

DXNN DX-ModelZoo User Manual

2026/10/23 (v1.0.0)

DEEPX.ai

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Table of contents

1. Quick Start 4
1.1 Quick Start Overview 4
1.2 Basic Workflow 4
1.3 Advanced Options (choose as needed) 6
1.4 Environment Variables 7
2. DX-ModelZoo 8
2.1 Key Features 8
2.2 Prerequisites 8
2.3 Get Started 8
2.4 Supported Tasks 9
2.5 Documentation Guide 10
3. Installation 11
3.1 Prerequisites 11
3.2 Installation Overview 12
3.3 Required Installation 12
3.4 Optional Installation 13
3.5 Core Dependencies 14
3.6 Extras Summary 15
3.7 Alternative Package Manager 15
4. Workflows 16
4.1 Model Evaluation 16
4.2 YAML Model Configuration 21
4.3 Datasets 32
4.4 Custom Models 42
4.5 Custom Datasets 51
5. Architecture 59
5.1 Architecture Overview 59
5.2 Pipeline 64
5.3 Registry Pattern 70
6. API Reference 77
6.1 CLI Reference 77
6.2 ModelBuilder 84
6.3 Session 91
6.4 DataLoader 100
6.5 Datasets 107

ⓒ Copyright 2026 DEEPX All Rights Reserved.

6.6 Evaluator 136
6.7 Preprocessing 145
6.8 Postprocessing 154

ⓒ Copyright 2026 DEEPX All Rights Reserved.

1. Quick Start

Quick walkthrough of the core DX-ModelZoo workflow: browse, evaluate, and compile models in minutes.

Prerequisites

Before running these commands, make sure you have installed DX-ModelZoo.

1.1 Quick Start Overview

Basic Workflow

- Step 1. Browse Available Models - Explore pre-configured models
- Step 2. View Model Information - Check model details and profiles
- Step 3. Evaluate with ONNX Runtime - Run CPU/GPU inference
- Step 4. Evaluate with NPU Runtime - Run NPU inference (optional)

Advanced Options (choose as needed)

- Optional 1. Compile a Model - Compile for NPU deployment
- Optional 2. Benchmark - Batch evaluation across multiple models
- Optional 3. Run with YAML File - Use custom configurations
- Optional 4. Create a Custom Model - Add your own models

1.2 Basic Workflow

1.2.1 Step 1: Browse Available Models

```
# Interactive TUI browser
dxmz list

# Text output (for CI/scripts)
dxmz list --all

# Filter by domain/task
dxmz list --domain cv --task image_classification
```


© Copyright 2026 DEEPX All Rights Reserved.

1.2.2 Step 2: View Model Information

```
dxmz info resnet50_224x224
```

Example output:

```
Name: resnet50_224x224
Inputs: [{'name': 'input', 'shape': [1, 3, 224, 224], 'dtype': 'float32', 'layout': 'NCHW'}]
Profiles: ['onnx', 'c-lite', 'q-lite', 'q-pro']
Dataset: {'type': 'ILSVRC2012', 'eval_path': '${DATA_ROOT}/ILSVRC2012/val'}
```

Available Profiles

- **onnx**: ONNX Runtime (CPU/GPU)
- **c-lite**: Compile profile with `opt_level: 0`
- **q-lite**: Compile and run on NPU with standard calibration
- **q-pro**: Compile and run on NPU with `use_q_pro: true`

1.2.3 Step 3: Evaluate with ONNX Runtime

```
dxmz eval resnet50_224x224 --profile onnx --save
```

See CLI Reference for all options (`--data-root`, `--model-path`, `--seed`, etc.).

With `--save`, a JSON file is saved to the `result/` directory:


```
{
  "model": "resnet50_224x224",
  "dataset": "imagenet",
  "metrics": [
    {"name": "Top-1", "metric_value": 69.76},
    {"name": "Top-5", "metric_value": 89.08}
  ],
  "fps": 624.7,
  "elapsed_time": 82,
  "profile": "onnx"
}
```


1.2.4 Step 4: Evaluate with NPU Runtime

© Copyright 2026 DEEPX All Rights Reserved.

```
# Evaluate a compiled dxnn model
dxmz eval resnet50_224x224 --profile q-lite
```

Compiled model availability

DX-ModelZoo does not compile a missing model during evaluation. For a `target: dxnn` profile, provide the `.dxnn` artifact in the configured artifact directory or set `DXMZ_MODEL_URL` so the matching artifact can be downloaded. Use `dxmz compile` to create the artifact explicitly.

NPU profiles are forced to batch_size=1

Profiles with `target: dxnn` always use `batch_size=1`, regardless of what is set in the YAML. Only ONNX profiles support adjustable `batch_size`.

1.3 Advanced Options (choose as needed)

1.3.1 Optional 1: Compile a Model

```
dxmz compile resnet50_224x224 --profile q-lite
```

See CLI Reference for all options (`--output`, `--profile q-pro`, etc.).

1.3.2 Optional 2: Benchmark

```
dxmz benchmark --profile onnx --save
```

See CLI Reference for domain/task filters and multi-device options (`--domain`, `--devices`, etc.).

Benchmark results are saved in JSON format:

```
result/benchmark_<profile>_<timestamp>.json
```


```
[
  {
    "model": "resnet50_224x224",
    "profile": "onnx",
    "metrics": [
      {"name": "Top-1", "metric_value": 69.76},
      {"name": "Top-5", "metric_value": 89.08}
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    ],
    "fps": 624.7,
    "elapsed_time": 82
  }
]
```

1.3.3 Optional 3: Run with a YAML File Directly

```
# Evaluate with a custom YAML
dxmz eval ./my_models/custom_model.yaml --profile onnx

# Compile with a custom YAML
dxmz compile ./my_models/custom_model.yaml --profile q-lite
```

1.3.4 Optional 4: Create a Custom Model

```
# Scaffold a new custom model (interactive wizard)
dxmz create
```

This creates a template in the `./custom/` directory with:

- YAML configuration file
- Custom preprocessing/postprocessing stubs
- Dataset loader template

1.4 Environment Variables

See CLI Reference for the full list of environment variables. The most common ones can be placed in a `.env` file for automatic loading:

.env

```
DATA_ROOT=/data/datasets
MODEL_ROOT=/data/models
DXMZ_MODEL_URL=https://sdk.deepx.ai/modelzoo
```


© Copyright 2026 DEEPX All Rights Reserved.

2. DX-ModelZoo

A unified framework for managing model preprocessing, postprocessing, evaluation, and NPU compilation — all driven by a single YAML configuration file.

2.1 Key Features

- **YAML-Centric Configuration**: Define entire model pipelines in a single YAML file — no Python code required. Preprocessing, postprocessing, and runtime profiles are all declared in one place.
- **Extensible Registry System**: Add custom preprocessing, postprocessing, datasets, and evaluators instantly using the `@REGISTRY.register()` decorator.
- **Auto-Discovery of Custom Ops**: Place a `custom_ops.py` in your model directory; it's automatically imported and registered.
- **Multi-Profile Execution**: Manage multiple runtime profiles — ONNX, NPU/dxnn, and more — within a single YAML configuration.
- **300+ Built-in Models**: Production-ready CV models spanning classification, detection, segmentation, and beyond.

2.2 Prerequisites

See Installation for the required Python version and DeepX Components.

2.3 Get Started

```
# Install DeepX Components
pip install dx_engine-*.whl dx_com-*.whl

# Clone and install DX-ModelZoo
git clone https://github.com/DEEPX-AI/dx-modelzoo.git
cd dx-modelzoo
pip install -e ".[cpu]"

# 1. Browse available models
dxmz list

# 2. View model information
dxmz info resnet50_224x224
```


© Copyright 2026 DEEPX All Rights Reserved.

```
# 3. Evaluate with ONNX Runtime
dxmz eval resnet50_224x224 --profile onnx

# 4. (Optional) Compile for NPU
dxmz compile resnet50_224x224 --profile q-lite
```

See Quick Start Guide for step-by-step instructions and Installation Guide for detailed setup.

2.4 Supported Tasks

Category	Task	Model Variants
Classification	Image Classification	137
	Zero-Shot Image Classification	6
	Image Retrieval	1
Object Detection	Object Detection	124
	Face Detection	18
	Oriented Object Detection	5
	Mainbody Detection	2
	Hand Detection	1
	3D Object Detection	1
Segmentation	Instance Segmentation	35
	Semantic Segmentation	24
	Zero-Shot Instance Segmentation	1
	Panoptic Driving Perception	1
Pose / Landmark	Pose Estimation	21
	Face Landmark	2
	Hand Landmark	1
	Keypoint Detection	1

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Category	Task	Model Variants
	Object Pose Estimation	1
Face / Person	Face Recognition	4
	Person Attribute	2
	Face Attribute	1
	Person Re-Identification	1
Image Restoration	Super Resolution	6
	Image Denoising	5
	Low-Light Enhancement	2
	Image Matting	2
Other	Depth Estimation	10
	Anomaly Detection	4
	Visual Place Recognition	3

> **422 model variants** across **29 CV tasks** — all configurable via YAML.
>
> Counts are YAML definitions (variants) under `src/dx_modelzoo/models/cv`, not distinct architectures. A single architecture may ship several variants (different input sizes, decode options, or PPU / pre-optimized builds).

2.5 Documentation Guide

Section	Description
Quick Start	Quick introduction and step-by-step usage
Installation	Detailed installation and environment setup
Guides	Model evaluation workflows and CLI usage
Architecture	Internal design, Registry pattern, pipeline flow
API Reference	CLI, ModelBuilder, Pre/Post-processing, Evaluator, etc.

© Copyright 2026 DEEPX All Rights Reserved.

3. Installation

Step-by-step instructions for installing DX-ModelZoo and its dependencies on Linux systems.

3.1 Prerequisites

Before installing DX-ModelZoo, ensure you have the following:

- **Python**: 3.8+
- **DeepX Components**:
- DeepX Runtime (DX-RT) ≥ 3.5.0 (compiled with `USE_ORT=ON`)
- DeepX Compiler (DX-Compiler) ≥ 2.5.0

Windows Users

DX-ModelZoo runs on Windows via WSL2 (Windows Subsystem for Linux).

To install WSL2:

```
# Run in PowerShell (admin mode)
wsl --install
```

Then reboot and launch Ubuntu from the Start menu. See Microsoft's WSL documentation for details.

