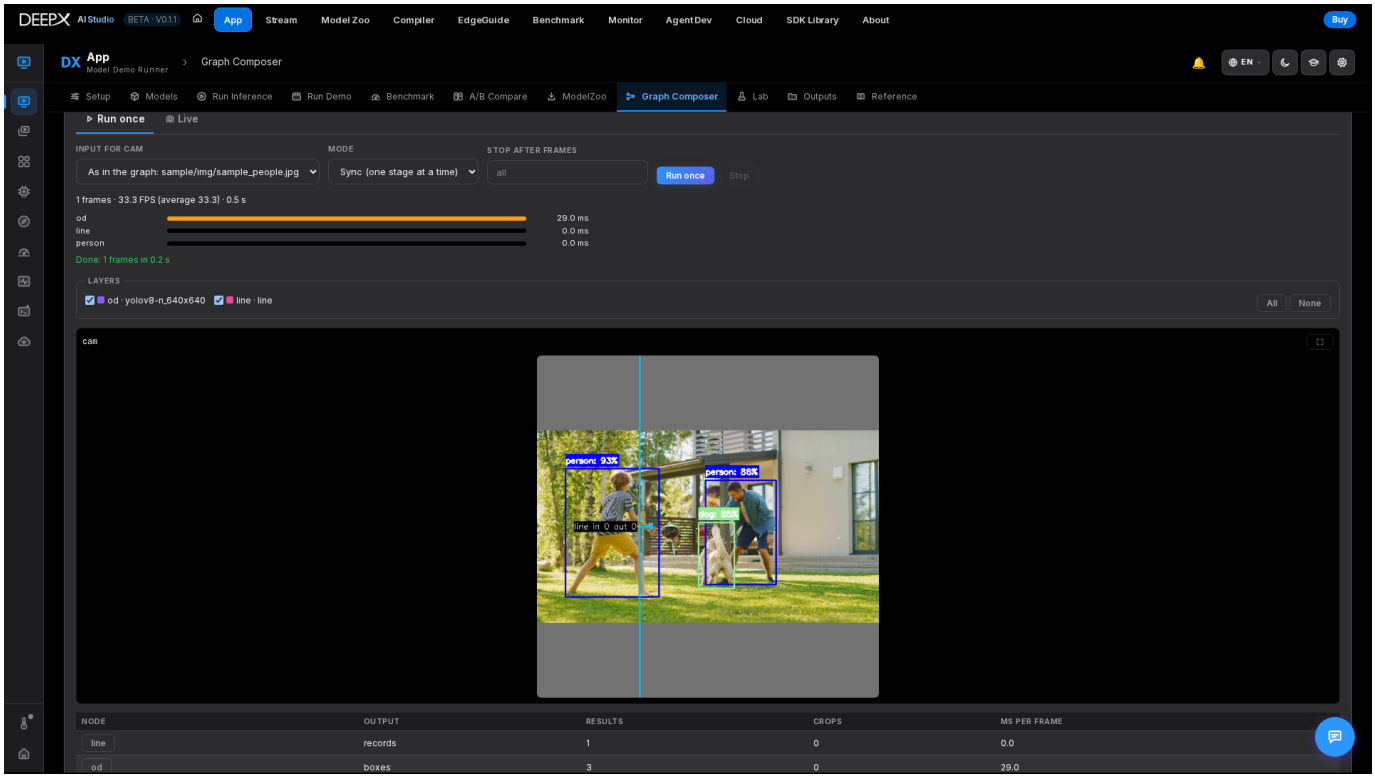


Run once and read the result

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Choose **Sync / Async** and **Run once**. Watch progress, FPS and per-node time, then inspect the rendered output and node results. Multiple sources appear as a grid. **Layers** toggle individual node drawings without changing the graph.

For a paused video, **Layers for this frame** executes the selected input frame again on the NPU. It does **not** restore the original tracker, zone, dwell, condition or gallery state: track IDs, event times and results can differ from the original video. It also needs a free NPU slot and the frame-reading dependencies.



Add logic, zones and events

Connect model results to **Filter**, **Zone**, **Line crossing**, **Count**, **Dwell**, **Condition** → **Event** or **Gallery match**, as offered by the installed runner. For example, open the intrusion template, select its Zone block and choose **Edit on the frame...**. Set a file input first; draw / adjust the polygon on its first frame, then run a video and inspect the emitted events. A camera or RTSP URL has no local first frame for this editor; use a representative file input to define the zone.

Line-crossing and dwell logic need detector tracking enabled. **Gallery match** needs its gallery prepared; a gallery build uses the NPU. **User logic** passes data through in Studio until you implement its exported C++ stub.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Filter the event list by node, event, track ID or time. Clicking an event seeks the result video to that moment; download **JSONL** for later analysis. No events can mean the rule was never met — check inputs, tracking and thresholds before assuming a failure.

Live, save and export

- **Live** runs a camera, RTSP stream or looping video until **Stop**. Only one Graph Live runs at a time, and leaving Graph Composer stops it. Graph Run once, Graph Live, the Run page's Continuous / Live slots and gallery builds exclude each other. Multiple ordinary Continuous slots may run together; stop those slots before Graph.
- **Save / Ctrl+S** stores a project on the Studio host under `outputs/dx_app/graph/projects/`. Only the **latest 50 versions per project** are kept; use the version selector to reopen a retained version.
- **JSON export / import** transfers the editable graph definition. It is not a model bundle or a standalone executable project.
- **Export C++ project** first shows an export plan, then produces a **zip** containing a standalone CMake project, the selected engine code, `main.cpp`, `graph.json`, `setup.sh`, `run.sh` and a README. Model files are excluded by default; **Include the model files** requires all of them to be installed. Otherwise the exported setup downloads them. User logic becomes pass-through stubs under `user_logic/` that you must implement. Follow the exported README for DX-RT, CMake, C++17 and OpenCV requirements; export success alone does not validate another host or guarantee identical stateful results. The server retains the **newest five exports** under `outputs/dx_app/graph/exports/`; download the zip to keep a copy.

If an action is blocked, read the validation / runtime reason: prepare the missing runner or model, correct the input / connection, reduce memory, or stop the NPU job named by the message, then retry. Missing hardware is **not** an automatic Graph mock mode. The **Tutorial → Graph Composer** walkthrough can show recorded results without starting the NPU or replacing unsaved work; those previews are not your own run.

You can browse Studio without an NPU, but real inference requires the hardware and runtime. Availability and errors are shown rather than making every page a mock demo.

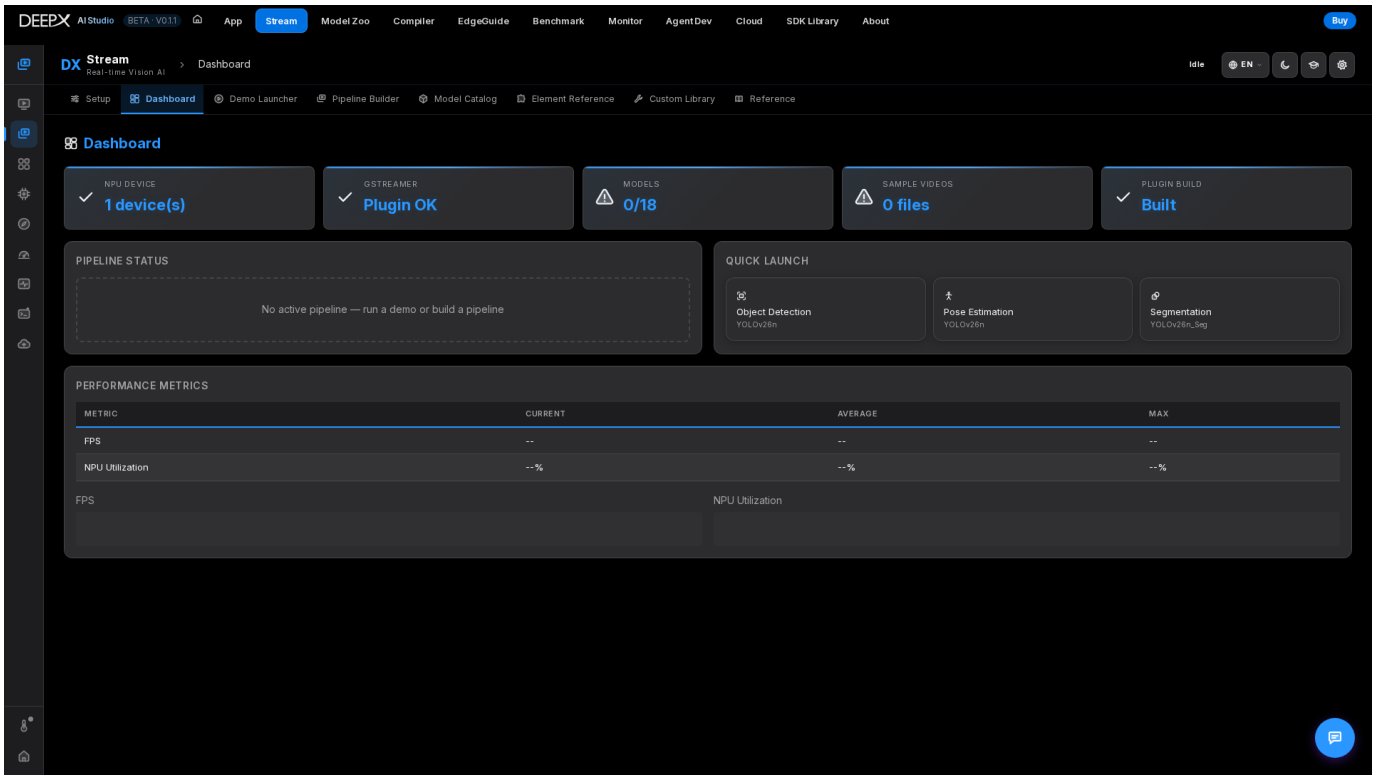
Related

Run the `.dxnn` files produced by **DX Compiler**; the same NPU telemetry is visualized live in **DX Monitor**.

© Copyright 2026 DEEPX All Rights Reserved.

9.6 DX Stream

Build and run real-time GStreamer vision-AI pipelines on the DEEPX NPU from your browser, with live playback — WebRTC on the board or LAN, MJPEG from another computer or a tunnel.



9.6.1 Pages

- **Setup** — one-click guided install of everything a pipeline needs, in six steps: Build Tools & Libraries, DX-Runtime Dependencies, NPU Linux Driver, GStreamer Plugin Build, Model & Video Download, and WebRTC Dependencies. Run one step or **Set up the rest**; an environment check and **Deep Diagnostics** show what's ready.
- **Dashboard** — module status, quick-launch demo cards, and live FPS / latency / NPU-util metrics with sparklines.
- **Demo Launcher** — the preset pipelines by category (object / face detection, pose, segmentation, depth, multi-stream, RTSP, …). The chosen demo opens on a stage: pick **Playback — Local (WebRTC) / Remote (MJPEG)**, an RTSP URL where it applies, and **Start**; FPS, resolution and model are shown beside the video, and **Terminal command ›** gives the equivalent command line. Stop or switch anytime.
- **Pipeline Builder** — a drag-and-drop GStreamer editor: pick elements from a searchable palette, wire them on the canvas, edit properties, then Run/Stop with live playback. Start

ⓒ Copyright 2026 DEEPX All Rights Reserved.

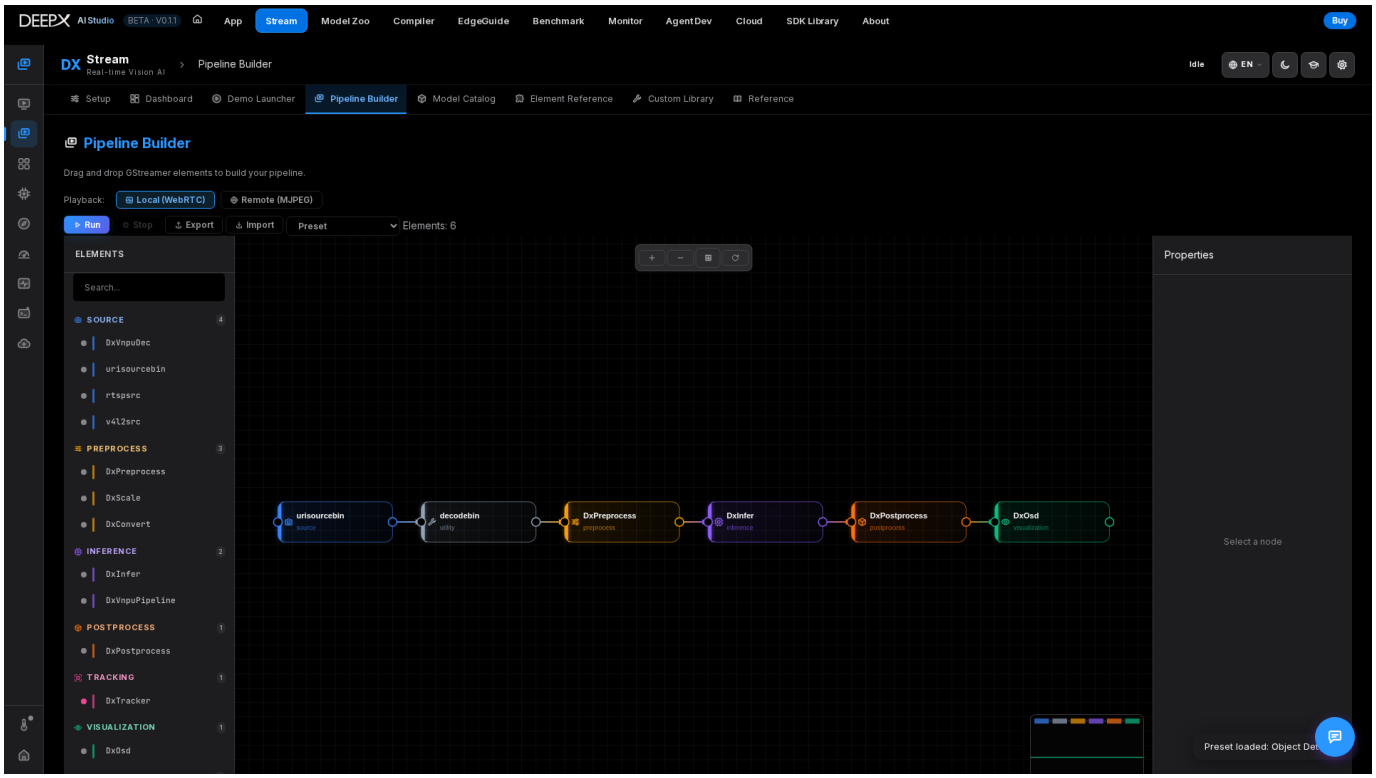
from a built-in preset, export / import the pipeline as JSON, and see the equivalent `gst-launch` command. Its **Playback** selector also offers **Local (WebRTC) / Remote (MJPEG)**; this choice is shared with Demo Launcher.