GPU Requirements (Optional)

For GPU inference:

- NVIDIA GPU (Pascal/Turing/Ampere architecture)
- NVIDIA Driver ≥ 535.230.02
- CUDA 12.2
- cuDNN 9.1

© Copyright 2026 DEEPX All Rights Reserved.

3.2 Installation Overview

Required Steps

- Step 1. Install DeepX Components - DX-RT & DX-Compiler
- Step 2. Clone Repository - Get the source code
- Step 3. Basic Installation - CPU or GPU inference setup

Optional Steps (choose as needed)

- Optional 1. GPU Inference - For CUDA acceleration
- Optional 2. Development Environment - For contributors
- Optional 3. Documentation Build - To build docs locally
- Optional 4. Full Installation - All features at once

3.3 Required Installation

3.3.1 Step 1: Install DeepX Components

```
# Install DX-RT python package
pip install dx_engine-*.whl

# Install DX-Compiler python package
pip install dx_com-*.whl
```

NOTE

Refer to the official DX-RT and DX-Compiler installation guides for detailed setup instructions.

3.3.2 Step 2: Clone Repository

```
git clone https://github.com/DEEPX-AI/dx-modelzoo.git
cd dx-modelzoo
```

3.3.3 Step 3: Basic Installation — choose CPU **or** GPU

Pick exactly one ONNX Runtime backend. `onnxruntime` and `onnxruntime-gpu` must not be installed in the same environment (ONNX Runtime install guide).

© Copyright 2026 DEEPX All Rights Reserved.

```
# CPU inference
pip install -e ".[cpu]"
```

```
# GPU inference (CUDA) — see the CUDA note under Optional Installation
pip install -e ".[gpu]"
```

This installs the selected ONNX Runtime build and all core dependencies.

Switching between CPU and GPU

Remove the other distribution before switching, otherwise both remain installed and provider resolution becomes undefined:

```
# CPU -> GPU
pip uninstall -y onnxruntime && pip install -e ".[gpu]"

# GPU -> CPU
pip uninstall -y onnxruntime-gpu && pip install -e ".[cpu]"
```

Verify with `pip list | grep onnxruntime` — exactly one entry should remain.

3.4 Optional Installation

3.4.1 Optional 1: GPU Inference (CUDA) environment setup

CUDA Environment Setup

See GPU Requirements above for driver/CUDA/cuDNN versions.

After installing `.[gpu]`, add NVIDIA library paths to `LD_LIBRARY_PATH`:

```
# Add to ~/.zshrc or ~/.bashrc
export LD_LIBRARY_PATH=$(python -c "import nvidia.cublas.lib;
print(nvidia.cublas.lib.__path__[0])"):$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$(python -c "import nvidia.cudnn.lib;
print(nvidia.cudnn.lib.__path__[0])"):$LD_LIBRARY_PATH
```

3.4.2 Optional 2: Development Environment

```
pip install -e ".[dev]"  # Includes pytest, ruff
```


© Copyright 2026 DEEPX All Rights Reserved.

3.4.3 Optional 3: Documentation Build

Documentation build requires Python 3.9 or newer because the configured PDF plugin does not support Python 3.8.

```
pip install -e ".[docs]"
mkdocs serve -f docs/mkdocs.yml  # Preview at http://localhost:8000
```

Build a static site and PDF from repository root:

```
mkdocs build -f docs/mkdocs.yml
```

The `docs` extra includes MkDocs, Material, mkdocstrings, the include-markdown plugin, and the PDF plugin configured in `docs/mkdocs.yml`.

3.4.4 Optional 4: Full Installation (cpu + dev + docs)

```
pip install -e ".[all]"
```

This installs every optional extra at once.

3.5 Core Dependencies

DX-ModelZoo keeps its core dependencies minimal:

```
pyyaml>=5.0
numpy>=1.20,<3.0
Pillow>=8.0
opencv-python>=4.5
python-dotenv>=0.19
tqdm>=4.60
loguru>=0.6
typer>=0.9
rich>=12.0
textual>=0.40
h5py
scipy
faster-coco-eval
open_clip_torch
transformers
pyclipper>=1.3.0.post6
shapely>=2.0.7
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

PyTorch stack is part of the core runtime

`torch` is a core dependency in `pyproject.toml`, and the source tree also relies on `torchvision` APIs for preprocessing/data loading paths. Do not treat PyTorch as an optional compile-only add-on.

3.6 Extras Summary

Extra	Packages	Purpose
`cpu`	`onnxruntime>=1.15`	CPU-based ONNX inference
`gpu`	`onnxruntime-gpu>=1.17`	GPU-based ONNX inference
`dev`	`pytest>=7.0`, `ruff>=0.1`	Testing & linting
`docs`	`mkdocs`, `mkdocs-material`, `mkdocstrings`, `mkdocs-include-markdown-plugin`, `mkdocs-to-pdf`	Documentation build
`all`	`cpu` + `dev` + `docs`	Everything

3.7 Alternative Package Manager

3.7.1 Using uv

For faster dependency resolution, you can use `uv` instead of `pip`:

```
uv pip install -e ".[cpu]"
```

Dependency resolution is much faster with `uv`

```
pip install uv              # Install uv
uv pip install -e ".[gpu]"  # Fast installation
```


© Copyright 2026 DEEPX All Rights Reserved.

4. Workflows

4.1 Model Evaluation

This guide provides a comprehensive overview of model evaluation in DX-ModelZoo, covering evaluation workflows, options, and advanced usage patterns.

> **Prerequisites**
>
> Before evaluating models, make sure you have installed DX-ModelZoo and reviewed the Quick Start guide.

4.1.1 Running an Evaluation

Basic Usage

The minimal command requires only a model name and profile:

```
dxmz eval <model_name> --profile <profile> --seed 42
```

Evaluation Options

`--data-root` / `--model-root` override the `DATA_ROOT` / `MODEL_ROOT` environment variables, `--model-path` points directly at a model file, and `--seed` makes evaluation reproducible. See CLI Reference for the full option list.

`--save` writes `result/eval_<model>_<profile>.json`.

Multiple models can be evaluated in one command:

```
# Evaluate multiple models sequentially
dxmz eval resnet50_224x224 mobilenetv2_224x224 --profile onnx --seed 42
# Results are saved separately for each model when --save is used
```

Evaluation Flow

The following sequence diagram shows the internal call flow:

© Copyright 2026 DEEPX All Rights Reserved.

```
sequenceDiagram
    participant CLI as dxmz eval
    participant MB as ModelBuilder
    participant S as Session
    participant D as Dataset
    participant E as Evaluator

    CLI->>MB: resolve_model_target - resnet50_224x224
    MB->>MB: load_yaml + validate
    MB->>S: build_session - onnx
    MB->>D: build_dataset
    MB->>E: build_evaluator - session, dataset
    E->>E: set_preprocessing
    E->>E: set_postprocessing
    E->>E: eval
    loop Each batch
        E->>D: __getitem__ + preprocessing
        E->>S: session.run - input
        E->>E: postprocessing - output
        E->>E: process_batch_result
    end
    E-->>CLI: {"metrics": [...], "fps": 624.7}
```

Sync vs Async Evaluation

ⓒ Copyright 2026 DEEPX All Rights Reserved.

The evaluator automatically selects the mode based on the session's async support:

Sync Mode **Async Mode**

```
for batch in dataloader:
    output = session.run(input)       # Synchronous inference
    postprocessing(output)
    update_metrics()
```

- Default mode
- Used with ONNX Runtime

```
sliding window (max_inflight = min(device_count × 4, total_samples)):
    session.run_async(input)          # Async inference request
    output = session.wait(job_id)     # Wait for result
    postprocessing(output)
    update_metrics()
```

- Automatically enabled for DxRuntime (NPU)
- Loader prefetch depth is bounded by `workers × 4`
- In-flight inference requests are bounded by `device_count × 4`

Batch Size

ONNX profiles support adjustable `batch_size`:

resnet50_224x224.yaml

```yaml
profiles:
  onnx:
    target: onnx
    runtime:
      device: gpu
      batch_size: 4    # Batch size 4
```

> ⚠ **dxnn Profiles**
>
> Profiles with `target: dxnn` are always forced to `batch_size=1`. Any other value set in the YAML is ignored.

4.1.2 Evaluation Output

© Copyright 2026 DEEPX All Rights Reserved.

Result Format

Evaluation results are structured as follows:

```
{
  "model": "resnet50_224x224",
  "dataset": "imagenet",
  "metrics": [
    {"name": "Top-1", "metric_value": 69.76},
    {"name": "Top-5", "metric_value": 89.08}
  ],
  "fps": 624.7,
  "elapsed_time": 82,
  "start_time": "2026-07-23 14:30:00",
  "profile": "onnx"
}
```

FPS Calculation

FPS is reported differently for the sync and async evaluation paths, so the two numbers are **not** directly comparable.

Path	Formula	Included	Excluded
Sync (default)	`images / inference_time`	Inference only (`run_inference`, incl. model input extraction)	Data loading, pre/ postprocessing
Async (session `use_async`)	`images / wall_time`	Everything: loader wait, inference, postprocessing	—

The sync path measures the inference section per batch; the async path uses total wall time of the pipeline. Use the **Time Breakdown** below to see how much load / infer / post each contributes.

Time Breakdown

A time breakdown is automatically printed after evaluation:

ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
Time breakdown — load: 12.3s (15%) | infer: 65.2s (79%) | post: 4.8s (6%)
```

Phase	Description
`load`	Data loading + preprocessing time
`infer`	Model inference time
`post`	Postprocessing + metric update time

4.1.3 Built-in Evaluators

Common evaluators include the following (see the Evaluator API for the complete list of all built-in evaluators):

Evaluator	Task	Metrics
`image_classification`	Classification	Top-1, Top-5 Accuracy
`object_detection`	Detection (COCO/VOC)	mAP@0.5, mAP@0.5:0.95
`semantic_segmentation`	Segmentation	mIoU
`depth_estimation`	Depth	$\delta < 1.25$, RMSE
`pose_estimation`	Pose	AP, AR
`face_landmark`	Landmark	NME
`face_recognition`	Face verification	Accuracy
`super_resolution`	Super resolution	PSNR
`oriented_object_detection`	Oriented detection	mAP
`zero_shot_image_classification`	Zero-shot	Top-1 Accuracy
`image_retrieval`	Image Retrieval	Top-1, Top-5 Accuracy

© Copyright 2026 DEEPX All Rights Reserved.

4.2 YAML Model Configuration

Every model in DX-ModelZoo is defined by a single YAML file. No Python code is needed — preprocessing, postprocessing, evaluation, and compilation settings are all included.

4.2.1 Full Structure

```yaml
name: resnet50_224x224              # Model name
task: image_classification    # Task type
reference:                  # Paper/repository URL
description:                # Model description (multiline)

inputs:                   # Input tensor definitions
  - name: input
    shape: [1, 3, 224, 224]
    dtype: float32
    layout: NCHW

preprocessing:              # Preprocessing pipeline
  - type: resize
    size: [256, 256]
  - type: centercrop
    height: 224
    width: 224
  # ...

postprocessing:             # Postprocessing pipeline
  - type: topk
    k: [1, 5]

dataset:                  # Dataset configuration
  type: ILSVRC2012
  eval_path: ${DATA_ROOT}/ILSVRC2012/val

evaluator:                  # Evaluator type
  type: image_classification

artifacts:                # Model file paths
  path: ${MODEL_ROOT}/${MODEL_NAME}

calibration: &calibration
  method: ema
  num: 100
  seed: 42

profiles:                 # Runtime profiles
  onnx:
    target: onnx
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
    runtime:
      device: gpu
      batch_size: 1
  q-lite:
    target: dxnn
    compile:
      calibration: *calibration
    runtime:
      device: 0
```

4.2.2 Section Details

`name` / **`task`**

```
name: YOLOv8s           # Unique name (should match filename)
task: object_detection   # Task type
```

Supported tasks:

Task	Description
`image_classification`	Image classification
`object_detection`	Object detection
`semantic_segmentation`	Semantic segmentation
`instance_segmentation`	Instance segmentation
`face_detection`	Face detection
`face_landmark`	Facial landmark detection
`pose_estimation`	Pose estimation
`depth_estimation`	Depth estimation
`super_resolution`	Super resolution
`oriented_object_detection`	Oriented bounding box detection (OBB)

`reference` / **`description`**

Optional metadata fields for documentation:

© Copyright 2026 DEEPX All Rights Reserved.

```
reference: https://arxiv.org/abs/1512.03385  # Paper or GitHub URL
description: |
  ResNet-50 trained on ImageNet-1K.
  Top-1 accuracy: 76.13%
  Top-5 accuracy: 93.15%
```

- **reference**: URL to the paper, GitHub repo, or model card
- **description**: Multiline string describing the model, training details, and performance

inputs

Defines the model's input tensors. Multi-input models are supported.

```
# Single input (most CV models)
inputs:
  - name: input
    shape: [1, 3, 224, 224]
    dtype: float32
    layout: NCHW

# Multiple inputs (e.g., BERT NLP models)
inputs:
  - name: input_ids
    shape: [1, 256]
    dtype: int64
  - name: attention_mask
    shape: [1, 256]
    dtype: int64
  - name: token_type_ids
    shape: [1, 256]
    dtype: int64
```

preprocessing

The preprocessing pipeline applied to input data. Steps execute in order.

```
preprocessing:
  - type: resize
    mode: torchvision
    size: [256, 256]
    interpolation: BILINEAR
  - type: centercrop
    height: 224
    width: 224
  - type: convertcolor
    form: BGR2RGB
  - type: div
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    x: 255
  - type: normalize
    mean: [0.485, 0.456, 0.406]
    std: [0.229, 0.224, 0.225]
  - type: transpose
    axis: [2, 0, 1]
  - type: expanddim
    axis: 0
```

Built-in preprocessing types:

Type	Parameters	Description
`resize`	`size`, `mode`, `interpolation`	Image resize
`centercrop`	`height`, `width`	Center crop
`convertcolor`	`form`	Color conversion (e.g., BGR2RGB)
`div`	`x`	Divide values (normalization)
`normalize`	`mean`, `std`	Mean/std normalization
`transpose`	`axis`	Axis permutation (HWC→CHW)
`expanddim`	`axis`	Add dimension (batch)
`bgr_to_y_channel`	—	BGR → Y channel conversion

NPU arithmetic behavior

For a `target: dxnn` profile, arithmetic operations such as `div`, `normalize`, `transpose`, and `expanddim` are skipped when runtime input dtype is `uint8`. They remain for `float32` input. Set `appendix.npu_skip_arithmetic: false` when a model requires CPU-side arithmetic preprocessing.

postprocessing

The postprocessing pipeline applied to model outputs.

```
# Classification
postprocessing:
  - type: topk
    k: [1, 5]
```


© Copyright 2026 DEEPX All Rights Reserved.

```yaml
# Object Detection
postprocessing:
  - type: nms
    conf_thres: 0.25
    iou_thres: 0.45
    max_output_boxes: 300

# Segmentation
postprocessing:
  - type: segmentation_argmax

# No postprocessing
postprocessing:
  - type: identity
```

dataset

Dataset configuration for evaluation. Use `${DATA_ROOT}` to specify the root path.

```yaml
dataset:
  type: ILSVRC2012
  eval_path: ${DATA_ROOT}/ILSVRC2012/val
```

Built-in datasets:

Type	Task	Example Path
ILSVRC2012	Classification	ILSVRC2012/val
COCO	Detection	COCO
PascalVOC2007	Detection	VOCdevkit/VOC2007
Cityscapes	Segmentation	cityscapes
ADE20K	Segmentation	ADEChallengeData2016
WiderFace	Face detection	WIDER_FACE
COCOPose	Pose estimation	COCO
NYUDepthv2	Depth estimation	NYU_Depth_V2
BSD68	Denoising	BSD68
BSD100	Super resolution	BSD100/bicubic_4x

© Copyright 2026 DEEPX All Rights Reserved.

Type	Task	Example Path
DOTAv1	Oriented detection	DOTA/val
LFW	Face verification	lfw

evaluator

Specifies the evaluator type for computing metrics.

```yaml
evaluator:
  type: image_classification
```

The evaluator type typically matches the task type. Common evaluators:

Evaluator	Metrics
image_classification	Top-1, Top-5 Accuracy
object_detection	mAP@0.5, mAP@0.5:0.95
semantic_segmentation	mIoU, Pixel Accuracy
instance_segmentation	bbox mAP, segm mAP
face_detection	mAP, Recall
pose_estimation	AP, AR

artifacts

Defines where to find model files (ONNX, weights, etc.).

```yaml
artifacts:
  path: ${MODEL_ROOT}/${MODEL_NAME}
```

- **path**: Directory containing model files
- Uses `${MODEL_ROOT}` environment variable + `${MODEL_NAME}` (auto-set to the `name` field)
- DX-ModelZoo automatically downloads models if not found locally

profiles

© Copyright 2026 DEEPX All Rights Reserved.

Defines multiple runtime profiles. Each profile specifies its own target, compilation options, and runtime options.

```yaml
profiles:
  # ONNX Runtime profile
  onnx:
    target: onnx
    runtime:
      device: gpu          # gpu or cpu
      batch_size: 4        # Batch size (adjustable for onnx only)
      async: false         # Default for onnx profiles

  # NPU Compile-Lite (opt_level=0)
  c-lite:
    target: dxnn
    compile:
      opt_level: 0
      calibration: *calibration
    runtime:
      device: 0
      async: true          # Default for dxnn profiles
      buffer_count: 6      # Engine I/O buffer count (1-100)
      use_ort: true        # Run unsupported ops on ONNX Runtime

  # NPU Quantize-Lite
  q-lite:
    target: dxnn
    compile:
      calibration: *calibration
    runtime:
      device: 0

  # NPU Quantize-Pro (automatic Q-PRO pipeline)
  q-pro:
    target: dxnn
    compile:
      calibration:
        <<: *calibration
        num: 1024
        use_q_pro: true
    runtime:
      device: 0
      async: true
```

NOTE

`calibration.use_q_pro: true` enables dx_com's automatic Q-PRO pipeline. Calibration settings are passed through the profile's `compile` block.

© Copyright 2026 DEEPX All Rights Reserved.

PROFILE NAMING CONVENTION

Profile	Target	Description
`onnx`	onnx	ONNX Runtime inference
`c-lite`	dxnn	Compile profile with `opt_level: 0`
`q-lite`	dxnn	NPU profile with standard calibration
`q-pro`	dxnn	NPU profile with `calibration.use_q_pro: true`

CALIBRATION OPTIONS

These keys are typically defined once in a top-level `calibration:` block (using a YAML anchor, e.g. `calibration: &calibration`) and reused across profiles via `calibration: *calibration`, as shown in Full Structure.

Key	Description
`compile.calibration.method`	Calibration method, such as `ema`
`compile.calibration.num`	Number of calibration samples
`compile.calibration.seed`	Calibration random seed
`compile.calibration.use_q_pro`	Enable automatic Q-PRO calibration

RUNTIME OPTIONS

Key	Type	Description
`runtime.device`	`str | int | list`	Execution device. ONNX typically uses `cpu` / `gpu`; DXNN uses NPU device IDs.
`runtime.batch_size`	`int`	Evaluation batch size. Adjustable for ONNX; DXNN evaluation is effectively batch size 1.

© Copyright 2026 DEEPX All Rights Reserved.

Key	Type	Description
`runtime.async`	`bool`	Enable async inference. Defaults to `false` for ONNX profiles and `true` for DXNN profiles.
`runtime.buffer_count`	`int`	DXNN-only engine I/O buffer count. Must be in `[1, 100]`.
`runtime.use_ort`	`bool`	DXNN-only fallback for unsupported ops via ONNX Runtime. Defaults to `true`.

compile.ppu_config

`ppu_config` is an optional DXNN-only compile block for postprocessing. Types `0` and `1` use PPU hardware; type `2` is a CPU-side TopK optimization for DFL-based YOLO models and does not use PPU hardware.

```yaml
compile:
  ppu_config:
    type: 0|1|2        # 0=anchor-based, 1=anchor-free single-head, 2=DFL-based CPU TopK
(deprecated)
    num_classes: 80
    conf_thres: 0.5     # type 0,1 only
    activation: sigmoid # type 0 only
    topk: 100           # type 2 only
    layer: ...          # dict for type 0, list for type 1,2
```

Key	Required For	Description
`type`	all	Mode: `0` anchor-based PPU, `1` anchor-free single-head PPU, `2` DFL-based CPU TopK (no PPU hardware)
`num_classes`	all	Number of classes
`conf_thres`	type `0`, `1`	Confidence threshold
`activation`	type `0`	Activation name such as `sigmoid`

© Copyright 2026 DEEPX All Rights Reserved.