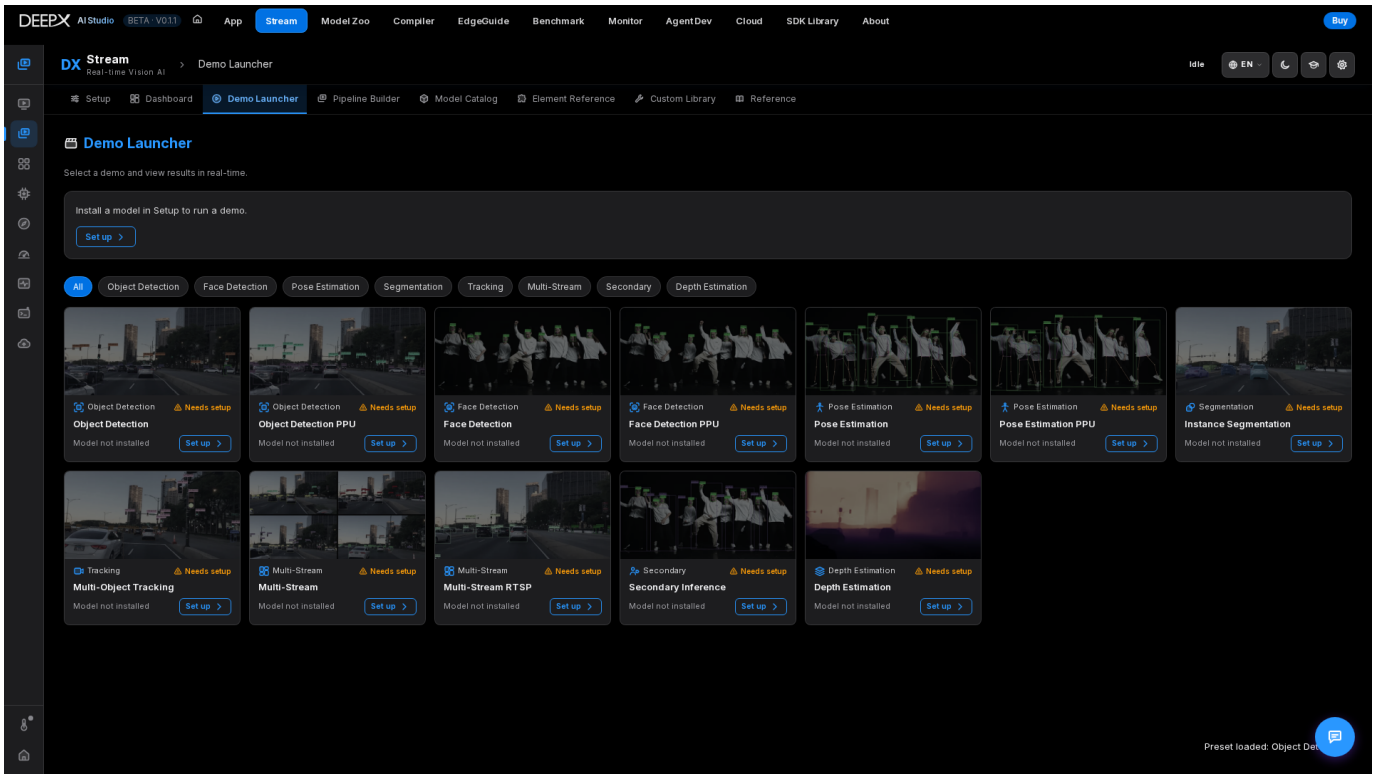
- **Model Catalog** — browse, search, and download models for use in pipelines.
- **Element Reference** — browse DX Stream's GStreamer elements by category, with their properties and pads.
- **Custom Library** — upload and build your own C post-processing library (with a build log), or upload a `.dxnn` model for DxInfer.
- **Reference** — searchable in-app documentation.

9.6.2 First pipeline

1. Open **Setup** and check the runtime / driver, plugin, models and playback dependencies. Installation is an explicit action and may need administrator access. A ready Dashboard is a summary, not a substitute for the launch checks.
2. For a preset, open **Demo Launcher**, choose a demo and a supported input, select **Playback**, then **Start**. Use **Remote (MJPEG)** through an SSH tunnel; WebRTC requires its system packages and a working peer-to-peer path.
3. To change the pipeline, open **Pipeline Builder**, load a preset, select an element to edit its properties, and check model / input paths on the Studio host. Choose **Playback** here before **Run**. Inspect video and reported metrics, then **Stop** before changing to a different job. JSON export preserves the pipeline definition; it does not package models or install plugins on another host.

ⓒ Copyright 2026 DEEPX All Rights Reserved.





9.6.3 Notes

- If a stream fails or stalls, a persistent error is shown (no silent black screen), with retry.
- If runtime validation blocks launch, open **Setup** → **DX-Runtime Dependencies** and **NPU Linux Driver**, inspect the failed check / **Deep Diagnostics**, and follow the suggested repair.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Re-check after an authorized runtime change or requested reboot; retry only when validation succeeds. Unsupported runtime versions are not silently replaced.

- For a missing model / plugin, finish the corresponding Setup step and confirm its status. For a black or stalled remote view, check **Playback**, connectivity and the error message before restarting the pipeline; browser connectivity alone does not prove inference is running.
- Existing model or video files without Studio completion evidence are **unverified**, not automatically missing. Run **Model & Video Download** to verify or complete the asset set. Without a completed video archive inventory, individual file records do not establish that the whole archive is verified.

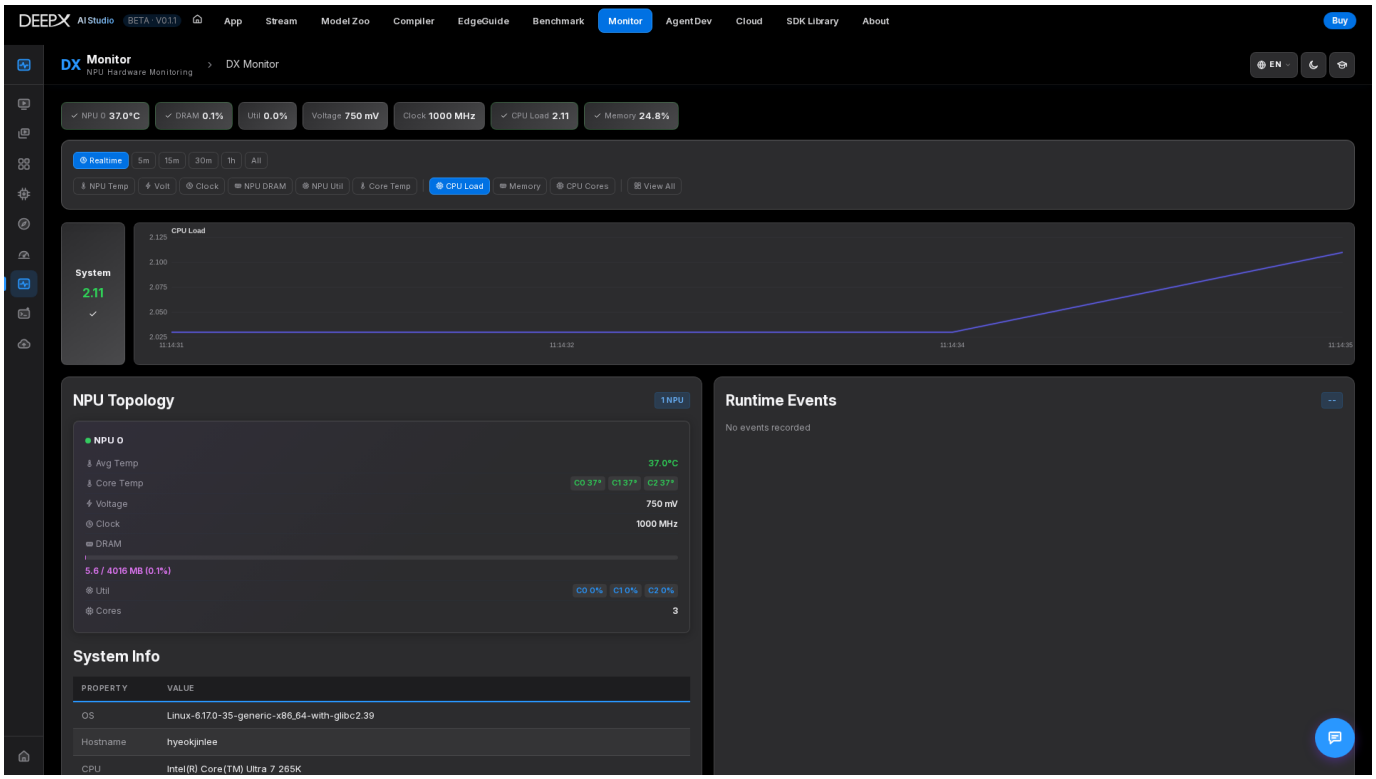
Related

DX Stream pipelines run the same `.dxnn` models compiled in **DX Compiler**.

© Copyright 2026 DEEPX All Rights Reserved.

9.7 DX Monitor

A real-time hardware dashboard for your DEEPX NPU and host — temperature, voltage, clock, DRAM, and utilization, alongside CPU and memory, plus the installed SDK / driver versions.



9.7.1 Using it

Step 1. Open the dashboard — live **NPU** telemetry per device (temperature, voltage, clock, DRAM, per-core utilization) and **system** stats (CPU per-core, load, memory).

Step 2. Pick a **time range** (Realtime / 5m / 15m / 30m / 1h / All — up to ~12h history) and switch which metric the charts show, or "View All" as a grid.

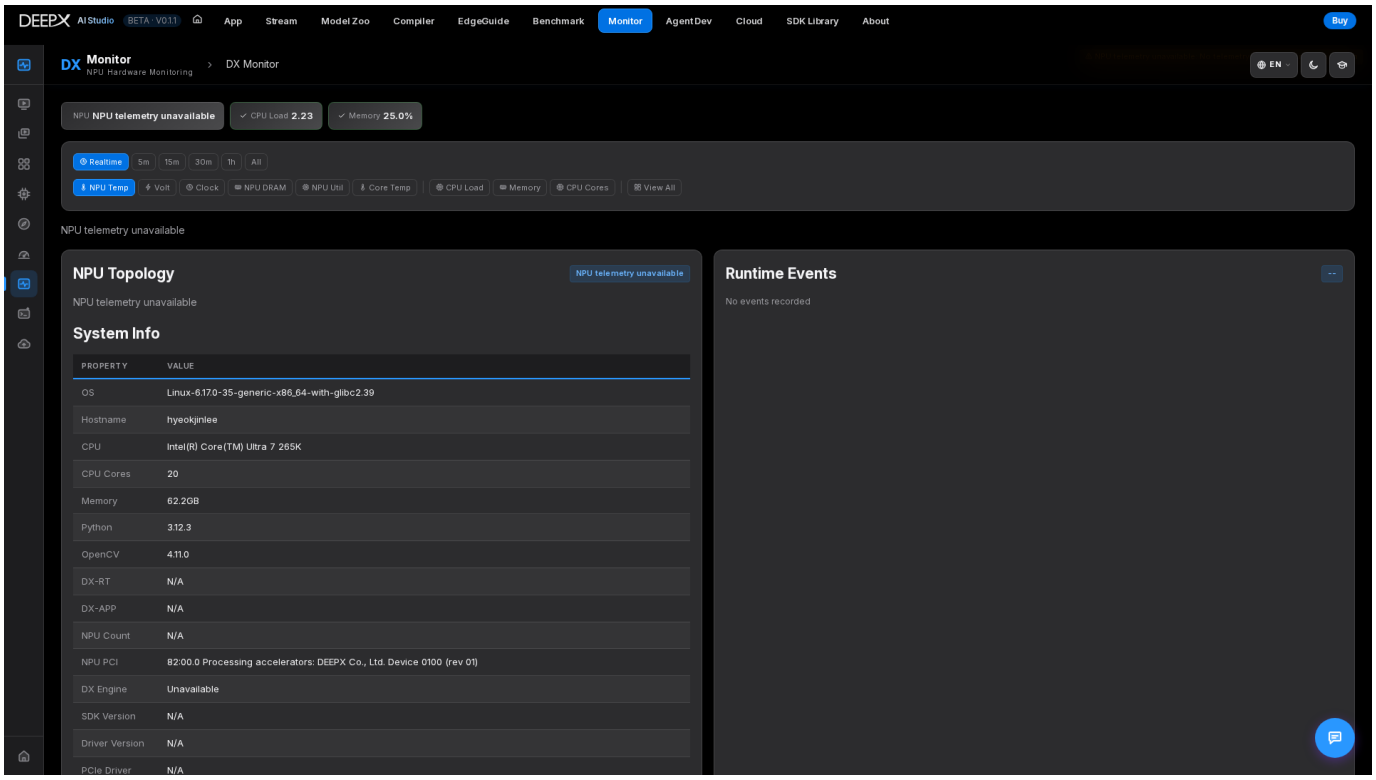
Step 3. Watch the live charts while you run inference or benchmarks in the other tools; charts stream over SSE (with a polling fallback shown when SSE is down).

For example, keep Monitor open, start an image or video run in DX App, and compare the NPU utilization / temperature before and during the run. Confirm that telemetry is **real and current** before interpreting the chart as evidence of that run.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.7.2 Read availability before the numbers

State	Meaning	What to do
Real / current	The telemetry worker is supplying current NPU samples.	Read per-device values together with the selected time window.
NPU telemetry unavailable	No usable NPU samples are available; host stats can still work.	Check the shown diagnostic, DX-RT / `dx_engine`, driver and `dxrt-cli --status`; repair the environment and re-check.
NPU telemetry is not current (stale)	The source is not providing current samples.	Do not treat retained values as live; check the diagnostic and source recovery.
Mock	Explicit UI demonstration data, enabled by `DX_MONITOR_EXPLICIT_MOCK=1`.	Do not use it as measured hardware or performance evidence.



ⓒ Copyright 2026 DEEPX All Rights Reserved.

An SSE connection reports **transport**, not NPU health: it can carry unavailable or stale status. The browser accumulates chart history while it receives samples (up to about 12 hours); it is not a server-side recording archive. A fresh page session starts collecting again, and reconnecting cannot recover samples missed while disconnected. Wait for fresh samples after recovery before judging a run.

9.7.3 Key features

- **Live NPU telemetry** per device with warn / critical **threshold alerts** (limit lines on the charts), per-core NPU temperature, and per-channel DRAM temperature.
- **NPU Topology** — firmware, chip / variant, board, DDR type / freq / size, per-core util.
- **DDR ECC alerts** — single-/double-bit error (SBE/DBE) warnings.
- **Runtime Events log** — INFO / WARN / ERROR / CRITICAL entries from the runtime.
- **System stats** — CPU per-core, load, memory.
- **Version info** — DX-RT / PCIe driver / DX Engine / SDK / OS / Python versions, uptime.
- With no NPU, shows **NPU telemetry unavailable** — it never shows made-up values; CPU and memory are always real. Mock telemetry appears only when started with `DX_MONITOR_EXPLICIT_MOCK=1` (marked Mock).

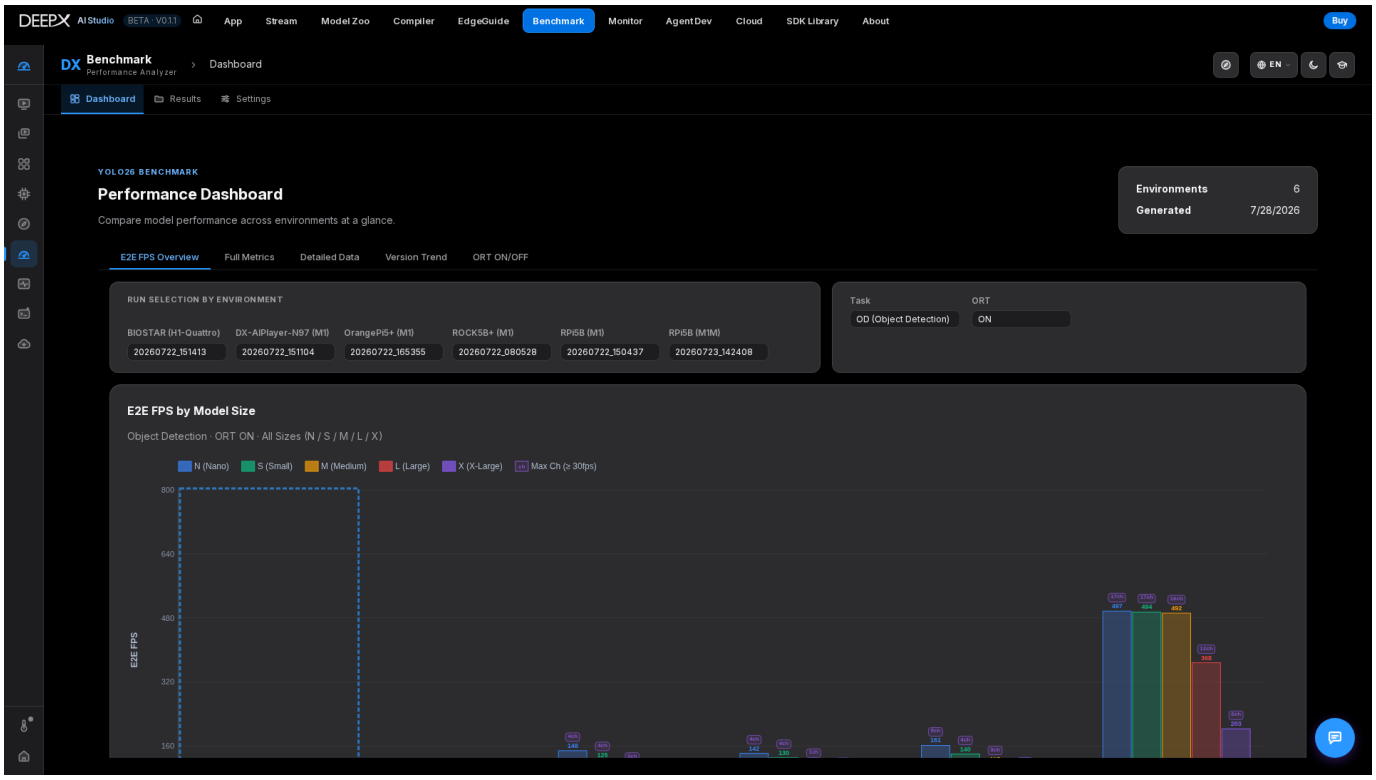
Skipping hardware initialization is not a request for mock telemetry. Only the explicit mock setting above enables synthetic NPU values.

> **Note**
>
> The same version and device information is available on the command line via `dxrt-cli --status`.

© Copyright 2026 DEEPX All Rights Reserved.

9.8 DX Benchmark

Browse and compare DEEPX NPU benchmark results in your browser — throughput, latency, and multi-stream numbers per model, with charts.



9.8.1 Pages

ⓒ Copyright 2026 DEEPX All Rights Reserved.

There are three top-level tabs: **Dashboard**, **Results**, and **Settings**.

- **Dashboard** — throughput (FPS), latency (ms), and multi-stream **channel** figures per model. Compare across **NPU environments / host PCs side by side**, filter by task and model size, toggle ONNX-runtime on/off, and click a chart bar or table row to drill in.
  Panels show environment details and the benchmarked-model specs. It has five sub-tabs:
  - **E2E FPS Overview** — end-to-end FPS by model size, with a run selector per environment.
  - **Full Metrics** — the full metric set (latency, throughput, E2E FPS, max channels) per model; drilling into an environment also shows its **NPU identity** (Product, SKU, Modules, Device Count) alongside host/driver/firmware details.
  - **Detailed Data** — a per-model, per-task table that also surfaces **NPU thermal data** (min/max temperature and clock, with a **Throttled** badge when thermal throttling was detected during the run) and a **run-status** column (Partial / Timeout / Error) for any metric that didn't complete cleanly.
  - **Version Trend** — a metric line chart across historical runs, to see performance move between versions.
  - **ORT ON/OFF** — a side-by-side comparison of ONNX-runtime on vs. off, per model size.
- **Results** — browse past runs by hardware and run-ID; open each run's Markdown report and raw JSON.
- **Settings** — the (read-only) deployment config: directories, thermal cooldown, iterations / warm-up / FPS threshold. Submitting a change is rejected by the server (HTTP 501, "deployment-fixed"); adjust these via configuration files before starting the server instead.

A one-click link hands the current selection to **DX EdgeGuide**, which uses the same data to recommend hardware.

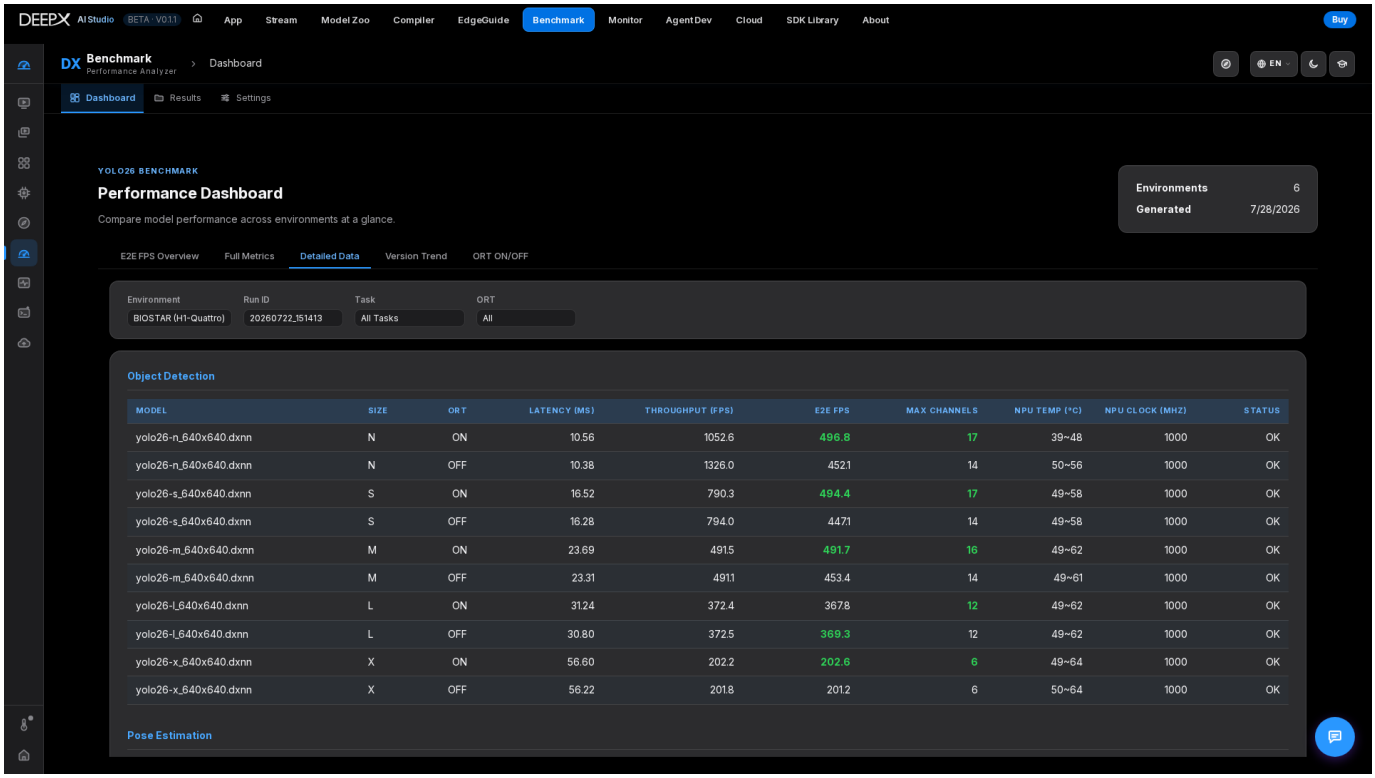
What "max-channel" means: the highest number of simultaneous video streams a model sustains on that NPU while meeting the FPS threshold.

9.8.2 Example: compare a model fairly

1. In **Dashboard** → **E2E FPS Overview**, select the host / NPU environments and their runs. Keep task, model size and ORT setting consistent; compare the displayed environment versions rather than assuming two boards ran identical software.
2. Click a chart bar to inspect the environment details. Select the **Full Metrics** sub-tab to compare model latency / throughput, end-to-end pipeline FPS and multi-stream channels.
3. Open **Detailed Data** and check thermal values, **Throttled** and run status. **Partial**, **Timeout** or **Error** is not a clean zero-performance result; use **Results** to read the run report / JSON before drawing a conclusion.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

4. If a run or metric is missing, choose an available run or produce new results with the standalone benchmark CLI. Changing this read-only dashboard does not execute a new benchmark or repair incomplete source data.

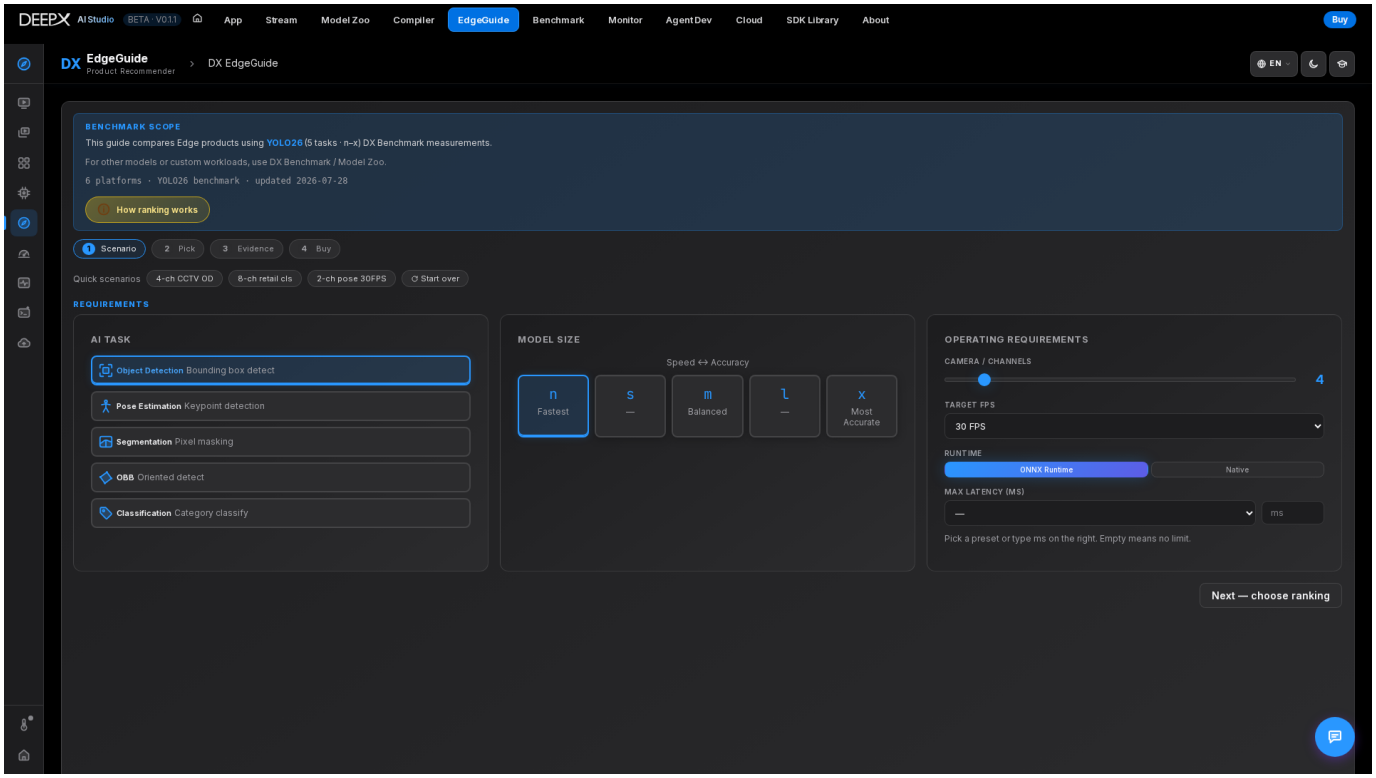


> **Note**
>
> The dashboard is **read-only**; benchmarks are produced by the standalone `dx-benchmark` CLI (`cd dx-benchmark && ./run.sh run`), and their results feed this view (and DX EdgeGuide).

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.9 DX EdgeGuide

Pick the best DEEPX Edge AI product (NPU board + host) for your computer-vision workload. Describe the workload; EdgeGuide ranks the platforms that can handle it using real **YOLO26 benchmark** measurements, then lets you drill into a platform, compare it with another, review the evidence, and get product info / a price quote.



9.9.1 Using it

The whole tool is one page with a four-step strip — **Scenario** → **Pick** → **Evidence** → **Buy** — where panels open as you go.

1. Describe the workload (Requirements panel)

- Optionally click a **Quick scenario** chip (**4-ch CCTV OD / 8-ch retail cls / 2-ch pose 30FPS**) to prefill.
- **AI task**: Object Detection / Pose Estimation / Segmentation / OBB / Classification.
- **Model size**: n (fastest) → x (most accurate).
- **Operating requirements**: cameras / **channels**, **target FPS**, runtime (ONNX Runtime vs Native), **max latency**.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

- Click **Next — choose ranking**, choose a **ranking priority** — Most channels / Best Performance (camera-pipeline FPS) / Lowest Power (NPU TDP; host power not included) — then **Get Recommendations**. (After the first run, changing any input auto-refreshes the results; **Start over** clears everything.)

2. Read the recommendations

A verdict line shows the **Top pick** and whether your channel / FPS target is met. Each card carries a **Measured** or **Estimated** badge and one line saying why — for example "12 ch held 30.4 FPS each ≥ 30 FPS target (measured) · 13 ch dropped to 28.1 FPS", or the division an estimate comes from, "534 FPS ÷ 60 FPS = 8.9 → 8 ch (estimated from the measured total; tested up to 18 ch)".

How the channel count is worked out: the multi-stream benchmark measured the stream counts around 30 FPS per channel. When a measured count holds your target and the next one misses it, that count is the answer (measured). If every measured count holds the target, the answer is the larger of that count and `floor(T_sat / target)`; if none holds it, `floor(T_sat / target)`. `T_sat` is the lowest measured total FPS — around saturation the total stays nearly flat as streams are added — and an estimate never exceeds twice the highest tested count; **Estimated+** means the estimate hit that cap, so the real capacity may be higher. The quotient is shown cut to one decimal, never rounded up past the floor. A measurement taken while the NPU throttled is kept (its FPS already ran at throttled clocks) but flagged, and ranks after an equal cool result. The **limit** note says **NPU** when the NPU was at least 85 % busy at the most loaded measured stream count, otherwise **host** (CPU / decode / PCIe), and shows that NPU load — a faster host helps in the second case, a bigger NPU in the first. Notes also say when latency was not measured — with a latency budget set, an unmeasured latency never counts as a plain pass. **How ranking works** shows every formula with its meaning in your language, plus a summary of the current session. Click a card (or a bar in the throughput chart) to open details.

3. Explore / compare

The Details panel shows key facts, platform specs, a Performance Radar, a sortable benchmark table, and a per-model-size chart; **task tabs** let you view the same platform's numbers for other AI tasks. The charts and table are interactive — click a model-size bar or a table row to re-run the recommendation at that size. Use **Compare with** to line up a second platform side by side (with a summary count of Meets / Insufficient).