Key	Required For	Description
`topk`	type `2`	Top-k candidate count
`layer`	all	Per-layer PPU description (`dict` for type `0` , `list` for type `1` / `2`)

> ⚠ **PPU type 2 is deprecated**
>
> DX-Compiler deprecated type `2` in v2.4.0. Use `dx_com.pre_optimize()` for the recommended TopK-first optimization instead.

4.2.3 Environment Variable Substitution

YAML files can reference environment variables using the `${VAR_NAME}` syntax:

```yaml
dataset:
  eval_path: ${DATA_ROOT}/ILSVRC2012/val  # Substitutes DATA_ROOT env var

artifacts:
  path: ${MODEL_ROOT}/${MODEL_NAME}      # MODEL_NAME equals the name field
```

Special variables:

Variable	Description	Example
`${MODEL_NAME}`	Value of the YAML `name` field (not an OS environment variable)	`resnet50_224x224`

`${DATA_ROOT}` , `${MODEL_ROOT}` , and `${DXMZ_MODEL_URL}` follow the same `${VAR_NAME}` substitution syntax — see CLI Reference for the full list and how to set them.

`DXMZ_MODEL_URL` is used by the session factory to construct download URLs:

- ONNX: `${DXMZ_MODEL_URL}/onnx/<model_name>.onnx`
- DXNN: `${DXMZ_MODEL_URL}/<profile>-dxnn/latest/<model_name>_<profile>.dxnn`

© Copyright 2026 DEEPX All Rights Reserved.

These values can be set via:

- Environment variables: `export DATA_ROOT=/data/datasets`
- `.env` file in the project root
- Command-line options: `dxmz eval ... --data-root /data/datasets`

ⓒ Copyright 2026 DEEPX All Rights Reserved.

4.3 Datasets

Configuration reference and download instructions for commonly used evaluation datasets in DX-ModelZoo. For the complete list of all registered datasets, see the Datasets API.

Quick Setup

Create symlinks to your actual dataset locations:

```
export DATA_ROOT=/your/datasets/folder
# or symlink
ln -s /your/datasets/folder ./download/datasets
```

Dataset YAML paths containing `${DATA_ROOT}` resolve from `DATA_ROOT`. Use `--data-root <path>` only when overriding `DATA_ROOT` for one command.

4.3.1 Dataset Summary

Dataset	Task	Eval Path
ImageNet ILSVRC2012	Image Classification	`${DATA_ROOT}/ILSVRC2012/val`
DOTA v1	Oriented Object Detection	`${DATA_ROOT}/DOTAv1`
Pascal VOC 2007/2012	Object Detection / Semantic Segmentation	`${DATA_ROOT}/PascalVOC/VOCdevkit/VOC2012`
COCO 2017	Instance Segmentation / Pose Estimation	`${DATA_ROOT}/COCO`
Oxford-IIIT Pet	Semantic Segmentation	`${DATA_ROOT}/Oxford-IIIT_Pet`
Cityscapes	Semantic Segmentation	`${DATA_ROOT}/cityscapes`
AFLW2000-3D	3D Face Landmark	`${DATA_ROOT}/AFLW20003D`
Hand Keypoints	Hand Landmark	`${DATA_ROOT}/hand-keypoints`
CelebA	Face Attribute	`${DATA_ROOT}/CelebA`
LFW	Face Recognition	`${DATA_ROOT}/LFW`
WiderFace	Face Detection	`${DATA_ROOT}/widerface`

© Copyright 2026 DEEPX All Rights Reserved.

Dataset	Task	Eval Path
PETA	Pedestrian Attribute	${DATA_ROOT}/PETA
BSD100	Super Resolution	${DATA_ROOT}/BSD100/bicubic_4x
BSD68	Image Denoising	${DATA_ROOT}/BSD68
CBSD68	Image Denoising	${DATA_ROOT}/CBSD68
LOL	Low-Light Enhancement	${DATA_ROOT}/LOL
NYU Depth v2	Depth Estimation	${DATA_ROOT}/nyudepthv2
Market-1501	Person Re-ID	${DATA_ROOT}/Market1501

4.3.2 Classification

ImageNet ILSVRC2012

- **Download**: ImageNet ILSVRC2012
- **License**: Academic/Research use only
- **Eval path**: ${DATA_ROOT}/ILSVRC2012/val

```
ILSVRC2012/
├── val/
│   ├── n01440764/
│   │   └── .jpeg ×50
│   ├── n01443537/
│   │   └── .jpeg ×50
│   ├── n01484850/
│   │   └── .jpeg ×50
│   ├── n01491361/
│   │   └── .jpeg ×50
│   ├── n01494475/
│   │   └── .jpeg ×50
│   ├── n01496331/
│   │   └── .jpeg ×50
│   └── ... (994 more directories)
└── .txt ×3
```

4.3.3 Object Detection

© Copyright 2026 DEEPX All Rights Reserved.

DOTA v1

- **Download**: DOTA v1
- **License**: Research use only
- **Eval path**: `${DATA_ROOT}/DOTAv1`

```
DOTAv1/
├── images/
│   ├── test/
│   │   └── .jpg ×937
│   ├── train/
│   │   └── .jpg ×1411
│   └── val/
│       └── .jpg ×458
└── labels/
    ├── train/
    │   └── .txt ×1411
    ├── train_original/
    │   └── .txt ×1411
    ├── val/
    │   └── .txt ×458
    └── val_original/
        └── .txt ×458
```

Pascal VOC 2007/2012

- **Download**: Pascal VOC 2007/2012
- **License**: "VOC2007" challenge data

Dataset classes:

- **PascalVOC2007**: Object Detection
- **PascalVOC2012**: Semantic Segmentation
- **Eval path (VOC2012)**: `${DATA_ROOT}/PascalVOC/VOCdevkit/VOC2012`
- **Eval path (VOC2007)**: `${DATA_ROOT}/PascalVOC/VOCdevkit/VOC2007`

```
PascalVOC/
└── VOCdevkit/
    ├── VOC2007/
    │   ├── Annotations/
    │   ├── ImageSets/
    │   ├── JPEGImages/
    │   ├── SegmentationClass/
    │   └── SegmentationObject/
    └── VOC2012/
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    ├── Annotations/
    ├── ImageSets/
    ├── JPEGImages/
    ├── SegmentationClass/
    └── SegmentationObject/
```

4.3.4 Segmentation

COCO 2017

- **Download**: COCO 2017
- **License**: Annotations under CC BY 4.0; images remain under their original owners' rights and Flickr Terms of Use. See COCO Terms of Use.

Dataset classes:

- **COCO**: Instance Segmentation, Object Detection, Semantic Segmentation
- **COCOPose**: Pose Estimation
- **Eval path**: `${DATA_ROOT}/COCO`

```
COCO/
├── annotations/
│   ├── instances_val2017.json
│   └── person_keypoints_val2017.json
└── val2017/
    └── .jpg ×5000
```

Oxford-IIIT Pet

- **Download**: Oxford-IIIT Pet
- **License**: CC BY-SA 4.0 (free for research and commercial use)
- **Eval path**: `${DATA_ROOT}/Oxford-IIIT_Pet`

```
Oxford-IIIT_Pet/
├── annotations/
│   ├── trimaps/
│   │   └── .png ×14780
│   ├── xmls/
│   │   └── .xml ×3686
│   └── .txt ×3,  ×2
└── images/
    └── .jpg ×7390, .mat ×3
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

Cityscapes

- **Download**: Cityscapes
- **License**: Academic/Research use only
- **Eval path**: `${DATA_ROOT}/cityscapes`

```
cityscapes/
├── gtFine/
│   └── val/
│       ├── frankfurt/
│       ├── lindau/
│       └── munster/
├── leftImg8bit/
│   └── val/
│       ├── frankfurt/
│       ├── lindau/
│       └── munster/
└── .txt ×2
```

4.3.5 Pose / Landmark

AFLW2000-3D

- **Download**: AFLW2000-3D
- **License**: Research use only
- **Eval path**: `${DATA_ROOT}/AFLW20003D`

```
AFLW20003D/
├── Code/
│   └── .pkl ×3, .npy ×2, .txt ×1
└── .jpg ×2000, .mat ×2000
```

Hand Keypoints

- **License**: Custom dataset
- **Eval path**: `${DATA_ROOT}/hand-keypoints`

```
hand-keypoints/
├── images/
│   ├── train/
│   │   └── .jpg ×18776
│   └── val/
│       └── .jpg ×7992
├── labels/
```


© Copyright 2026 DEEPX All Rights Reserved.

```
│   ├── train/
│   │   └── .txt ×18776
│   └── val/
│       └── .txt ×7992
└── .yaml ×1
```

4.3.6 Face

CelebA

- **Download**: CelebA
- **License**: Non-commercial research only
- **Eval path**: `${DATA_ROOT}/CelebA`

```
CelebA/
├── img_align_celeba/
│   └── .jpg ×19962
└── .csv ×4
```

LFW (Labeled Faces in the Wild)

- **Download**: LFW (Labeled Faces in the Wild)
- **License**: Research use only
- **Eval path**: `${DATA_ROOT}/LFW`

```
LFW/
├── lfw-deepfunneled/
│   ├── Aaron_Eckhart/
│   │   └── .jpg ×1
│   ├── Aaron_Guiel/
│   │   └── .jpg ×1
│   ├── Aaron_Patterson/
│   │   └── .jpg ×1
│   ├── Aaron_Peirsol/
│   │   └── .jpg ×4
│   ├── Aaron_Pena/
│   │   └── .jpg ×1
│   ├── Aaron_Sorkin/
│   │   └── .jpg ×2
│   └── ... (5743 more directories)
└── .csv ×10
```


© Copyright 2026 DEEPX All Rights Reserved.

WiderFace

- **Download**: WiderFace
- **License**: Non-commercial research only
- **Eval path**: `${DATA_ROOT}/widerface`

```
widerface/
├── eval_tools/
│   ├── ground_truth/
│   │   └── .mat ×4
│   ├── plot/
│   │   ├── baselines/
│   │   ├── figure/
│   │   └── .m ×4,  ×1
│   └── .m ×6,  ×1
├── wider_face_split/
│   └── .txt ×4, .mat ×3
├── WIDER_train/
├── WIDER_val/
│   └── images/
│       ├── 0--Parade/
│       ├── 1--Handshaking/
│       ├── 10--People_Marching/
│       ├── 11--Meeting/
│       ├── 12--Group/
│       ├── 13--Interview/
│       └── ... (55 more directories)
└── .zip ×1
```

4.3.7 Person

PETA (PEdesTrian Attribute)

- **Download**: PETA (PEdesTrian Attribute)
- **License**: Research use only
- **Eval path**: `${DATA_ROOT}/PETA`

```
PETA/
├── images/
│   └── .png ×19000
└── .mat ×1,  ×1
```

4.3.8 Image Restoration

© Copyright 2026 DEEPX All Rights Reserved.

BSD100

- **Download**: BSD100
- **License**: BSD license (free for research and commercial use)
- **Eval path**: `${DATA_ROOT}/BSD100/bicubic_4x`

```
BSD100/
├── bicubic_2x/
│   ├── HR/
│   │   └── .png ×100
│   └── LR/
│       └── .png ×100
├── bicubic_3x/
│   ├── HR/
│   │   └── .png ×100
│   └── LR/
│       └── .png ×100
└── bicubic_4x/
    ├── HR/
    │   └── .png ×100
    └── LR/
        └── .png ×100
```

BSD68

- **Download**: BSD68
- **License**: BSD license (free for research and commercial use)
- **Eval path**: `${DATA_ROOT}/BSD68`

```
BSD68/
└── .png ×68
```

CBSD68

- **Download**: CBSD68
- **License**: BSD license (free for research and commercial use)
- **Eval path**: `${DATA_ROOT}/CBSD68`

```
CBSD68/
└── .png ×68
```


© Copyright 2026 DEEPX All Rights Reserved.

LOL (Low-Light)

- **Download**: LOL (Low-Light)
- **License**: Research use only
- **Eval path**: `${DATA_ROOT}/LOL`

```
LOL/
├── eval15/
│   ├── high/
│   │   └── .png ×15
│   └── low/
│       └── .png ×15
└── our485/
    ├── high/
    │   └── .png ×485
    └── low/
        └── .png ×485
```

4.3.9 Depth

NYU Depth v2

- **Download**: NYU Depth v2
- **License**: Research use only
- **Eval path**: `${DATA_ROOT}/nyudepthv2`

```
nyudepthv2/
└── val/
    └── official/
        └── .h5 ×654
```

4.3.10 Other

Market-1501

- **Download**: Market-1501
- **License**: Research use only
- **Eval path**: `${DATA_ROOT}/Market1501`

```
Market1501/
├── bounding_box_test/
│   └── .jpg ×19732
```


© Copyright 2026 DEEPX All Rights Reserved.

```
└── query/
    └── .jpg ×3368
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

4.4 Custom Models

This guide explains how to add a new model to DX-ModelZoo. All you need is a single YAML file and, optionally, custom preprocessing/postprocessing operations.

Prerequisites

Before creating custom models, make sure you have:

- Installed DX-ModelZoo with development dependencies
- Basic understanding of YAML configuration
- Model file in ONNX format (or other supported formats)
- Familiarity with preprocessing/postprocessing concepts

Decision Flow

```
graph LR
    A[Write YAML] --> B{Need custom ops?}
    B -->|No| C[Ready to use]
    B -->|Yes| D[Write custom_ops.py]
    D --> E[Register with @REGISTRY]
    E --> C
```

Creating a model involves the following 3 steps:

4.4.1 Step 1: Creating YAML File

Model YAML files are organized by domain, task, and model family:

Directory structure:

```
src/dx_modelzoo/models/
└── <domain>/          # e.g., cv, nlp, audio
    └── <task>/        # e.g., image_classification, object_detection
        └── <Family>/  # e.g., ResNet, EfficientNet
            └── <ModelName>.yaml
```

Example:

```
src/dx_modelzoo/models/
└── cv/
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    └── image_classification/
        └── MyModel/
            └── MyModel_v1.yaml    ← create here
```

Environment Variables in YAML

DX-ModelZoo supports environment variable substitution in YAML files (e.g. `${DATA_ROOT}`, `${MODEL_ROOT}`, `${MODEL_NAME}`, `${DXMZ_MODEL_URL}`). See YAML Configuration for the full list and syntax.

Minimal YAML Example

MyModel_v1.yaml

```yaml
name: MyModel_v1
task: image_classification

inputs:
  - name: input
    shape: [1, 3, 224, 224]
    dtype: float32

preprocessing:
  - type: resize
    size: [224, 224]

postprocessing:
  - type: topk
    k: [1, 5]

evaluator:
  type: image_classification

dataset:
  type: ILSVRC2012
  eval_path: ${DATA_ROOT}/ILSVRC2012/val

artifacts:
  path: ${MODEL_ROOT}/${MODEL_NAME}

profiles:
  onnx:
    target: onnx
    runtime:
      device: gpu
      batch_size: 1
```


© Copyright 2026 DEEPX All Rights Reserved.

See YAML Configuration for the complete field reference, including multi-step preprocessing pipelines, NPU (`q-lite` / `q-pro`) profiles, and calibration settings.

> **Leverage Built-in Types**
>
> When using built-in types for preprocessing, postprocessing, datasets, and evaluators, the model can be fully defined in YAML alone — no Python code required.

4.4.2 Step 2: Custom Operations

If the built-in types are insufficient, create a `custom_ops.py` file in the **same directory as your model YAML**:

```
src/dx_modelzoo/models/
└── cv/
    └── image_classification/
        └── MyModel/
            ├── MyModel_v1.yaml
            └── custom_ops.py      ← create here
```

Custom Preprocessing

custom_ops.py

```python
import numpy as np
from dx_modelzoo.preprocessing import PREPROCESSING_REGISTRY


@PREPROCESSING_REGISTRY.register("my_letterbox")
class MyLetterbox:
    """Aspect-ratio-preserving resize with padding"""

    def __init__(self, target_size: list[int], pad_value: int = 114):
        self.target_size = tuple(target_size)  # (H, W)
        self.pad_value = pad_value

    def __call__(self, img: np.ndarray) -> np.ndarray:
        h, w = img.shape[:2]
        th, tw = self.target_size
        scale = min(tw / w, th / h)
        new_w, new_h = int(w * scale), int(h * scale)

        import cv2
        resized = cv2.resize(img, (new_w, new_h))
        canvas = np.full((th, tw, 3), self.pad_value, dtype=np.uint8)
        top = (th - new_h) // 2
```


© Copyright 2026 DEEPX All Rights Reserved.

```
        left = (tw - new_w) // 2
        canvas[top:top + new_h, left:left + new_w] = resized
        return canvas
```

Custom Postprocessing

custom_ops.py (continued)

```
from dx_modelzoo.postprocessing import POSTPROCESSING_REGISTRY


@POSTPROCESSING_REGISTRY.register("my_decode")
class MyDecode:
    """Custom decoding logic"""

    def __init__(self, conf_threshold: float = 0.25):
        self.conf_threshold = conf_threshold

    def __call__(self, outputs: list[np.ndarray], **kwargs) -> dict:
        # kwargs carries pipeline context (runtime_target, and origin_hw/input_hw
        # for detection models) — always accept **kwargs
        predictions = outputs[0]
        mask = predictions[..., 4] > self.conf_threshold
        return {"detections": predictions[mask]}
```

Warning

`PostprocessingPipeline` forwards context keyword arguments to every step. A `__call__(self, outputs)` signature raises `TypeError: __call__() got an unexpected keyword argument 'runtime_target'`. For coordinate outputs, rescale with `origin_hw` / `input_hw` — see Postprocessing API.

Using Custom Operations in YAML

MyDetector.yaml

```
preprocessing:
  - type: my_letterbox        # custom type
    target_size: [640, 640]
    pad_value: 114
  - type: div
    x: 255
  - type: transpose
    axis: [2, 0, 1]
  - type: expanddim
    axis: 0
```


© Copyright 2026 DEEPX All Rights Reserved.

```yaml
postprocessing:
  - type: my_decode          # custom type
    conf_threshold: 0.3
```

Auto-Import

`custom_ops.py` placed in the same directory as the model YAML is automatically imported by `ModelBuilder`. No manual import is needed — just define your classes with `@REGISTRY.register()` and they become available in the YAML configuration.

4.4.3 Step 3: Verification

Run the evaluation to confirm the model works as expected:

```bash
# Run by model name
dxmz eval MyModel_v1 --profile onnx

# Expected output:
# Loading model: MyModel_v1
# Dataset: ILSVRC2012 (50000 samples)
# Evaluating... [==============================] 100%
# Top-1 Accuracy: 76.13%
# Top-5 Accuracy: 92.86%
# FPS: 245.3

# Run with dataset path override
dxmz eval MyModel_v1 --profile onnx --data-root /path/to/datasets

# Run by YAML path
dxmz eval path/to/MyModel_v1.yaml --profile onnx

# Specify model file directly
dxmz eval MyModel_v1 --profile onnx --model-path /path/to/model.onnx
```

4.4.4 Complete Example

A full custom detection model configuration:

models/custom/MyDet/MyDet_s.yaml

```yaml
name: MyDet_s
task: object_detection
reference: https://github.com/my-org/mydet
description: >
  Lightweight object detection model (COCO mAP 42.1)
```


© Copyright 2026 DEEPX All Rights Reserved.

```yaml
inputs:
  - name: images
    shape: [1, 3, 640, 640]
    dtype: float32
    layout: NCHW

preprocessing:
  - type: my_letterbox
    target_size: [640, 640]
  - type: convertcolor
    form: BGR2RGB
  - type: div
    x: 255
  - type: transpose
    axis: [2, 0, 1]
  - type: expanddim
    axis: 0

postprocessing:
  - type: my_decode
    conf_threshold: 0.25

evaluator:
  type: object_detection

dataset:
  type: COCO
  eval_path: ${DATA_ROOT}/COCO

artifacts:
  path: ${MODEL_ROOT}/${MODEL_NAME}

calibration: &calibration
  method: ema
  num: 100
  seed: 42

profiles:
  onnx:
    target: onnx
    runtime:
      device: gpu
  q-lite:
    target: dxnn
    compile:
      calibration: *calibration
    runtime:
      device: 0
      async: true
  q-pro:
    target: dxnn
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```yaml
  compile:
    calibration:
      <<: *calibration
      num: 1024
      use_q_pro: true
  runtime:
    device: 0
    async: true
```

4.4.5 Model-Specific Evaluator Integration

For models that require custom decoding of model outputs (e.g., pose estimation, text detection), the recommended pattern is:

- Step 1. Define the decoder in `custom_ops.py` with `@POSTPROCESSING_REGISTRY.register()`
- Step 2. Reference it in the YAML `postprocessing` section
- Step 3. The evaluator receives the decoded output in `process_batch_result()`

```mermaid
graph LR
    A[Session.run] -->|raw output| B["custom_ops.py<br/>(decode)"]
    B -->|standardized data| C["Evaluator<br/>(metrics)"]
```

Example: Full Pipeline

models/custom/PP_OCRv5_Det/custom_ops.py

```python
@POSTPROCESSING_REGISTRY.register("db_text_decode")
class DBTextDecode:
    """Binarise DB probability map → list of polygon arrays."""
    def __init__(self, db_thresh=0.3, db_box_thresh=0.5, **kwargs):
        self.db_thresh = db_thresh
        self.db_box_thresh = db_box_thresh

    def __call__(self, outputs, **kwargs):
        prob_map = outputs[0]
        # ... decode logic ...
        return polygons  # List[np.ndarray [4,2]]
```

PP_OCRv5_Det.yaml

```yaml
postprocessing:
  - type: db_text_decode
    db_thresh: 0.3
    db_box_thresh: 0.5
```


© Copyright 2026 DEEPX All Rights Reserved.

```
evaluator:
  type: text_detection
```

The evaluator then receives `polygons` directly and only computes IoU-based precision/recall metrics.