4. Buy / contact

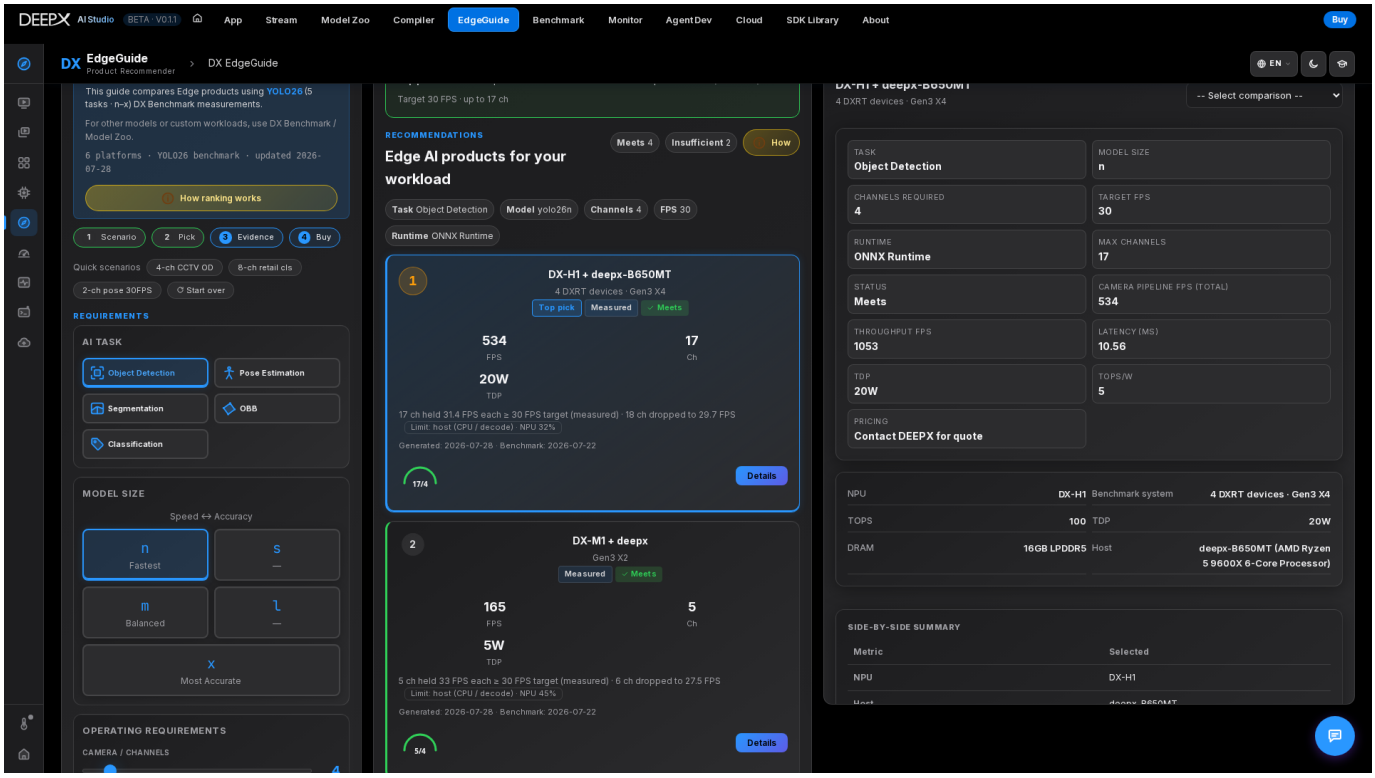
ⓒ Copyright 2026 DEEPX All Rights Reserved.

Pricing is by quote — the Buy panel's **Product info** and **Request quote** buttons go to DEEPX.

Example: check a four-camera target

Choose **4-ch CCTV OD**, keep Object Detection, a fixed model size, **30 FPS** and one runtime setting, then request recommendations. Open the top card and **Evidence**; compare a second platform without changing those inputs. Read the badge and arithmetic: **Measured** brackets the target with tested stream counts, while **Estimated** derives a count from measured throughput. Neither is a production guarantee for a different camera, decoder or host.

If nothing meets the target, inspect the channel / latency shortfall, try a smaller model or fewer channels, and rerun. Missing latency is not proof that a latency budget is met; **Estimated+** is a capped estimate, not additional tested capacity.



EdgeGuide recommends **hardware from benchmark evidence**. It does not create AWS resources or deploy an application; those cloud workflows are in DX Cloud.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

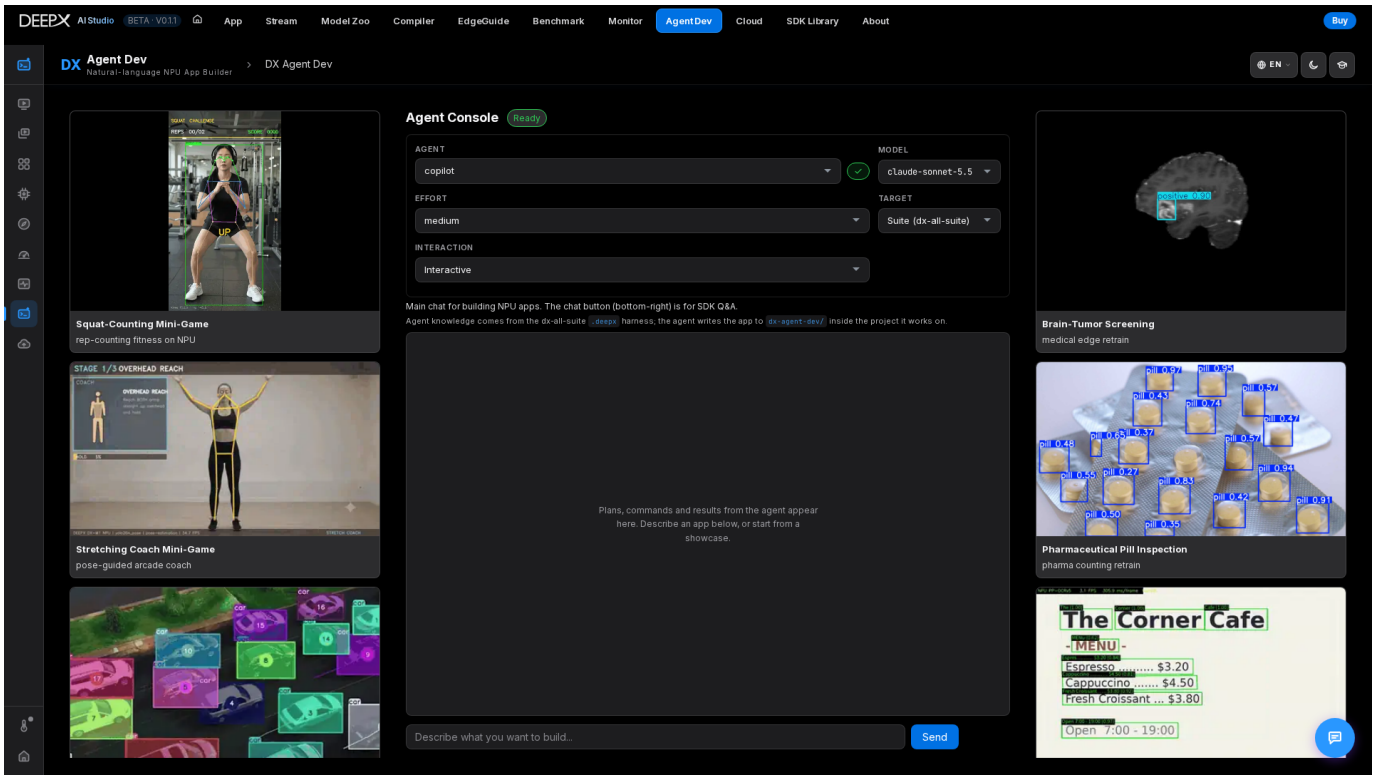
9.9.2 Key features

- **Real measurements** — recommendations come from the YOLO26 benchmark matrix, not spec-sheet math. Channel counts are measured where a tested stream count brackets the target, and otherwise estimated from the measured total FPS — every card says which.
- **Open methodology** — the **How ranking works** button shows the exact formulas and a live summary; it also states the honest limits (the figures are not a full spec sheet; the power ranking covers the NPU only; contact DEEPX for production sizing and pricing).
- **Multi-stream evidence** — per-channel and total FPS behind each channel estimate.
- **Shareable & resumable** — the current selection is captured in the URL (a link, e.g. arriving from DX Benchmark, reproduces the inputs and re-runs), and your last inputs are also restored locally on reload.
- **AI Help** chat for follow-up questions grounded in your requirements / results.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.10 DX Agent Dev

A natural-language console for building DEEPX NPU apps: describe what you want, and a coding agent generates and runs it. It is also a chat assistant and a gallery of ready-made showcases.



9.10.1 Using it

Step 1. **Connect an agent** — the console detects installed coding CLIs. Five are supported: **Claude Code** (`claude`), **GitHub Copilot CLI** (`copilot`), **Cursor** (`cursor-agent`), **Codex** (`codex`), and **OpenCode** (`opencode`). A badge shows whether each is signed in; if a login is needed it shows the exact command to run, then re-checks.

Step 2. **Set the run** using the console controls:
- **Agent picker** — which of the 5 CLIs to use. The agent and model you pick are saved as the Studio agent (`~/.dx-ai-studio/agent.json`) and shared by Home, Agent Dev and the chat.
- **Model picker** — a dropdown of that agent's models; for CLIs that support listing them (Copilot — the models this account may use —, Cursor, OpenCode) it's enumerated live from the CLI, otherwise it falls back to a built-in list. A hint warns if the chosen model is too weak and suggests a stronger one.
- **Reasoning-effort picker** — the effort levels that agent's CLI supports.
- **Target** — which repo the agent works in: Suite, dx-runtime, dx_app, dx_stream, dx-

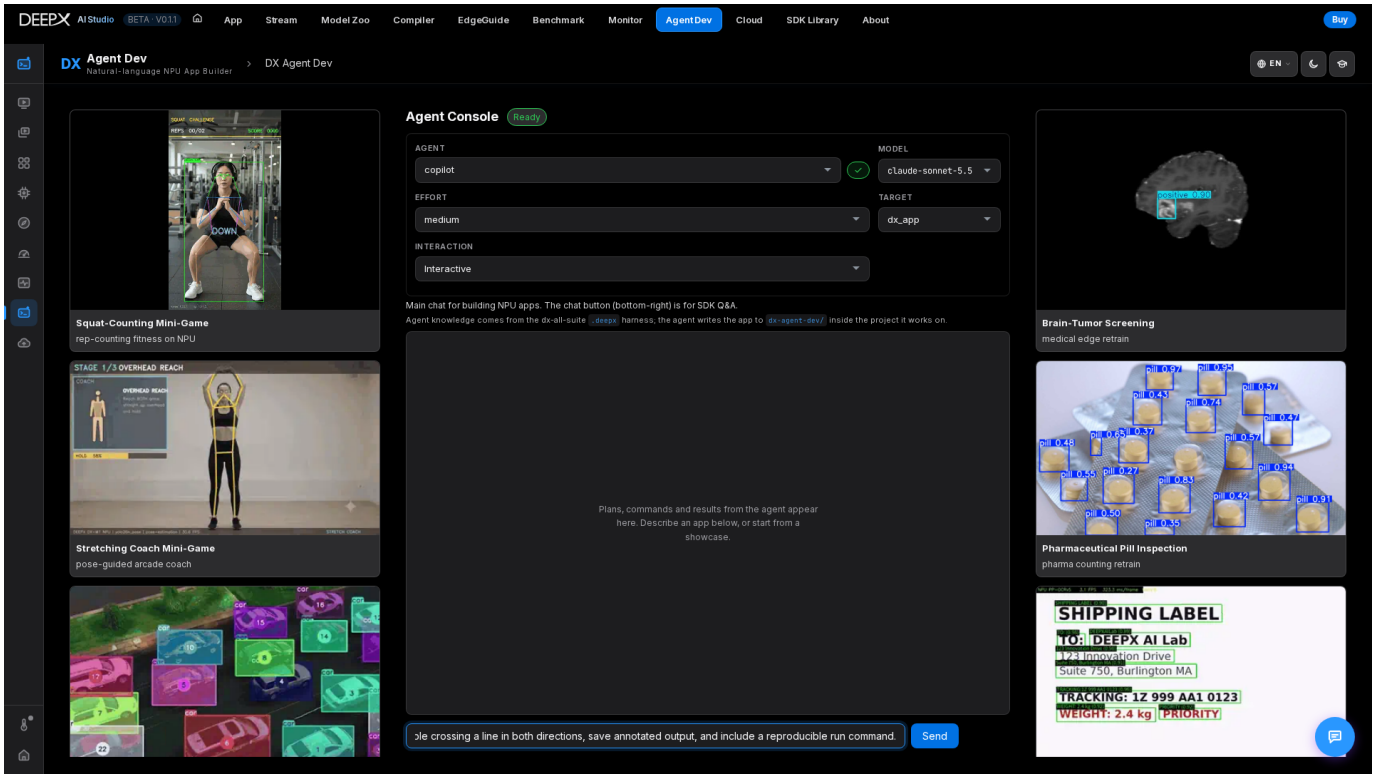
ⓒ Copyright 2026 DEEPX All Rights Reserved.

compiler.
- **Interaction** — Interactive (asks questions) or **Autopilot (no questions)**.

Step 3. **Describe your task** in the prompt — the agent works and streams its progress live; an expandable **activity panel** shows each turn's shell / tool output next to the reply. The agent writes the generated app to `dx-agent-dev/<session_id>/` inside the sub-project it worked on (for example `dx-runtime/dx_app/dx-agent-dev/…`). The same run can also be started from the **prompt box on the hub home** and followed there (activity, replies, resume).

Step 4. **Chat** — the main console chat is for building NPU apps via the connected coding agent. The **chat button** (bottom-right) is a separate affordance: a general SDK Q&A assistant, not part of the Agent Console run.

Step 5. **Showcases** — browse the gallery of example NPU apps for reference. Open a showcase to inspect its README, recorded workflow and artifacts; viewing these does not start a new agent or NPU run.



Follow up, stop or recover

For example, choose **Target** → **dx_app** and **Interactive**, ask the agent to explain the input / output plan for an image-inference app, then reply to its question in the same console. Follow-up messages continue the conversation when the CLI supports resume.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Inspect **Agent activity** and the generated files before treating a build as usable; an agent's prose alone is not verification.

Use **Stop** during a run to request cancellation and wait for **Stopped**. Cancellation does not undo files or external actions already performed. If the connection drops, read the displayed error / status and reconnect to check the existing run before submitting the task again. Reply in the retained conversation to continue where supported; do not assume a disconnected browser means the work completed or was undone.

If no CLI is available, or the harness isn't found

- **No coding-agent CLI installed/authenticated** — install and log into one of the 5 above, then reload the page. Showcases remain useful for browsing. The separate chat assistant can use a configured local / cloud provider without a coding CLI; with no connected AI it gives basic built-in guidance, not a real AI build.
- **`.deepx` harness directory not found** — the agent knowledge/skills directory is missing. Set the `DX_HARNESS_ROOT` environment variable to your dx-all-suite checkout, or run DX AI Studio from within the suite, then reload the page.

9.10.2 Key features

- **Bring-your-own agent** — uses whatever coding CLI you have installed and signed in; one login powers both the generator and the chat.
- **Studio chat (the button at the bottom right)** — separate from the Agent Console; it answers through your signed-in coding agent by default, or a **local / self-hosted** OpenAI- or Ollama-compatible endpoint (models are auto-discovered) or a cloud API key; **Refresh knowledge** for the DEEPX SDK documentation is under Settings ▸ Advanced.
- **Live streaming** of the agent's work, with a wall-clock timeout so a stuck run frees the slot for the next request.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.11 DX Cloud (AWS)

DEEPX on AWS from inside Studio: connect an AWS account, check the two Marketplace products, create or select a CloudFormation stack, then compile an `.onnx` into a `.dxnn` — no local `dxcom` needed. The Greengrass product also provides device and runtime-component status and a device registration script. Open **DX Cloud** from the hub tile, at `/cloud/`, or with `Alt` + `9`.