> ⚠ **`text_detection` is not a built-in evaluator**
>
> There is no registered `text_detection` evaluator (nor a `TextDetectionEvaluator`) in v1.0.0. The example above assumes **you register your own** evaluator under that key with `@EVALUATOR_REGISTRY.register("text_detection")`; otherwise the build fails with an unknown-evaluator error. See Evaluator API — Writing a Custom Evaluator.

Pipeline Evaluators (Multi-Model)

Some evaluators orchestrate multiple models (e.g., OCR = Detection + Recognition). In these cases:

- The evaluator manages session lifecycle and data flow
- Each model's decode logic still lives in its respective `custom_ops.py`
- The evaluator calls postprocessors explicitly via lazy-init imports

```
# Inside OCR Evaluator (simplified)
def process_batch_result(self, batch_data, output, metrics_state):
    # Det model output → polygons (via custom_ops)
    polygons = self._get_db_decoder()([output])

    for poly in polygons:
        crop = extract_crop(image, poly)                    # evaluator logic
        rec_input = self._get_rec_preprocessor()(crop)      # custom_ops
        rec_output = self.rec_session.run(rec_input)        # session
        text = self._get_ctc_decoder()(rec_output)          # custom_ops

    # Compute Deteval metrics (evaluator logic)
    ...
```

4.4.6 Troubleshooting

Item	Check
Is the YAML file located under `models/<domain>/<task>/<Family>/` ?	☑

© Copyright 2026 DEEPX All Rights Reserved.

Item	Check
Does the `name` field match the filename?	☑
Do the `inputs` shape/dtype match the model?	☑
Are custom ops registered with `@REGISTRY.register()`?	☑
Is `custom_ops.py` in the same directory as the YAML?	☑
Are custom postprocessors receiving `**kwargs`?	☑
Are the required runtimes defined in `profiles`?	☑
Does `dxmz eval` run successfully?	☑

ⓒ Copyright 2026 DEEPX All Rights Reserved.

4.5 Custom Datasets

This guide explains how to create and integrate custom datasets into DX-ModelZoo for your specific evaluation needs.

Prerequisites

Before creating custom datasets, make sure you have:

- Installed DX-ModelZoo with development dependencies
- Basic understanding of Python classes and abstract base classes
- Familiarity with the evaluation workflow

Overview

```
graph LR
    A[DatasetBase ABC] --> B[Custom Dataset]
    B --> C[DATASET_REGISTRY.register]
    C --> D[Reference in YAML]
    D --> E[DataLoader auto-uses it]
```

4.5.1 DatasetBase Interface

All datasets must inherit from the `DatasetBase` abstract class:

```python
from dx_modelzoo.common.dataloader import DatasetBase

class DatasetBase(ABC):
    def __init__(self, data_dir: str):
        self.data_dir = data_dir
        self._preprocessing = None

    @property
    def preprocessing(self):
        """Preprocessing pipeline (automatically set by the Evaluator)"""
        return self._preprocessing

    @preprocessing.setter
    def preprocessing(self, value):
        self._preprocessing = value

    @abstractmethod
    def __len__(self) -> int:
```


© Copyright 2026 DEEPX All Rights Reserved.

```
        """Return the size of the dataset"""
        ...

    @abstractmethod
    def __getitem__(self, idx: int) -> tuple:
        """Return a sample by index"""
        ...

    @abstractmethod
    def validate(self) -> None:
        """Validate dataset files and directory structure"""
        ...
```

4.5.2 Evaluator Compatibility

The return format of `__getitem__` must match the Evaluator's `extract_inputs` method:

Evaluator (registry key)	Return Format	Description
`image_classification`	`(image, label)`	label: int
`object_detection`	`(image, origin_shape, image_id)`	Dataset must also expose `coco_annotation` and `remap_class_id()`
`instance_segmentation`	`(image, origin_shape, image_id)`	Same contract as `object_detection`
`semantic_segmentation`	`(image, mask)`	mask: np.ndarray
`depth_estimation`	`(image, depth_map)`	depth_map: np.ndarray
`face_detection`	`(image, boxes)`	boxes: np.ndarray

Compatibility with Built-in Evaluators

To use a built-in Evaluator, your dataset's return format must match exactly what that Evaluator's `extract_inputs` expects.

© Copyright 2026 DEEPX All Rights Reserved.

> ⚠ **There is no `coco` or `segmentation` evaluator key**
>
> Registry keys are exact. Use `object_detection` for COCO-style detection and `semantic_segmentation` for segmentation. Run `dxmz list` / check `src/dx_modelzoo/evaluator/` for the full set of registered keys; an unknown key fails at build time.

4.5.3 Implementation

Creating a Custom Dataset Class

Create a new Python file in the `src/dx_modelzoo/dataset/` directory:

src/dx_modelzoo/dataset/my_dataset.py

```
import os
from pathlib import Path

import cv2
import numpy as np

from dx_modelzoo.common.dataloader import DatasetBase
from dx_modelzoo.dataset import DATASET_REGISTRY


@DATASET_REGISTRY.register("my_custom_dataset")
class MyCustomDataset(DatasetBase):
    """Custom image classification dataset

    Directory structure:
        data_dir/
        ├── images/
        │   ├── 00001.jpg
        │   └── ...
        └── labels.txt        # "filename label" format
    """

    def __init__(self, data_dir: str):
        super().__init__(data_dir)
        self._samples = self._load_annotations()

    def validate(self) -> None:
        self.ensure_exists(
            self.data_dir,
            "Create the dataset directory and provide labels.txt before evaluation.",
        )
        label_file = Path(self.data_dir) / "labels.txt"
        if not label_file.is_file():
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
            raise FileNotFoundError(f"Missing annotation file: {label_file}")

    def _load_annotations(self) -> list[tuple[str, int]]:
        """Parse the annotation file and return a list of (image_path, label) tuples"""
        label_file = Path(self.data_dir) / "labels.txt"
        samples = []
        with label_file.open("r", encoding="utf-8") as f:
            for line in f:
                filename, label = line.strip().split()
                img_path = os.path.join(self.data_dir, "images", filename)
                samples.append((img_path, int(label)))
        return samples

    def __len__(self) -> int:
        return len(self._samples)

    def __getitem__(self, idx: int) -> tuple:
        img_path, label = self._samples[idx]

        # Load image (BGR format)
        img = cv2.imread(img_path)

        # Apply preprocessing (pipeline set by the Evaluator)
        if self._preprocessing is not None:
            img = self._preprocessing(img)

        return img, label
```

Where to Apply `preprocessing`

`self.preprocessing` is automatically configured by the Evaluator via `set_preprocessing()`. The standard pattern is to apply it in `__getitem__` after loading the image.

4.5.4 Registration

Dataset modules are imported from `src/dx_modelzoo/dataset/__init__.py` to trigger registry registration. Add an import for the new module there; placing a file in the directory alone does not register it.

Reference in YAML

MyModel.yaml

```
dataset:
  type: my_custom_dataset       # must match the register() name
  eval_path: ${DATA_ROOT}/my_data
```


© Copyright 2026 DEEPX All Rights Reserved.

Running Evaluation

```
dxmz eval MyModel --profile onnx --data-root /path/to/datasets
```

Example: Complete Workflow

```
# 1. Create dataset structure
mkdir -p /data/my_dataset/images
echo "img001.jpg 0" > /data/my_dataset/labels.txt
echo "img002.jpg 1" >> /data/my_dataset/labels.txt

# 2. Set environment variable
export DATA_ROOT=/data

# 3. Run evaluation
dxmz eval MyModel --profile onnx --data-root /data
```

What happens:

- Step 1. DX-ModelZoo loads your YAML configuration
- Step 2. Finds `dataset.type: my_custom_dataset` in the YAML
- Step 3. Resolves `dataset.type: my_custom_dataset` through `DATASET_REGISTRY`
- Step 4. Instantiates `MyCustomDataset` and runs `validate()`
- Step 5. Creates DataLoader with your dataset
- Step 6. Runs evaluation

4.5.5 COCO-Format Example

A full example implementing a COCO-format custom dataset:

src/dx_modelzoo/dataset/my_coco.py

```
import json
from pathlib import Path

import cv2
import numpy as np

from dx_modelzoo.common.dataloader import DatasetBase
from dx_modelzoo.dataset import DATASET_REGISTRY


@DATASET_REGISTRY.register("my_coco_format")
class MyCocoDataset(DatasetBase):
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
    """Custom dataset in COCO JSON format, compatible with `object_detection`"""

    def __init__(self, data_dir: str):
        super().__init__(data_dir)
        self._coco = self._load_annotations()
        self._images = self._coco.dataset["images"]

    def validate(self) -> None:
        self.ensure_exists(
            self.data_dir,
            "Create the dataset directory and provide annotations.json before evaluation.",
        )
        if not (Path(self.data_dir) / "annotations.json").is_file():
            raise FileNotFoundError(f"Missing annotation file: {Path(self.data_dir) / 'annotations.json'}")

    def _load_annotations(self):
        from faster_coco_eval import COCO as COCOApi

        return COCOApi(str(Path(self.data_dir) / "annotations.json"))

    @property
    def coco_annotation(self):
        """Ground truth in COCO API form — required by ObjectDetectionEvaluator."""
        return self._coco

    def remap_class_id(self, model_class_id: int) -> int:
        """Map a model class index to the dataset's category ID."""
        return int(model_class_id)

    def __len__(self) -> int:
        return len(self._images)

    def __getitem__(self, idx: int) -> tuple:
        img_info = self._images[idx]
        img_path = Path(self.data_dir) / img_info["file_name"]
        img = cv2.imread(str(img_path))
        origin_shape = (img.shape[0], img.shape[1])

        if self._preprocessing is not None:
            img = self._preprocessing(img)

        # ObjectDetectionEvaluator expects exactly (image, origin_shape, image_id)
        return img, origin_shape, img_info["id"]
```

Corresponding YAML configuration

```
dataset:
  type: my_coco_format
  eval_path: ${DATA_ROOT}/my_coco_data
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
evaluator:
  type: object_detection
```

> **Warning**
>
> Returning `(image, annotation)` raises `ValueError: not enough values to unpack (expected 3, got 2)` inside the evaluator's postprocessing-context builder. `origin_shape` is `(H, W)` of the **original** image and is used to map boxes back from model input space.