The internal Python package is named `dx_edge`. This page is not **DX EdgeGuide** (the product recommendation tool) or the suite's separate `dx-edge` guides.

9.11.1 AWS Marketplace

- **DEEPX Compiler Solution** (AWS Marketplace) runs `dxcom` on an Amazon EC2 instance. Deployed as a CloudFormation stack it becomes an event-driven pipeline: an `.onnx` and its `.json` uploaded to the stack's model bucket start a Lambda trigger → Step Functions → a compiler instance, which writes the `.dxnn` next to them. The workflow includes instance termination; verify cleanup separately from the presence of a result file.
- **DEEPX Greengrass Solution** (AWS Marketplace) deploys the `com.deepx.dx-runtime` component (NPU driver, firmware, `dx_rt` and `dx_stream`) to compatible, registered Greengrass core devices in an IoT Thing Group, with a cloud compile pipeline in its stack. This is runtime deployment, not automatic deployment or inference of the compiled model.

The full walkthrough — stack parameters, ZTP checks, cost and cleanup — is in the DX-Edge guides.

9.11.2 Connect to AWS

Studio's base web runtime uses Python 3.8+ and the standard library. **Actual AWS API operations require optional `boto3`**, installed in the Python environment running Studio, for example `python3 -m pip install -e '.[aws]'` from the Studio package root. Without it the page and guide remain available, but AWS operations cannot run.

DX Cloud uses boto3's credential chain on the **Studio host**, optionally selecting a named profile from `~/.aws`. It has no UI for entering, uploading or storing access keys. Configuring credentials only on a remote browser's computer does not configure the Studio host. The AWS CLI commands below help set up credentials or SSO; the backend calls boto3, not the `aws` executable. The same setup guide is in step ① of the page, with a copy button on every command.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

1. Install AWS CLI v2. One script for Linux `x86_64` and `aarch64` — it picks the package for this machine (install `unzip` and `curl` first if they are missing: `sudo apt install -y unzip curl`):

```
ARCH="$(uname -m)"   # x86_64 or aarch64
case "$ARCH" in
  x86_64|aarch64)
    curl "https://awscli.amazonaws.com/awscli-exe-linux-${ARCH}.zip" -o awscliv2.zip
    unzip -q awscliv2.zip
    sudo ./aws/install --update
    aws --version ;;
  *) echo "AWS CLI v2 for Linux supports x86_64 and aarch64 only (this machine: $ARCH)" ;;
esac
```

See the AWS CLI install guide.

2. Configure credentials (AWS CLI configuration guide).

- *Option A — access key:* create one in the IAM console (Users → Security credentials → Create access key), then run `aws configure`:

  ```
  $ aws configure
  AWS Access Key ID [None]: AKIA****************
  AWS Secret Access Key [None]: ****************************************
  Default region name [None]: ap-northeast-2
  Default output format [None]: json
  ```

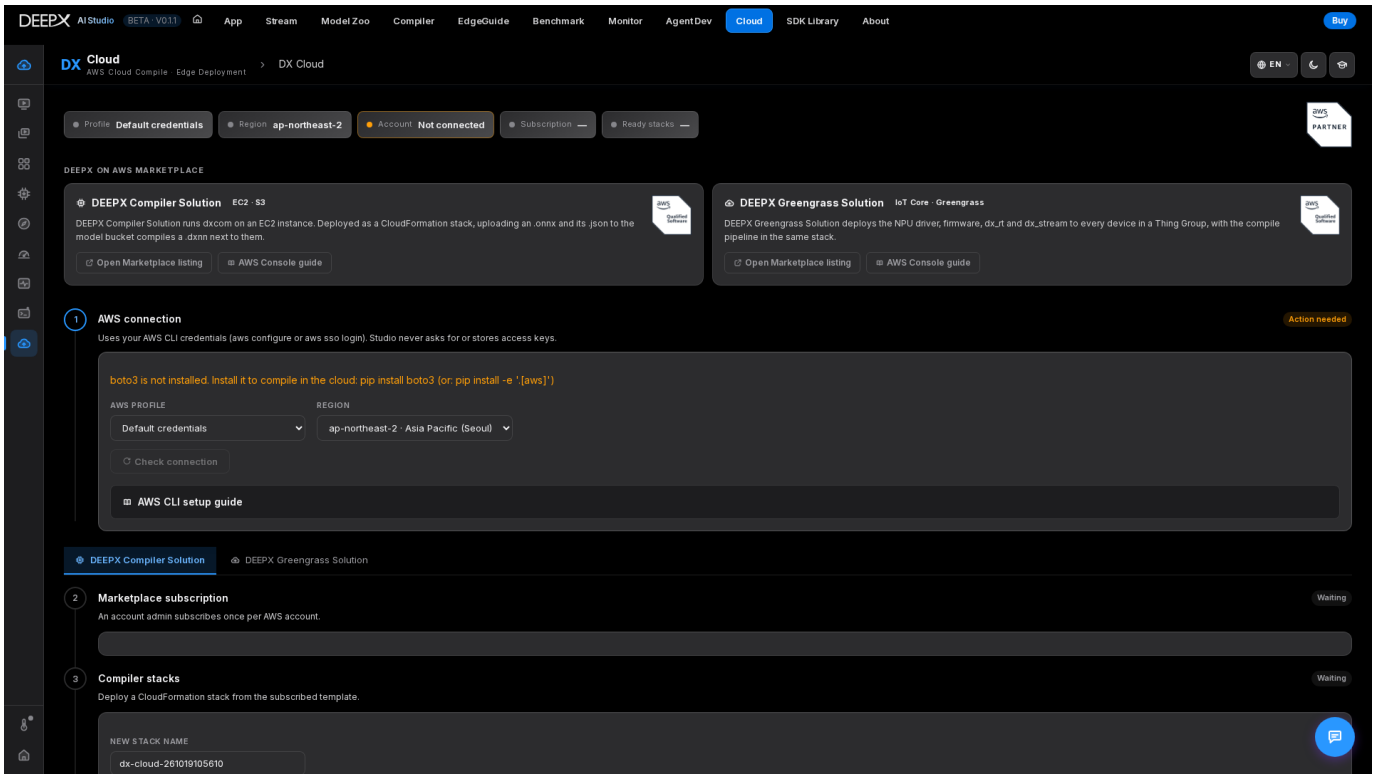
- *Option B — IAM Identity Center (SSO):* `aws configure sso` once to create a named profile, then `aws sso login --profile <name>` whenever the session expires.

3. Verify with `aws sts get-caller-identity` (add `--profile <name>` for the named profile you intend to use).

4. Permissions. The identity needs CloudFormation, S3, Step Functions, SSM `GetParameter`, EC2, IAM (stacks are created with `CAPABILITY_IAM`), CloudWatch Logs and Lambda; the Greengrass tab also needs Greengrass V2 and IoT. An account admin accepts the Marketplace subscription.

5. Check connection. Select **AWS profile** and **region**, then press **Check connection** in step ①. The profile list reloads without a restart and the account and ARN appear when identity verification succeeds. The region initially follows the profile, falling back to `ap-northeast-2`. If SSO has expired, sign in again on the Studio host, then repeat the check. A successful identity check does not establish permissions for every subsequent operation.

ⓒ Copyright 2026 DEEPX All Rights Reserved.



Preparation-state overview: boto3 is not installed and AWS is disconnected. These three UI views were captured without AWS operations and do not demonstrate successful compilation or deployment.

9.11.3 Subscribe and prepare a stack

1. Choose **DEEPX Compiler Solution** or **DEEPX Greengrass Solution**. Each product has its own **Marketplace subscription** check; one subscription does not establish the other's status. An account admin accepts the relevant Marketplace terms, then you use **Refresh** in Studio.
2. The check resolves the selected product template's AMI SSM parameter in the chosen account and region. It is not a billing check or a guarantee that a particular AMI version, region or instance type is currently available. Check the Marketplace listing and account permissions if it fails. The bundled template labels and older AWS Console guide images are reference versions, not live availability claims.
3. Use **Find stacks** to select an existing compatible stack, or enter a stack name and create one in Studio. Watch its status and recent CloudFormation events. Wait for a ready stack (`CREATE_COMPLETE` or `UPDATE_COMPLETE` with `ModelBucketName`) before uploading a model. A failed or in-progress Studio-created stack can remain visible for diagnosis; visible does not mean ready.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

> **Deployment uses real AWS resources**
>
> Review software subscription terms and AWS resource costs before creating a stack or compiling. Storage and logs can incur charges even when no compiler instance is running.

Studio stack creation uses the selected region's **default VPC and first returned default subnet**, generates a bucket name, and leaves the remaining vendored-template parameters at their defaults (including the compiler instance type). It requests `CAPABILITY_IAM` because the templates create IAM resources. The Studio form is not a custom VPC/subnet/instance editor. If the default network is missing, have an admin prepare it or use the AWS Console deployment guide to create a suitably configured stack, then find that compatible stack in Studio. The subnet must reach the AWS services the pipeline needs.

For a create failure, read the recent resource event reason; the CloudFormation Console has the full event history. Correct subscription, IAM or network problems before retrying. Do not repeatedly create new stacks to hide a failed one.

9.11.4 Cloud Compile

ⓒ Copyright 2026 DEEPX All Rights Reserved.

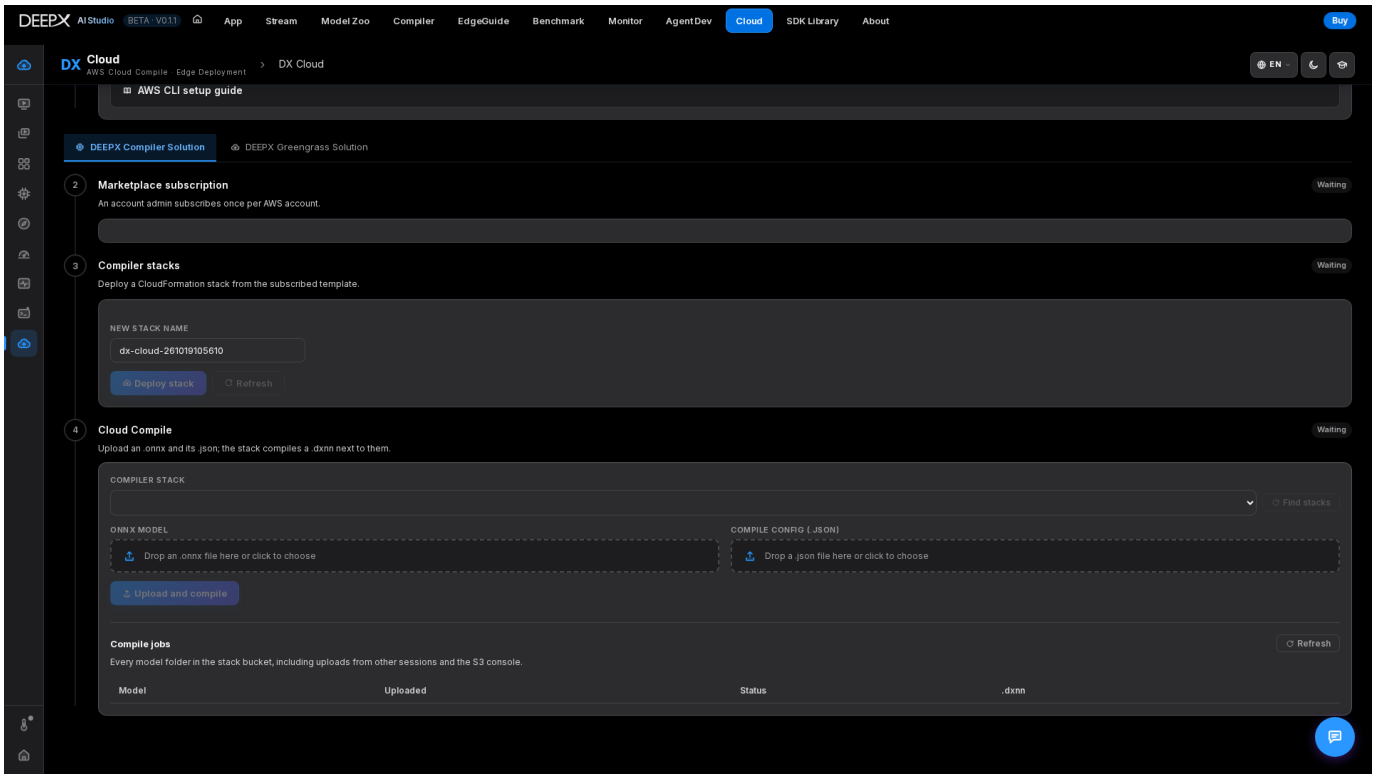
Both product tabs provide Cloud Compile:

1. With the connection verified, choose a **ready stack** in the appropriate product tab. Studio reads the bucket from the stack outputs; you do not type a bucket.
2. Choose the **ONNX model** and **compile config (.json)**, then press **Upload and compile**. ONNX is limited to **500 MiB**; JSON to **1 MiB** and must parse as a JSON **object**, not an array or scalar. These checks do not prove that the compiler supports the model or that the configuration is semantically correct.
3. Studio uploads the pair to a timestamped `models/<model>-<YYYYmmdd-HHMMSS>/` prefix. S3 triggers Lambda → Step Functions → EC2 compilation. The panel polls every 15 s and reports *pending*, *running*, *succeeded* or *failed*. Duration depends on model, calibration, instance capacity and AWS startup conditions; no completion time is guaranteed.
4. Use **Compile jobs** to revisit jobs in the selected stack, **Show log** for workflow steps and available compiler output, or the links to Step Functions, CloudWatch and S3. If the page was closed, reconnect, select the same account, region and stack, then refresh the job list rather than uploading again blindly.
5. When a `.dxnn` is available, choose its destination:

Action	Destination and effect
Download .dxnn	Downloads through the browser.
Save to Studio models	Writes to the **Studio server's** DX App models directory (`DX_APP_MODELS_DIR` , default `dx-runtime/dx_app/assets/models`); **overwrites the same filename**. It does not save on the browser host or deploy to a device.

For example, after a job completes, download the result for transfer elsewhere, or save it on the Studio host for later DX App use. Saving alone neither installs a runner nor verifies model compatibility or inference.

ⓒ Copyright 2026 DEEPX All Rights Reserved.



Cloud Compile workspace before AWS connection; no successful compile job is shown.

> **Result and cleanup are different states**
>
> Studio treats the existence of a `.dxnn` as compile success. The EC2 termination step can still be pending or fail afterwards. Check the workflow and EC2 resources before assuming all compute charges have stopped.

If a job remains pending, check that both input objects reached the same S3 prefix and that a Step Functions execution started. On failure, inspect **Show log** and CloudWatch before correcting the configuration or AWS prerequisites and submitting a new job. An access error can prevent status/log retrieval even when an AWS job exists.

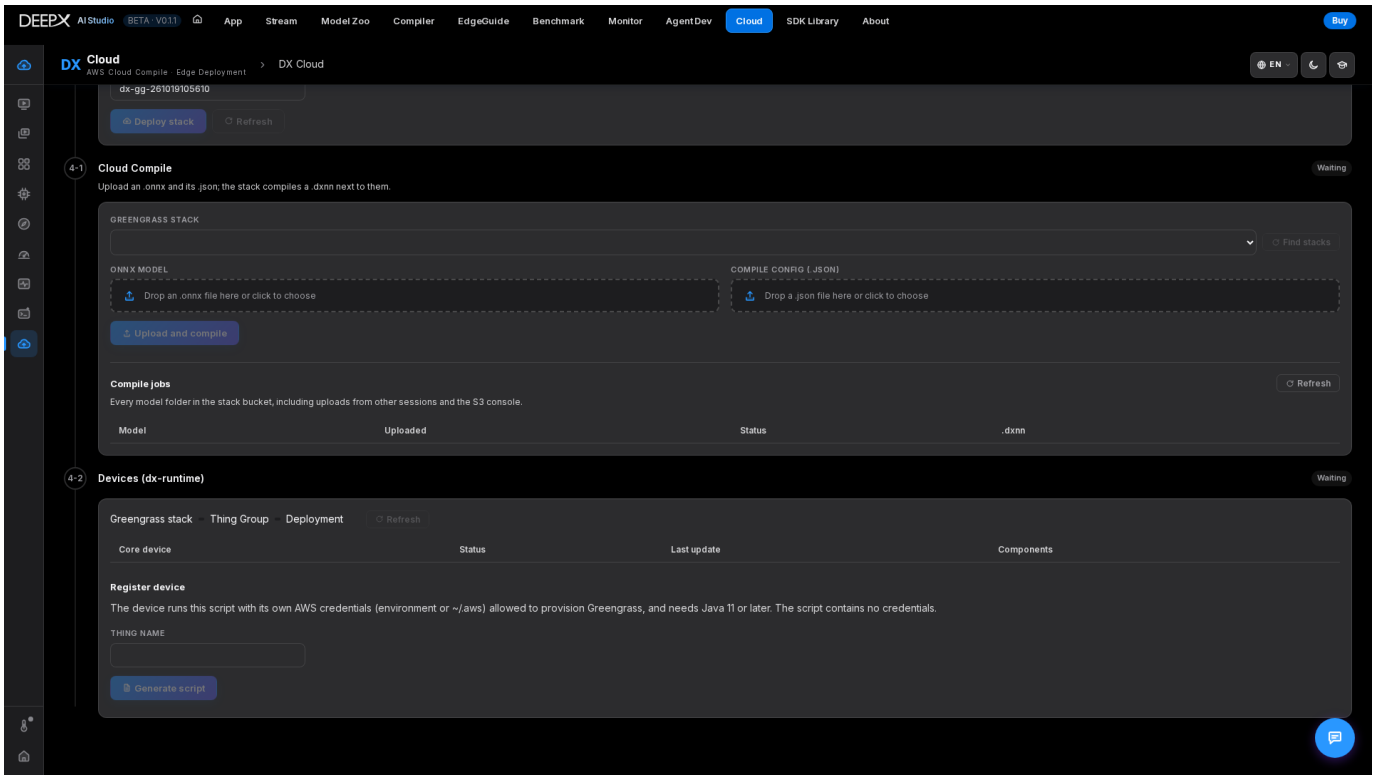
9.11.5 Greengrass devices and registration

1. Select a ready **Greengrass** stack. **Devices (dx-runtime)** shows its Thing Group, deployment status and registered core devices; open a device's component details to inspect the DEEPX/ Greengrass component versions and lifecycle states.
2. To register another device, enter its Thing name and generate the install script. Use **Copy** or **Download .sh** and review it before running it **on that device**. The script contains stack/ region/group/role settings, not AWS credentials.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

3. Prepare the device: Java 11+, `curl`, `unzip`, `sudo` permissions, device-side AWS credentials permitted to provision Greengrass, and network access to AWS and the runtime artifacts. Supported hardware/OS, kernel headers and runtime build requirements still apply; Studio-host credentials are not transferred to it.
4. Run the reviewed script yourself on the device, then **Refresh** Devices and check its installed components. If registration fails, examine the script's terminal error and device Greengrass logs, permissions and connectivity; an empty device list is not proof of a successful registration.

Studio only generates, copies or downloads the script. It does not SSH into the device or execute it. The stack's deployment of `com.deepx.dx-runtime` installs the runtime component; it does not automatically send the compiled `.dxnn` to devices or run inference.



Greengrass Devices setup area before AWS connection; no device registration or deployment success is shown.

9.11.6 Delete a stack and finish cleanup

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Select the stack's delete action and type its **exact stack name** in the confirmation dialog. Deletion is blocked while a stack operation is in progress. Follow deletion status and inspect CloudFormation events if it fails.

- The model S3 bucket has **Retain** policies and remains, including source models, results and object versions. If no longer needed, review and remove those objects and versions, then delete the bucket separately.
- Check for running compiler EC2 instances and leftover chargeable resources.
- Greengrass stack deletion does not uninstall software on devices or automatically remove their IoT identities, certificates and local runtime. Review devices and deployments separately; group membership can require cleanup for stack deletion.
- Marketplace subscriptions are separate. Cancel them through AWS Marketplace when no longer needed and review billing; deleting a stack is not a complete cost reset.

The built-in **AWS Console guide** retains its Console/terminal illustrations. These are separate from the three Studio UI views above; parameters and versions can differ from the currently vendored templates.

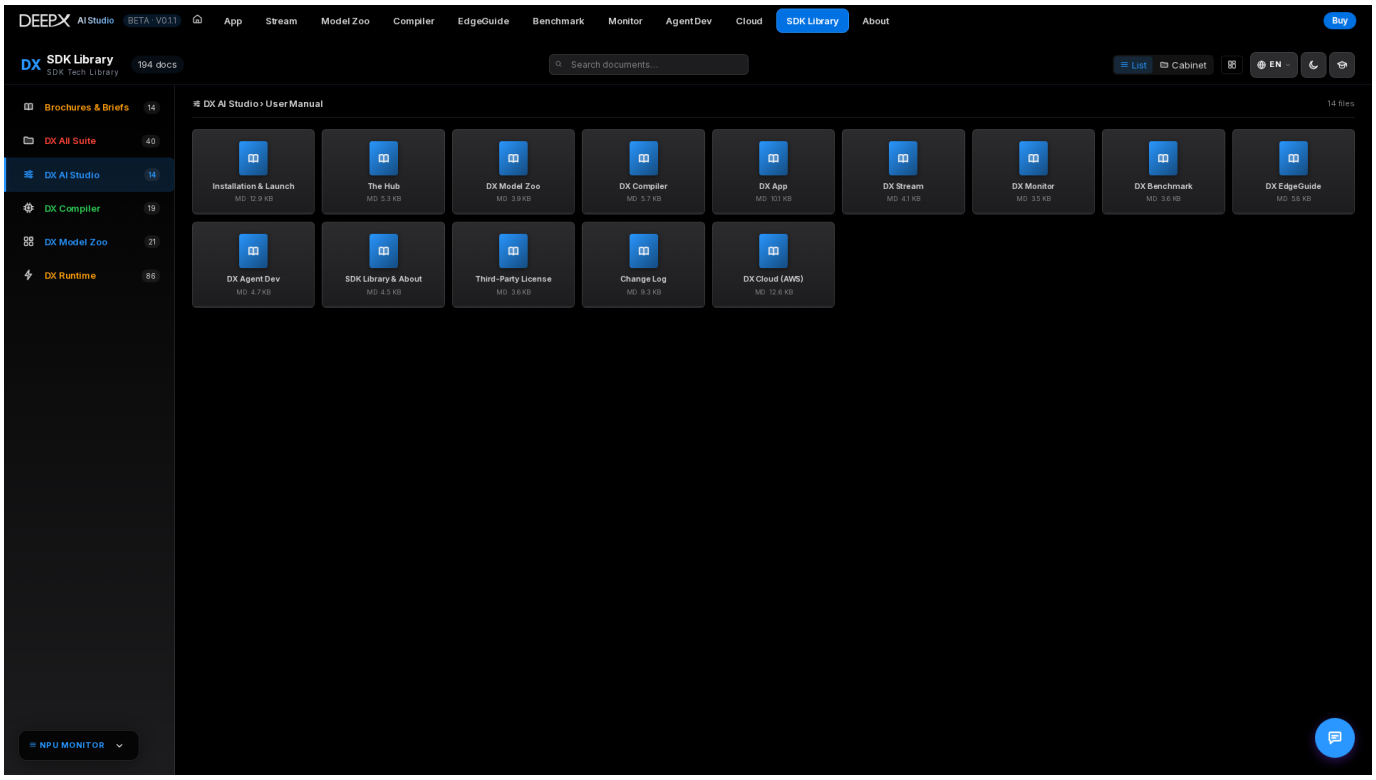
Related

To compile locally instead, see **DX Compiler**.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.12 SDK Library & About

Besides the nine tools, the hub hosts two reference views.



9.12.1 SDK Library

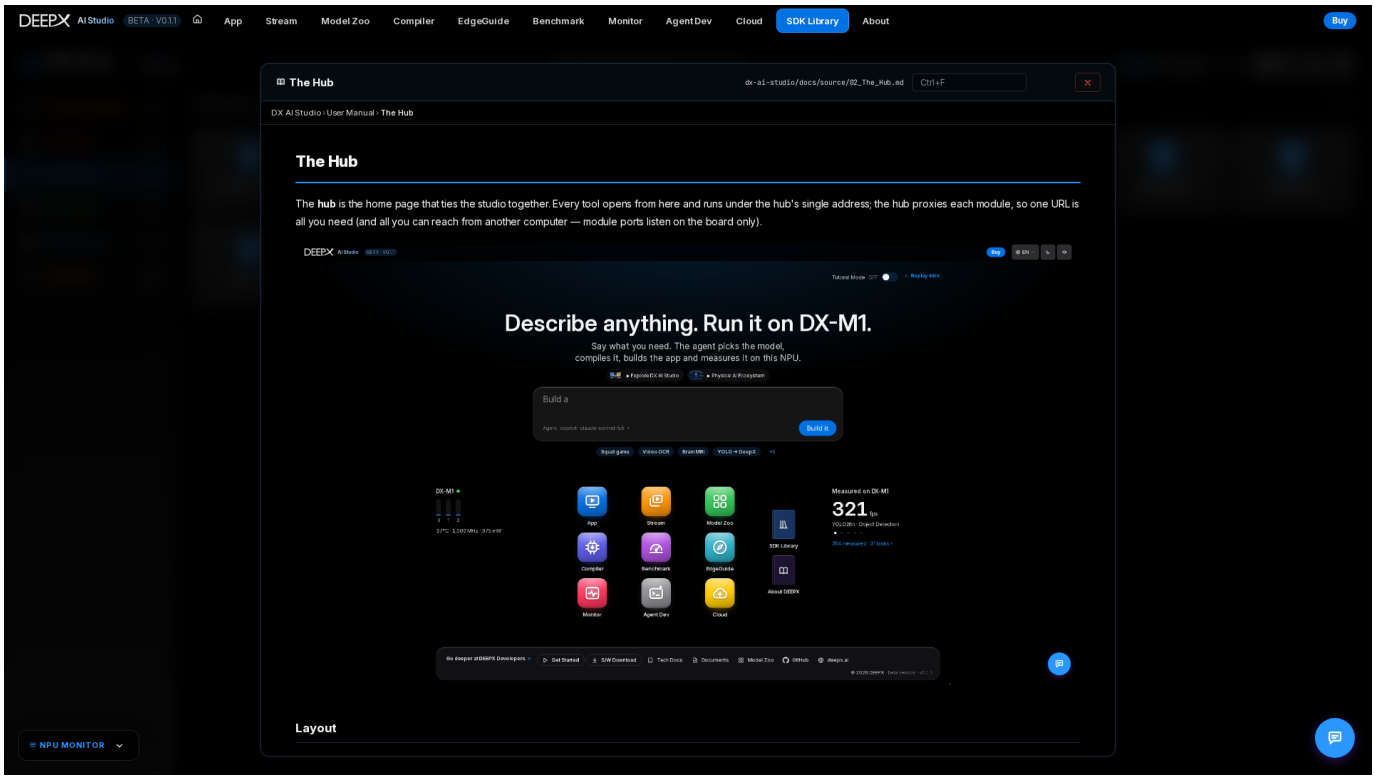
An
in-app library of DEEPX documentation and marketing material, reachable from the hub.

- **Documentation** — renders the DEEPX SDK markdown docs (architecture overview, environment setup, first-model walkthrough, version compatibility, FAQ, and each sub-project's guides) directly in the browser, with images and tables.
- **Brochures & Briefs** — product brochures and chip / module briefs as in-app PDFs.
- **Search & deep links** — library-wide search plus in-document find (`Ctrl+F` , with next / prev and a match counter). Toggle **list / cabinet** view, or open the **architecture diagram** overlay. The open document, view mode, and query are captured in the URL, so a link reopens exactly that state.

For example, select a documentation category, search for **runtime**, and open a matching guide. **List / Cabinet** changes how library entries are displayed; it is not the document viewer itself. Inside the opened document, use `Ctrl+F` to find a term and copy its URL to reopen that document. If an entry is unavailable, check that the corresponding suite

ⓒ Copyright 2026 DEEPX All Rights Reserved.

documentation is present; a library entry alone does not install the SDK or fetch missing source files.

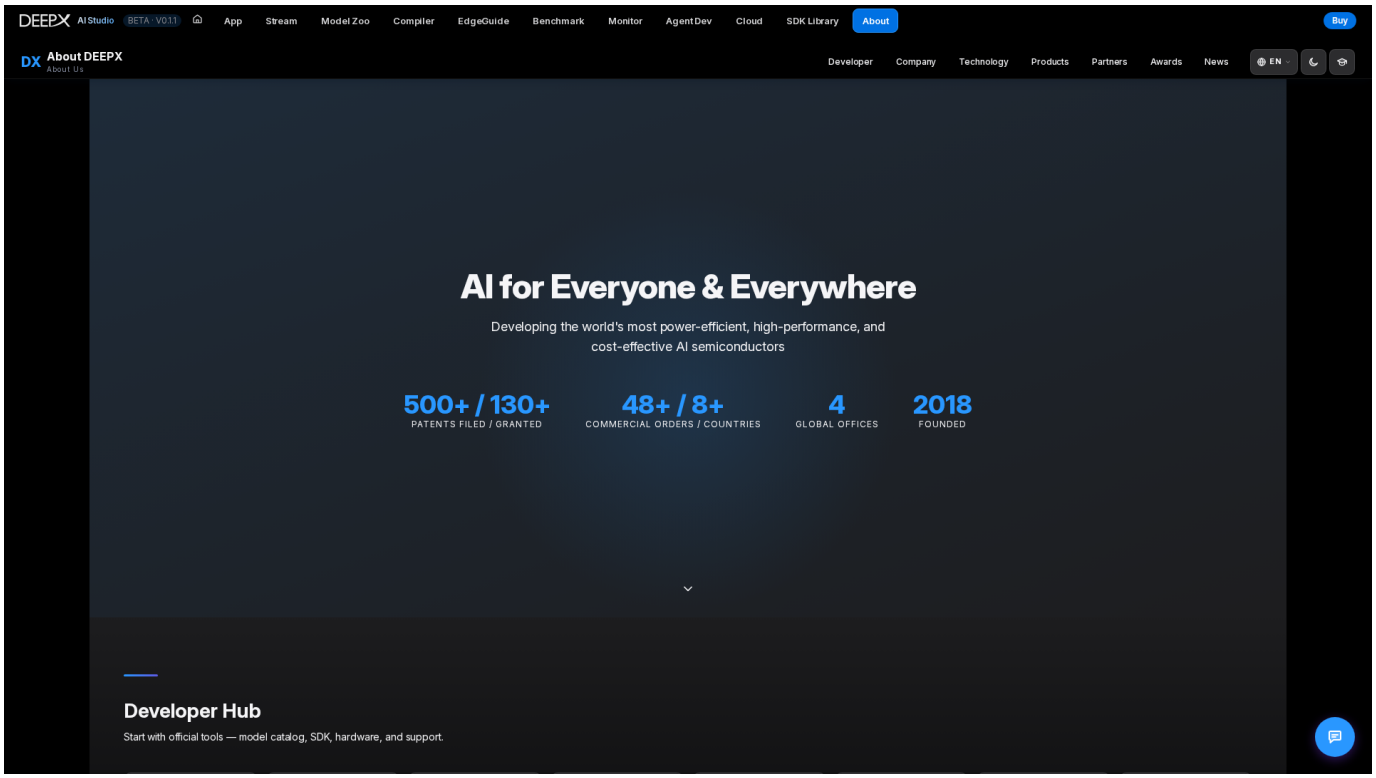


9.12.2 About DEEPX

The **About DEEPX** page introduces the company behind the studio — the "AI for Everyone & Everywhere" mission, at-a-glance figures (patents, commercial orders, global offices, year founded), and a **Developer Hub** with quick links to the model catalog, SDK, hardware, and support.