4.5.6 DataLoader Integration

Datasets are automatically batched by the `DataLoader`:

```python
from dx_modelzoo.common.dataloader import make_dataloader

dataset = MyCustomDataset("/path/to/data")
loader = make_dataloader(dataset, batch_size=4, shuffle=False, num_workers=4)

for batch in loader:
    images, labels = batch
    # images: np.ndarray (batch_size, ...)
```

> **NumPy-Based**
>
> The DataLoader is implemented in pure Python/NumPy. PyTorch is NOT required for the DataLoader itself, though torchvision is used by the preprocessing pipeline for dx_com fusion compatibility. Ensure that `__getitem__` returns NumPy arrays.

4.5.7 Troubleshooting

Item	Check
Does the class inherit from `DatasetBase`?	☑
Are `__len__` and `__getitem__` implemented?	☑
Is the class registered with `@DATASET_REGISTRY.register()`?	☑
Does `__getitem__` apply `self.preprocessing`?	☑
	☑

© Copyright 2026 DEEPX All Rights Reserved.

Item	Check
Does `dataset.type` in YAML match the registered name?	
Is the return format compatible with the Evaluator's `extract_inputs`?	☑

ⓒ Copyright 2026 DEEPX All Rights Reserved.

5. Architecture

5.1 Architecture Overview

The DX-ModelZoo architecture is designed around a YAML-centric configuration approach, where `ModelBuilder` and the Registry pattern connect all components for runtime execution.

5.1.1 Architecture Diagram

The following diagram shows how components connect at runtime:

```
flowchart LR
    subgraph CLI["CLI Layer"]
        direction TB
        DXMZ[dxmz CLI]
        EVAL[eval]
        COMP[compile]
        BENCH[benchmark]
        DXMZ --- EVAL & COMP & BENCH
    end

    subgraph Loader["Loader"]
        direction TB
        Y["📄 MyModel.yaml"]
        YL[YamlLoader]
        DI[Discovery]
        MB[ModelBuilder]
        Y --> YL --> MB
        DI --> MB
    end

    subgraph Registry["Registry"]
        direction TB
        PR[PREPROCESSING]
        PO[POSTPROCESSING]
        DR[DATASET]
        ER[EVALUATOR]
    end

    subgraph Runtime["Runtime"]
        direction TB
        PP[PreprocessingPipeline]
        POP[PostprocessingPipeline]
        DS[Dataset]
        DL[DataLoader]
```


© Copyright 2026 DEEPX All Rights Reserved.

```
        EV[Evaluator]
        SF[SessionFactory]
        ORT[OnnxRuntime]
        DXR[DxRuntime]
        SF --- ORT & DXR
        DS --> DL --> EV
    end

    CLI --> Loader
    MB --> PR --> PP
    MB --> PO --> POP
    MB --> DR --> DS
    MB --> ER --> EV
    MB --> SF
    PP -.-> DS
    POP -.-> EV
```

`ModelBuilder` connects the CLI, Registry, and Runtime layers shown above.

5.1.2 Core Design Principles

YAML-Centric

Every model definition is fully self-contained within a YAML file. Preprocessing, postprocessing, datasets, and profiles are configured declaratively without writing any Python code.

```yaml
# A model is fully defined by this alone
name: resnet50_224x224
preprocessing: [...]
postprocessing: [...]
evaluator: { type: image_classification }
dataset: { type: ILSVRC2012, eval_path: ... }
profiles: { onnx: ..., q-lite: ... }
```

Registry Pattern Everywhere

Four registries manage all extensible components:

Registry	Role	Examples
`PREPROCESSING_REGISTRY`	Preprocessing operations	`resize`, `normalize`, `div`
`POSTPROCESSING_REGISTRY`	Postprocessing operations	`topk`, `nms`, `identity`
`DATASET_REGISTRY`	Dataset loaders	

© Copyright 2026 DEEPX All Rights Reserved.

Registry	Role	Examples
		`ILSVRC2012`, `COCO`, `PascalVOC2007`
`EVALUATOR_REGISTRY`	Evaluation logic	`image_classification`, `object_detection`

Postprocessing vs. Evaluator

DX-ModelZoo strictly separates output decoding from metric computation:

Component	Role
Postprocessing (`custom_ops.py` or shared `postprocessing/`)	Transforms raw model tensors into a standardized, task-specific format
Evaluator	Consumes the standardized output and computes quality metrics (mAP, accuracy, AUROC, etc.)

Builder Pattern

`ModelBuilder` is responsible for transforming YAML into runtime components:

```
sequenceDiagram
    participant CLI as dxmz eval
    participant MB as ModelBuilder
    participant R as Registries
    participant RT as Runtime

    CLI->>MB: ModelBuilder - resnet50_224x224.yaml
    MB->>MB: load_yaml + validate
    MB->>R: PREPROCESSING_REGISTRY.get - resize
    MB->>R: POSTPROCESSING_REGISTRY.get - topk
    MB->>R: DATASET_REGISTRY.get - ILSVRC2012
    MB->>R: EVALUATOR_REGISTRY.get - image_classification
    MB->>RT: create_session - onnx
    MB-->>CLI: session, dataset, evaluator, pre/post
```

5.1.3 Source Directory Structure

The corresponding source code is organized as follows:

© Copyright 2026 DEEPX All Rights Reserved.

```
src/dx_modelzoo/
├── main.py              # CLI entry point (Typer)
├── command/
│   ├── compile.py       # NPU compilation logic
│   ├── benchmark.py     # Multi-model batch evaluation
│   └── create_yaml.py   # Interactive YAML creation
├── common/
│   ├── dataloader.py    # Pure numpy DataLoader
│   ├── enums.py         # DeviceType, EvaluationType, DatasetType
│   ├── registry.py      # Registry pattern
│   ├── seed.py          # Global seed management
│   └── ...
├── preprocessing/       # Registered preprocessing ops + pipeline
├── postprocessing/      # Shared ops + decode utilities
├── evaluator/           # Task evaluators
├── dataset/             # Registered dataset classes
├── loader/
│   ├── yaml_loader.py   # YAML parse + validation
│   ├── model_builder.py # Build all runtime objects from YAML
│   ├── model_scaffold.py# YAML creation scaffolding
│   └── discovery.py     # Model directory scanning
├── session/
│   ├── onnx_session.py  # ONNX Runtime backend
│   ├── dx_session.py    # DxRuntime (NPU) backend
│   ├── runtime_config.py# RuntimeConfig dataclasses
│   └── factory.py       # Session creation + auto-download
├── models/              # 300+ YAML model configs
└── tui/                 # Interactive model browser + wizard
```

5.1.4 Data Flow

The data flow through the full evaluation process:

```
graph TB
    A[Raw Image] -->|Dataset.__getitem__| B[np.ndarray BGR]
    B -->|PreprocessingPipeline| C[Preprocessed Tensor]
    C -->|Session.run| D[Raw Model Output]
    D -->|PostprocessingPipeline<br/>includes custom_ops| E[Standardized Result]
    E -->|Evaluator.process_batch_result| F[Metrics]
```

Stage	Component	Data Format
Load	Dataset	BGR `np.ndarray` (H, W, 3)
Preprocessing	PreprocessingPipeline	float32 `np.ndarray` (1, C, H, W)
Inference	Session	

© Copyright 2026 DEEPX All Rights Reserved.

Stage	Component	Data Format
		`list[np.ndarray]` (raw model outputs)
Postprocessing	PostprocessingPipeline + custom_ops	Task-specific standardized format
Evaluation	Evaluator	metrics dict

NPU Mode

When using a `target: dxnn` profile, arithmetic preprocessing operations (`div`, `normalize`, `transpose`, etc.) are automatically skipped. This is because the NPU handles these operations at the hardware level.

© Copyright 2026 DEEPX All Rights Reserved.

5.2 Pipeline

The DX-ModelZoo preprocessing and postprocessing pipelines execute operations sequentially in the order defined in YAML configuration.

5.2.1 Pipeline Overview

The following diagram shows how preprocessing, inference, and postprocessing connect:

```
graph TB
    subgraph "PreprocessingPipeline"
        direction TB
        A[Raw Image] --> B[resize]
        B --> C[centercrop]
        C --> D[convertcolor]
        D --> E[div]
        E --> F[normalize]
        F --> G[transpose]
        G --> H[expanddim]
    end

    subgraph "Session"
        I[Model Inference]
    end

    subgraph "PostprocessingPipeline"
        direction TB
        J[Raw Output] --> K[topk / nms / ...]
    end

    H --> I --> J
```

5.2.2 PreprocessingPipeline

Applies preprocessing operations sequentially to the input data (images).