The **About DEEPX** page presents company and product information down one scrolling page with a sticky section nav (scroll-spy): a **Developer** section (dev-resource cards, CTAs, and distribution channels), a **Company** section (vision, milestones timeline, global offices), a **Technology** section (the DXNN SDK full-stack architecture), and **Products**, **Partners**, **Awards**, and **News** sections — plus a shortcut to the DEEPX store. Available in all six UI languages.

ⓒ Copyright 2026 DEEPX All Rights Reserved.



9.12.3 Chat / AI Assistant

The **chat assistant** (bottom-right button) is available in every DX AI Studio module (not just this one). Click it anytime to ask about DEEPX models, the SDK, the compiler, or how any module works — it draws on bundled DEEPX SDK knowledge to answer.

The line at the top of the chat always says who answers. With nothing set, it answers through the **coding agent you are signed in to** (`claude` , `copilot` , `cursor-agent` , `codex` or `opencode`) — no API key needed, and the same Studio agent that Home and Agent Dev use. Note that a coding agent answers from its vendor's cloud with that account's usage.

Change (or ⚙) asks one question — *Which AI should answer?*

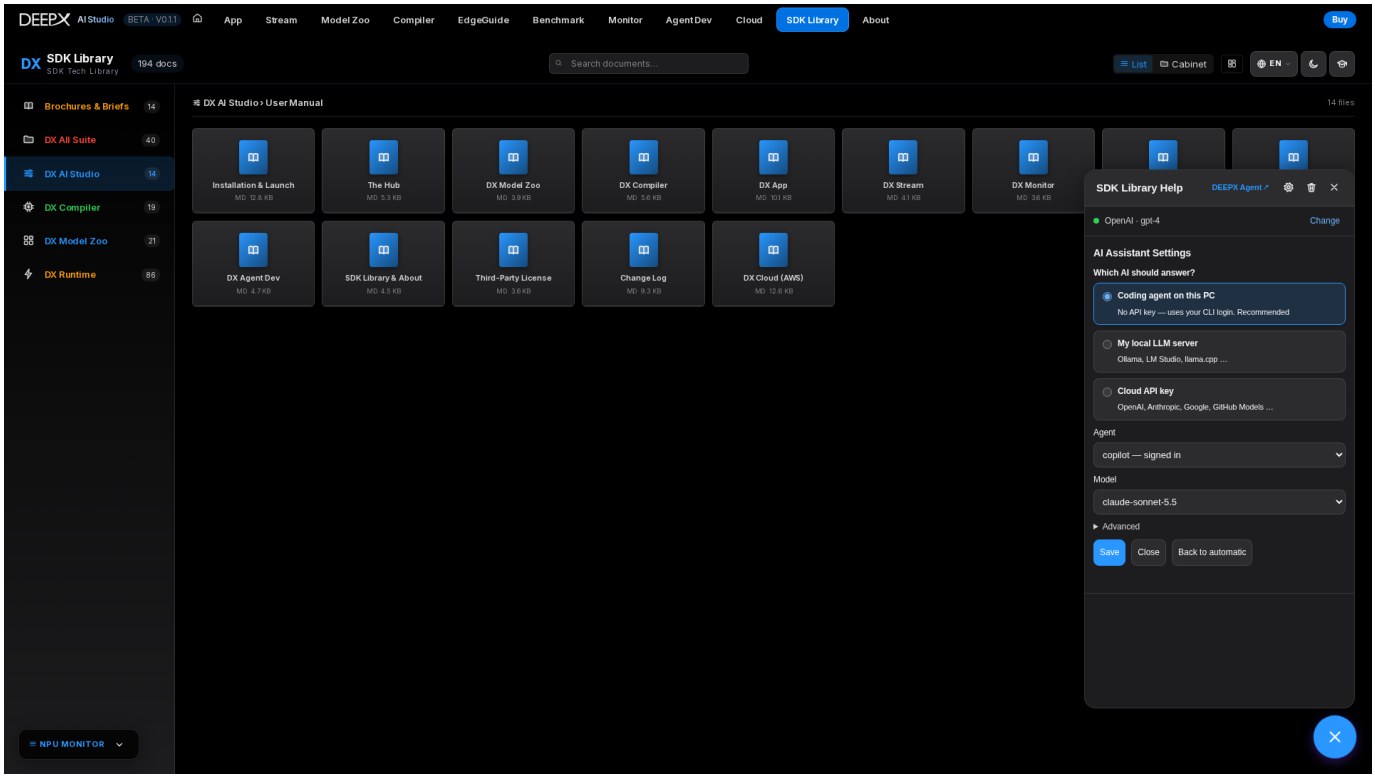
- **Coding agent on this PC** — pick an installed CLI; each shows whether it is signed in, or the command that signs it in.
- **My local LLM server** — your own Ollama-compatible server (Ollama, LM Studio, llama.cpp …).
- **Cloud API key** — OpenAI, Anthropic, Google, GitHub Models, or a custom OpenAI-compatible endpoint.

Saving tests the connection first; if it fails, the reason is shown with **Save anyway**. Leaving the key field blank keeps the saved key. **Back to automatic** forgets the choice. Temperature and **Refresh knowledge** (re-sync the assistant's SDK knowledge with the

ⓒ Copyright 2026 DEEPX All Rights Reserved.

latest `.deepx` docs) are under **Advanced**. With no agent signed in and nothing set, the assistant gives basic built-in guidance and says how to connect an AI.

For example, choose **My local LLM server**, enter your reachable endpoint, select one of its discovered models and save. If the test fails, check the URL / service and retry; **Save anyway** stores the selection despite that failure, not proof of a working AI connection. Use **Back to automatic** to return to the signed-in Studio agent.



The chat walkthrough (who answers, the one settings question, Refresh knowledge) is in the launcher's onboarding tour, rather than repeated per module.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

9.13 Third-Party License Notice

DX-AI-Studio is a GUI over the DEEPX SDK. The models and sample data it lets you browse, download, compile, and run come from third parties and carry **their own licenses** — the studio does not grant you any rights to them.

9.13.1 Sample models

Models surfaced in **DX Model Zoo** and used by **DX App** / **DX Compiler** (as `.dxnn` or ONNX) are supplied as evaluation / development examples. Availability in Studio does **not** itself grant commercial-deployment rights. They originate from different projects with different terms (including copyleft and non-commercial licenses); restrictions and attribution obligations must be checked for the selected model. Per-model license terms are shown in **DX Model Zoo's detail view** — check them before any commercial use. For production, obtain a proper license from the model's authors, or bring your own commercially licensed / proprietary model.

9.13.2 Sample data

Sample images and videos are from public academic / open-source datasets (e.g. ImageNet — non-commercial research only) or AI-generated. They are for **evaluation and demonstration only**. You are responsible for identifying each sample's source and complying with its terms, and must not redistribute or use them commercially without the copyright holder's permission.

9.13.3 Bundled software & fonts

The local UI includes fonts and JavaScript assets for offline rendering. External AI providers, downloads and AWS operations still need their own network access and terms.

Fonts with accompanying SIL Open Font License 1.1 notices — paths below are relative to `shared/static/fonts/` :

Font	Shipped notice	Copyright named in the notice
Inter	`LICENSE-Inter.txt`	The Inter Project Authors
Noto Sans SC / TC	`LICENSE-NotoSansSC.txt` , `LICENSE-NotoSansTC.txt`	Adobe; reserved font name “Source”

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Font	Shipped notice	Copyright named in the notice
Pretendard / Pretendard JP	`LICENSE-Pretendard.txt`, `LICENSE-PretendardJP.txt`	Kil Hyung-jin

CJK UI fonts are supplied as subsets. Other bundled font families include JetBrains Mono, Noto Sans and Noto Sans Mono. Before redistribution, retain the applicable font notices supplied with your distribution and check the upstream terms for each family.

JavaScript assets

Asset	Component / license information
`shared/static/vendor/chart.umd.min.js`	The header identifies **MiniChart**, a lightweight Chart.js-compatible canvas engine, not an upstream Chart.js bundle.
`shared/static/vendor/mermaid.min.js`	Mermaid package metadata declares MIT; the bundle also contains dependency-specific notices, including DOMPurify's Apache-2.0 / MPL-2.0 notice. Preserve those notices.
`dx_compiler/static/js/dagre.min.js`	The graph-layout bundle contains Chris Pettitt's copyright and MIT permission text, and bundled graphlib / lodash code.

MiniChart's Chart.js-compatible API does not mean that upstream Chart.js is bundled or that its license applies to MiniChart. For redistribution, consult your distribution's notices and obtain any additional component terms from its distributor.

For example, before redistributing a model demo, check its Model Zoo **license / source** fields and upstream terms, then preserve the relevant software / font notices with your package. A model license is not a substitute for the bundled UI assets' licenses.

9.13.4 Disclaimer

DEEPX provides these third-party models and datasets "as-is" for convenience and is not responsible for issues arising from their unauthorized or commercial use. By using these samples you assume all liability for license compliance.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

10. Kubernetes Quickstart

> New to Kubernetes? Start with the Kubernetes Beginner Guide, which explains each piece with a classroom analogy and hands-on examples.

Run DEEPX DX-M1 NPUs as native Kubernetes resources: request `deepx.ai/dx-m1: 1` in a Pod spec and the scheduler places the Pod on an NPU node, with the device node and runtime libraries injected automatically.

10.1 Architecture

```
host (per NPU node)          k3s cluster
─────────────────────────    ──────────────────────────────────────────────

dx-runtime/install.sh   ──►  driver (dxrt-driver-dkms) + firmware + dxrtd
containerd CDI enabled       (enable_cdi=true, cdi_spec_dirs incl /etc/cdi)

Helm: dx-npu ──► NodeFeatureRule (PCI 1ff4 → deepx.ai/dx-m1.present)
              └► device-plugin DaemonSet → advertises deepx.ai/dx-m1
                 ├─ writes /etc/cdi/deepx.json  (device node + libs + dxcli)
                 ├─ writes NFD feature file → fw/driver version node labels
                 └─ serves deepx_npu_* Prometheus metrics on :9400
Pod requests deepx.ai/dx-m1: 1 → containerd injects /dev/dxrtN + runtime libs
```

Components (mirroring the proven NVIDIA stack layout):

Component	Role
dx-k8s-device-plugin	Advertises `deepx.ai/dx-m1`, health checks, Pod allocation
CDI spec (generated by the plugin)	Injects `/dev/dxrtN`, runtime libs, and `dxcli` into containers
NFD integration	Node labels: presence, count, product, firmware/driver versions
`dx-npu` Helm chart	One-command install/upgrade of all of the above

© Copyright 2026 DEEPX All Rights Reserved.

10.2 Prerequisites

1. **Cluster tools.** Skip this step if you already have a Kubernetes cluster plus `kubectl` and `helm`.

```
# k3s — a single-binary Kubernetes; installs kubectl too
curl -sfL https://get.k3s.io | sh -

# use kubectl without sudo
mkdir -p ~/.kube
sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
sudo chown "$USER" ~/.kube/config
# k3s's kubectl reads /etc/rancher/k3s/k3s.yaml (root-only) unless KUBECONFIG is set
echo 'export KUBECONFIG=~/.kube/config' >> ~/.bashrc
export KUBECONFIG=~/.kube/config

# helm
curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

# jq — only used by the check commands in these docs
sudo apt-get install -y jq

kubectl get nodes      # STATUS Ready
helm version
```

   On vanilla Kubernetes, install kubectl from the official guide and helm from helm.sh.

2. **Each NPU node** — host driver, firmware, and runtime:

```
cd dx-runtime && ./install.sh --runtime-only
dxcli -s   # must list the NPU(s)
# NPU pods run their own dxrtd; a host dxrtd on the same card makes them hang
sudo systemctl disable --now dxrt.service
```

   The device plugin allocates each card to one Pod, and that Pod runs its own `dxrtd`. Keep the host `dxrt.service` stopped on NPU nodes — two `dxrtd` instances on one card leave in-Pod inference hanging. This makes the node Kubernetes-only for the NPU: host-side apps that need `dxrtd` will not run while the service is off.

   The CDI spec mounts the host `libdxrt.so.3` and `libonnxruntime.so.1` into every NPU Pod, so the node's DX-RT must be the same build as the DX-RT in the Pod image (`dxcli --version` on the host and in the image). A mismatch fails with `undefined symbol` errors.

3. **containerd CDI enabled.** containerd 2.x turns CDI on by default (`enable_cdi = true`, spec dirs `/etc/cdi` and `/var/run/cdi`), so there is nothing to do. Check the version:

```
sudo k3s ctr version | grep -m1 Version    # v2.x → done
```


© Copyright 2026 DEEPX All Rights Reserved.

Only containerd 1.7 needs CDI switched on. On k3s, **do not** create `config.toml.tmpl` with just the CDI lines — k3s uses the template *instead of* its generated config, so the rest of the containerd settings would disappear. Start from the generated file and append:

```
cd /var/lib/rancher/k3s/agent/etc/containerd
sudo cp config.toml config.toml.tmpl
sudo tee -a config.toml.tmpl <<'EOF'
[plugins."io.containerd.grpc.v1.cri".cdi]
  enable_cdi = true
  cdi_spec_dirs = ["/etc/cdi", "/var/run/cdi"]
EOF
sudo systemctl restart k3s
```

Either way, the smoke test below proves it: if `/dev/dxrt0` shows up inside the pod, CDI works.

10.3 Install

```
helm install dx-npu oci://ghcr.io/deepx-ai/charts/dx-npu \
  -n dx-system --create-namespace --set nfd.enabled=true
```

Leave `nfd.enabled=false` (default) if the cluster already runs node-feature-discovery.

Verify the NPU is schedulable and labeled:

```
kubectl get node -o json | jq '.items[].status.allocatable' | grep deepx.ai/dx-m1
kubectl get node --show-labels | grep -o 'deepx.ai[^,]*'
# deepx.ai/dx-m1.present=true, .count, .product, .fw-version, .driver-version
```

10.4 Run a workload

Smoke test (thin image — everything arrives via CDI):

```
kubectl apply -f deploy/k8s/samples/test-pod.yaml
kubectl logs -f dx-m1-test
# shows /dev/dxrt0 and full `dxcli -s` device status from inside the Pod
```

Real inference with the suite runtime image. An init container fetches one model into an emptyDir, then `dxrun` reports NPU time, latency, and FPS:

```
kubectl apply -f deploy/k8s/samples/inference-pod.yaml
kubectl wait --for=jsonpath='{.status.phase}'=Succeeded pod/dx-m1-inference --timeout=300s
kubectl logs dx-m1-inference -c inference
```


© Copyright 2026 DEEPX All Rights Reserved.

Any Pod becomes an NPU Pod with just a resource request:

```
resources:
  limits:
    deepx.ai/dx-m1: 1
```

10.5 Metrics

The device plugin serves Prometheus metrics on `:9400` (headless Service `dx-npu-metrics`): `deepx_npu_up`, `deepx_npu_device_healthy`, and per-core `deepx_npu_core_{temperature_celsius,voltage_millivolts,clock_mhz}`. Set `metrics.serviceMonitor.enabled=true` when prometheus-operator is present.

10.6 Troubleshooting

Symptom	**Cause / fix**
`deepx.ai/dx-m1` missing from allocatable	Device plugin Pod not running — check `kubectl get pods -n dx-system` and that the node carries `deepx.ai/dx-m1.present=true`
No `deepx.ai/*` node labels	NFD not installed — set `nfd.enabled=true` or install NFD; without NFD, label manually: `kubectl label node <n> deepx.ai/dx-m1.present=true`
Pod scheduled but `/dev/dxrt*` missing	containerd CDI not enabled — apply the config.toml.tmpl change and restart k3s
`dxcli` reports no devices on host	Driver/firmware not installed — rerun `dx-runtime/install.sh`, then cold-boot if device init fails
Device reported unhealthy	Card present in sysfs but absent from `dxcli -s` — usually needs a host power cycle
Inference Pod stays `Running`, `dxrun` never exits	Host `dxrt.service` is running on the node — `sudo systemctl disable --now dxrt.service` (Prerequisites, step 1)
`dxrun: symbol lookup error: undefined symbol` in a Pod	

© Copyright 2026 DEEPX All Rights Reserved.

Symptom	Cause / fix
	Node DX-RT and Pod image DX-RT differ — CDI mounts the host `libdxrt.so.3`; install the same DX-RT build on the node and in the image
`GLIBC_2.xx not found` running `dxcli` in a pod	The Pod image's glibc is older than the host DX-RT that CDI injects — use `ubuntu:24.04` or newer as the base (see the note below)

glibc required by the host DX-RT

CDI mounts the node's `dxcli` and `libdxrt.so.3` into the Pod, so the Pod image needs at least the glibc those binaries were built against:

- Prebuilt `libdxrt-bin` package: glibc ≥ 2.27
- DX-RT built from source on the node: follows the node's glibc (e.g. `GLIBC_2.38` on Ubuntu 24.04)

Check the actual requirement on the node with `objdump -T /usr/local/bin/dxcli /usr/local/lib/libdxrt.so.3 | grep -o 'GLIBC_[0-9.]*' | sort -uV | tail -1`.

Full chart values and details: `deploy/k8s/README.md`.

© Copyright 2026 DEEPX All Rights Reserved.

11. Kubernetes Beginner Guide — Learning with the DX-M1 NPU

This
guide is for people who have never used Kubernetes. Each idea is first explained with a **kindergarten classroom**, then checked with real commands.

If you only need installation steps, see the Kubernetes Quickstart.

11.1 Part 1. The picture

11.1.1 Imagine a kindergarten

```
Kindergarten (Cluster)
 │
 ├─ Head teacher (Scheduler)       "Which classroom should this child go to?"
 │
 ├─ Classroom A (Node)             door sign: 1 special block
 │    ├─ Class monitor (Device Plugin)
 │    ├─ Special block (DX-M1)
 │    └─ Child (Pod)
 │
 └─ Classroom B (Node)             door sign: no special block
      └─ Child (Pod)
```

In the kindergarten	In Kubernetes	What it does
The whole kindergarten	**Cluster**	Several computers used as one
A classroom	**Node**	One computer
A child	**Pod**	One running program (a box around containers)
The head teacher	**Scheduler**	Decides which classroom each child goes to
A special block	**DX-M1 NPU**	A card that runs AI inference fast

© Copyright 2026 DEEPX All Rights Reserved.

11.1.2 A child raises a hand: "Can I have a special block?"

To play with a special block, a child must **raise a hand**. In the Pod's YAML that looks like:

```yaml
resources:
  limits:
    deepx.ai/dx-m1: 1     # "One special block, please!"
```

The head teacher only sends that child to a classroom **that still has a free block**. Never to classroom B, which has none.

11.1.3 How does the head teacher know where the blocks are?

On their own, they don't. A few helpers make it work.

Class monitor = Device Plugin (`dx-k8s-device-plugin`)

Every classroom with a block has one class monitor, who keeps reporting to the head teacher: "We have 1 special block, and it works!"

If the block breaks, the monitor reports it right away and the head teacher stops sending children there. The rule that puts exactly one monitor in every block classroom is called a **DaemonSet**.

Door sign = NFD (Node Feature Discovery) label

A sign on the classroom door: "Special block · 1 · version v2.7.6". A child (or a parent) can ask to go only to classrooms whose sign says v2.7.6. That is a **nodeSelector**.

Delivery helper = CDI (Container Device Interface)

When a child enters the classroom, the helper puts the block and its **instruction booklet** (driver libraries) right on the child's desk. The child doesn't pack anything. That is why a container image can use the NPU without the DEEPX runtime baked in.

Classroom setup box = Helm chart (`dx-npu`)

Installing the monitors, door-sign rules, and health log one by one is tedious, so they all come in one box. Open it once (`helm install`) and every classroom is ready.

School nurse = Prometheus, **Notice board = Grafana**

Every 30 seconds the nurse records each block's temperature, pulse (clock), and health. The notice board draws those records as graphs. If a block was sick overnight, one look at the board in the morning tells you.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

11.1.4 One page summary

```
Child (Pod) raises a hand     "deepx.ai/dx-m1: 1, please"
      │
      ▼
Head teacher (Scheduler)      reads the monitors' reports, picks a classroom with a free block
      │                       (can also pick by door sign / version)
      ▼
Delivery helper (CDI)         puts /dev/dxrt0 + runtime libraries + dxcli on the desk
      │
      ▼
Child plays with the block    AI inference runs
      │
      ▼
Nurse (Prometheus)            records temperature and health → graphs on the board (Grafana)
```

11.1.5 Three rules to remember

1. **No raised hand, no block.** A Pod that does not request `deepx.ai/dx-m1` gets no NPU.
2. **One block, one child.** A card belongs to one Pod at a time. Others wait in line (`Pending`).
3. **No monitor, no block.** If the device plugin stops, the block disappears from the head teacher's view.

11.2 Part 2. Try it yourself

These examples were run on a k3s node with a DX-M1 installed. They assume the `dx-npu` chart is already installed. If not, follow the Quickstart from **step 0 (installing k3s, kubectl, helm)**.

> If output looks summarized, prefix `kubectl logs` with `rtk proxy` to see the raw text (only relevant where rtk is installed).

11.2.1 Example 1. Does our classroom have a block?

```
kubectl get node -o json | jq '.items[].status.allocatable | with_entries(select(.key|test("deepx")))'
```

```
{ "deepx.ai/dx-m1": "1" }
```

The head teacher knows this classroom has one special block.

And the door sign:

© Copyright 2026 DEEPX All Rights Reserved.

```
kubectl get node -o json | jq '.items[].metadata.labels | with_entries(select(.key|test("deepx")))'
```

```
{
  "deepx.ai/dx-m1.count": "1",
  "deepx.ai/dx-m1.driver-version": "v2.7.0",
  "deepx.ai/dx-m1.fw-version": "v2.7.6",
  "deepx.ai/dx-m1.pcie-driver-version": "v2.7.0",
  "deepx.ai/dx-m1.present": "true",
  "deepx.ai/dx-m1.product": "M1"
}
```

11.2.2 Example 2. Raise a hand, get a block

```
kubectl apply -f deploy/k8s/samples/test-pod.yaml
kubectl wait --for=jsonpath='{.status.phase}'=Running pod/dx-m1-test --timeout=120s
kubectl logs dx-m1-test
```

```
=== /dev/dxrt* ===
crw-rw-rw- 1 root root 507, 0 ... /dev/dxrt0
=== dxcli -s ===
DXRT v3.5.0
 * Device 0: M1, Accelerator type
 ...
NPU 0: voltage 750 mV, clock 1000 MHz, temperature 43'C
```

This Pod is plain `ubuntu:24.04` with no DEEPX software, yet the block (`/dev/dxrt0`) and `dxcli` are inside. The **delivery helper (CDI)** brought them.

```
kubectl delete pod dx-m1-test
```

11.2.3 Example 3. What if you don't raise a hand?

Same Pod, minus the `deepx.ai/dx-m1: 1` line:

```
sed '/deepx.ai\/dx-m1: 1/d; /resources:/d; /limits:/d' deploy/k8s/samples/test-pod.yaml \
  | sed 's/name: dx-m1-test/name: dx-m1-norequest/' | kubectl apply -f -
kubectl wait --for=jsonpath='{.status.phase}'=Running pod/dx-m1-norequest --timeout=120s
kubectl logs dx-m1-norequest
```

```
=== /dev/dxrt* ===
no NPU injected
```

That's **rule 1**: no raised hand, no block.

© Copyright 2026 DEEPX All Rights Reserved.

```
kubectl delete pod dx-m1-norequest
```

11.2.4 Example 4. Asking for more blocks than exist

The classroom has one block; ask for two.

```
kubectl run greedy --image=ubuntu:24.04 --restart=Never \
  --overrides='{"spec":{"containers":[{"name":"greedy","image":"ubuntu:24.04","command":
["sleep","60"],"resources":{"limits":{"deepx.ai/dx-m1":"2"}}}]}}'
kubectl get pod greedy
kubectl describe pod greedy | tail -3
```

```
NAME     READY   STATUS    RESTARTS   AGE
greedy   0/1     Pending   0          8s

0/1 nodes are available: 1 Insufficient deepx.ai/dx-m1.
```

`Pending` means waiting in line; the head teacher says there aren't enough blocks (`Insufficient`). That's **rule 2**: NPUs are counted and handed out like CPU and memory.

```
kubectl delete pod greedy
```

11.2.5 Example 5. Picking classrooms by version

Use the door sign to choose. Run only on nodes with firmware `v2.7.6`:

```yaml
# fw-pick.yaml
apiVersion: v1
kind: Pod
metadata:
  name: fw-pick
spec:
  restartPolicy: Never
  nodeSelector:
    deepx.ai/dx-m1.fw-version: "v2.7.6"   # only classrooms with this sign
  containers:
    - name: c
      image: ubuntu:24.04
      command: ["sleep", "30"]
      resources:
        limits:
          deepx.ai/dx-m1: 1
```


© Copyright 2026 DEEPX All Rights Reserved.

```
kubectl apply -f fw-pick.yaml
kubectl get pod fw-pick          # → ContainerCreating / Running
```

Change the version to something that doesn't exist, like `"v9.9.9"`:

```
0/1 nodes are available: 1 node(s) didn't match Pod's node affinity/selector.
```

No classroom has that sign, so the Pod stays `Pending`. With many nodes on mixed firmware, this is how you place **a model validated only on a specific version** safely.

```
kubectl delete pod fw-pick
```

11.2.6 Example 6. Actually playing with the block (AI inference)

```
kubectl apply -f deploy/k8s/samples/inference-pod.yaml
kubectl wait --for=jsonpath='{.status.phase}'=Succeeded pod/dx-m1-inference --timeout=300s
kubectl logs dx-m1-inference -c inference | grep -E "NPU Processing|Latency|FPS" | tail -3
```

```
  - NPU Processing Time  : 18.107 ms
  - Latency              : 26.783 ms
  - FPS                  : 34.04
```

This Pod works in two steps:

1. A **prep helper (initContainer)** downloads the model file (YoloV7) first.
2. The child (main container) starts `dxrtd` and runs inference with `dxrun`, which reports the speed.

> Why start `dxrtd` inside the Pod? The DEEPX runtime talks to its helper program (`dxrtd`) over a socket that doesn't leave the container, so each Pod runs its own. The card belongs to that Pod alone, so they never collide.
>
> If the image isn't in a registry yet, build and import it locally first. The steps are in the header comment of `deploy/k8s/samples/inference-pod.yaml`.

```
kubectl delete pod dx-m1-inference
```

11.2.7 Example 7. When the class monitor steps out

```
kubectl delete pod -n dx-system -l app.kubernetes.io/component=device-plugin
kubectl get node -o json | jq '.items[].status.allocatable["deepx.ai/dx-m1"]'
```


© Copyright 2026 DEEPX All Rights Reserved.

It briefly reads `"0"`: with no monitor, the block vanishes from the head teacher's view (**rule 3**). Within seconds the DaemonSet brings a new monitor and it's back to `"1"`.

Note the door signs (labels) stay put. NFD owns the signs and the monitor owns the block count, so they move independently.

11.2.8 Example 8. Reading the nurse's log

With Prometheus installed:

```
kubectl port-forward -n monitoring svc/kps-kube-prometheus-stack-prometheus 9090:9090
```

Open `http://localhost:9090`, type into the search box, press **Execute**:

Question	Type this
Is the block healthy? (1 = healthy)	`deepx_npu_device_healthy`
Temperature per core?	`deepx_npu_core_temperature_celsius`
Blocks available to hand out now?	`kube_node_status_allocatable{resource="deepx_ai_dx_m1"}`
Hottest core?	`max(deepx_npu_core_temperature_celsius)`

The **Graph** tab shows how it changed over time. A Grafana dashboard lives at `deploy/k8s/dashboards/dx-npu.json`; paste it into Grafana under **Dashboards** → **New** → **Import**.

11.3 Part 3. Common snags

If this happens	In kindergarten terms	Check
Pod stuck in `Pending`	All blocks taken, or no classroom has the sign you asked for	Events at the bottom of `kubectl describe pod <name>`
allocatable is `0`	No monitor, or the block was reported sick	`kubectl get pods -n dx-system`, `deepx_npu_device_healthy`
No `/dev/dxrt0` inside the Pod	The delivery helper (CDI) didn't run	containerd CDI setup (Quickstart step 2)

© Copyright 2026 DEEPX All Rights Reserved.

If this happens	In kindergarten terms	Check
`ImagePullBackOff`	The child can't find their bag (image)	Image name, whether it's published, whether it was imported locally
`dxrt service is not running`	The block helper (`dxrtd`) wasn't started	Run `dxrtd &` first in the Pod command
`GLIBC_2.xx not found`	The desk is too old to read the booklet	Use `ubuntu:24.04` or newer as the Pod image (Quickstart has the details)

11.4 Glossary

Term	One line
Cluster	Several computers used as one
Node	One computer in it
Pod	The unit that runs; a box around one or more containers
Container	A portable bag holding a program and its files
Scheduler	Decides which Node a Pod goes to
DaemonSet	Rule for running "one on every matching Node"
Device Plugin	Program that tells Kubernetes about special hardware (the NPU)
Extended Resource	A resource beyond CPU and memory; here `deepx.ai/dx-m1`
Label / nodeSelector	A Node's door sign / choosing Nodes by that sign
NFD	Inspects Node hardware and puts up door signs automatically
CDI	Standard rule for putting devices and files into containers

© Copyright 2026 DEEPX All Rights Reserved.

Term	One line
initContainer	A prep container that runs before the main one
Helm / Chart	Kubernetes installer / one install box
Prometheus / Grafana	Number logbook / the board that draws it
Pending	Waiting because there is nowhere to place it

ⓒ Copyright 2026 DEEPX All Rights Reserved.