Class Structure

```
class PreprocessingPipeline(Compose):
    """Inherits from torchvision.transforms.Compose."""

    def __init__(
        self,
        steps_config: Union[List[Dict], Dict[str, List[Dict]]],
        npu_mode: bool = False,
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
    ):
        # Builds steps from config dicts; skips NPU-handled ops when npu_mode=True
        ...

    def __call__(self, img: np.ndarray) -> np.ndarray:
        for step in self.steps:
            img = step(img)
        return img
```

YAML to Pipeline Mapping

YAML Definition

```yaml
preprocessing:
  - type: resize
    size: [256, 256]
    mode: torchvision
  - type: centercrop
    height: 224
    width: 224
  - type: div
    x: 255
  - type: normalize
    mean: [0.485, 0.456, 0.406]
    std: [0.229, 0.224, 0.225]
  - type: transpose
    axis: [2, 0, 1]
  - type: expanddim
    axis: 0
```

The above YAML is transformed into the following:

```
# Automatically generated by ModelBuilder
pipeline = PreprocessingPipeline(
    steps_config=[
        {"type": "resize", "size": [256, 256], "mode": "torchvision"},
        {"type": "centercrop", "height": 224, "width": 224},
        {"type": "div", "x": 255},
        {"type": "normalize", "mean": [0.485, 0.456, 0.406], "std": [0.229, 0.224, 0.225]},
        {"type": "transpose", "axis": [2, 0, 1]},
        {"type": "expanddim", "axis": 0},
    ],
    npu_mode=False,
)
```

5.2.3 NPU Auto-Skip

© Copyright 2026 DEEPX All Rights Reserved.

For `target: dxnn`, arithmetic preprocessing is skipped when runtime input dtype is `uint8`; it remains for `float32` input unless `appendix.npu_skip_arithmetic` overrides the behavior.

NPU_SKIP_DEFAULT

Arithmetic operations skipped on the NPU are defined in `NPU_SKIP_DEFAULT`: `div`, `normalize`, `transpose`, `expanddim`, `mul`, `add`, `subtract` — see Preprocessing API for the full definition.

Behavior Comparison

ONNX Mode (Full Execution)	NPU Mode (Auto-Skip)

resize → centercrop → convertcolor → div → normalize → transpose → expanddim
☑ ☑ ☑ ☑ ☑ ☑ ☑

All operations are executed in order.

resize → centercrop → convertcolor → div → normalize → transpose → expanddim
☑ ☑ ☑ ⏭ ⏭ ⏭ ⏭

Operations included in `NPU_SKIP_DEFAULT` are skipped.

> **Why Operations Are Skipped**
>
> The NPU accepts `uint8` input and handles `div` (normalization), `normalize`, `transpose` (layout conversion), and `expanddim` (batch dimension) at the hardware level. Running these operations redundantly on the CPU would produce incorrect results.

Skip Flow

```
flowchart TD
    A[Step execution] --> B{npu_mode?}
    B -->|No| C[Execute step]
    B -->|Yes| D{type in NPU_SKIP_DEFAULT?}
    D -->|No| C
    D -->|Yes| E[Skip step]
```

5.2.4 PostprocessingPipeline

Applies postprocessing operations sequentially to the model output.

© Copyright 2026 DEEPX All Rights Reserved.

Class Structure

```
class PostprocessingPipeline:
    def __init__(self, steps: list):
        self.steps = steps

    def __call__(self, outputs):
        for step in self.steps:
            outputs = step(outputs)
        return outputs
```

No NPU Skip

The postprocessing pipeline does not have NPU skip logic. All postprocessing operations are always executed.

YAML to Pipeline Mapping

Detection Model Example

```
postprocessing:
  - type: nms
    conf_thres: 0.001
    iou_thres: 0.7
```

```
# Transformed result
pipeline = PostprocessingPipeline(
    steps_config=[{"type": "nms", "conf_thres": 0.001, "iou_thres": 0.7}]
)
# pipeline.steps -> [NMS(conf_thres=0.001, iou_thres=0.7)]
```

Postprocessing Patterns by Task

See Postprocessing Guide by Task for the full task-to-postprocessing mapping.

5.2.5 Full Execution Flow

The following sequence diagram shows the complete evaluation flow, from raw data to final metrics:

```
sequenceDiagram
    participant D as Dataset
    participant Pre as PreprocessingPipeline
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    participant S as Session
    participant Post as PostprocessingPipeline
    participant E as Evaluator

    D->>D: __getitem__ idx - raw image
    D->>Pre: preprocessing image
    Note over Pre: resize - crop - div - normalize - ...
    Pre-->>D: preprocessed tensor

    D-->>E: tensor, label
    E->>E: extract_inputs batch
    E->>S: session.run input
    S-->>E: raw outputs

    E->>Post: postprocessing outputs
    Note over Post: topk / nms / argmax / ...
    Post-->>E: decoded results

    E->>E: process_batch_result
    E->>E: compute_final_metrics
```

5.2.6 Custom Pipeline Extension

To add a custom operation, register it with the appropriate Registry:

```python
from dx_modelzoo.preprocessing import PREPROCESSING_REGISTRY

@PREPROCESSING_REGISTRY.register("my_augment")
class MyAugment:
    def __init__(self, probability: float = 0.5):
        self.probability = probability

    def __call__(self, img: np.ndarray) -> np.ndarray:
        # Custom logic
        return img
```

```yaml
preprocessing:
  - type: resize
    size: [224, 224]
  - type: my_augment       # Custom operation inserted
    probability: 0.8
  - type: div
    x: 255
```


© Copyright 2026 DEEPX All Rights Reserved.

Pipeline Debugging

Print the shape before and after each step's `__call__` to trace the data transformation process:

```
for step in pipeline.steps:
    print(f"{step.__class__.__name__}: {img.shape}", end=" → ")
    img = step(img)
    print(img.shape)
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

5.3 Registry Pattern

The DX-ModelZoo uses the **Registry pattern** to manage all extensible components. The `type` field in YAML maps to classes registered in a Registry, enabling object creation from strings alone.

5.3.1 Registry Class

`Registry` is a general-purpose class defined in `src/dx_modelzoo/common/registry.py`:

```
class Registry:
    """General-purpose registry managing name-to-class mappings"""

    def __init__(self, name: str) -> None:
        self.name = name
        self._registry: dict[str, type] = {}

    def register(self, name_or_cls: Union[str, type, None] = None):
        """Register a class via decorator.

        Can be used as:
            @REGISTRY.register("name")  — explicit name
            @REGISTRY.register          — auto-derive from class name
        """
        if name_or_cls is None or isinstance(name_or_cls, str):
            explicit_name = name_or_cls
            def decorator(cls):
                key = explicit_name or _derive_name(cls)
                self._registry[key] = cls
                return cls
            return decorator
        else:
            cls = name_or_cls
            key = _derive_name(cls)
            self._registry[key] = cls
            return cls

    def get(self, name: str) -> type:
        """Look up a class by name"""
        if name not in self._registry:
            raise KeyError(f"{self.name}: '{name}' not found")
        return self._registry[name]

    def list(self) -> list[str]:
        """Return all registered names"""
        return list(self._registry.keys())

    def __contains__(self, name: str) -> bool:
```


© Copyright 2026 DEEPX All Rights Reserved.

```
        return name in self._registry

    def __len__(self) -> int:
        return len(self._registry)
```

5.3.2 Core Registries

DX-ModelZoo defines four core registries:

Registry	Location	Purpose
PREPROCESSING_REGISTRY	preprocessing/__init__.py	Preprocessing operations
POSTPROCESSING_REGISTRY	postprocessing/__init__.py	Postprocessing operations
DATASET_REGISTRY	dataset/__init__.py	Datasets
EVALUATOR_REGISTRY	evaluator/__init__.py	Evaluators

```
graph TB
    subgraph "Registry Instances"
        A["PREPROCESSING_REGISTRY"]
        B["POSTPROCESSING_REGISTRY"]
        C["DATASET_REGISTRY"]
        D["EVALUATOR_REGISTRY"]
    end

    subgraph "YAML Config"
        Y1["type: resize"]
        Y2["type: topk"]
        Y3["type: ILSVRC2012"]
        Y4["type: image_classification"]
    end

    Y1 -->|get: resize| A
    Y2 -->|get: topk| B
    Y3 -->|get: ILSVRC2012| C
    Y4 -->|get: image_classification| D
```

5.3.3 Registration

Registering a custom type follows a two-step pattern:

Decorator Pattern (Standard)

© Copyright 2026 DEEPX All Rights Reserved.

```python
from dx_modelzoo.preprocessing import PREPROCESSING_REGISTRY

@PREPROCESSING_REGISTRY.register("my_op")
class MyOp:
    def __init__(self, param1: int = 10):
        self.param1 = param1

    def __call__(self, data):
        # Processing logic
        return data
```

Usage in YAML

```yaml
preprocessing:
  - type: my_op        # Looks up MyOp class from the Registry
    param1: 20         # Instantiated as MyOp(param1=20)
```

5.3.4 Build Process

When ModelBuilder parses YAML, it looks up classes through the Registry and creates instances:

```
sequenceDiagram
    participant MB as ModelBuilder
    participant R as Registry
    participant C as Registered Class

    MB->>MB: Parse YAML: type=resize, size=256
    MB->>R: PREPROCESSING_REGISTRY.get - resize
    R-->>MB: Return Resize class
    MB->>C: Resize size=256
    C-->>MB: resize instance
    Note over MB: Add to pipeline
```

Pipeline Build Process

```python
# Simplified internal logic of ModelBuilder.build_preprocessing()
def build_preprocessing(self, profile_name, ...):
    steps = []
    for step_config in self.config["preprocessing"]:
        type_name = step_config["type"]
        params = {k: v for k, v in step_config.items() if k != "type"}

        # Look up class from Registry and instantiate with parameters
        cls = PREPROCESSING_REGISTRY.get(type_name)
        steps.append(cls(**params))
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    return PreprocessingPipeline(steps, npu_mode=is_npu)
```

5.3.5 Registered Types

The following tables list all built-in registered types:

PREPROCESSING_REGISTRY

Name	Class	Parameters
resize	Resize	size, mode, interpolation
centercrop	CenterCrop	height, width
convertcolor	ConvertColor	form
div	Div	x
normalize	Normalize	mean, std
transpose	Transpose	axis
expanddim	ExpandDim	axis
bgr_to_y_channel	BgrToYChannel	—
bgr_to_y_channel_uint8	BgrToYChannelUint8	—
mul	Mul	x
add	Add	x
subtract	Subtract	x
totensor	ToTensor	—

POSTPROCESSING_REGISTRY

Name	Class	Purpose
identity	Identity	No postprocessing (passthrough)
topk	TopK	Top-k classes for classification

© Copyright 2026 DEEPX All Rights Reserved.

Name	Class	Purpose
nms	NMS	Non-maximum suppression (detection)
segmentation_argmax	SegmentationArgmax	Segmentation argmax

DATASET_REGISTRY

Name	Task
ILSVRC2012	Classification
COCO	Detection / Instance Segmentation
PascalVOC2007	Detection
PascalVOC2012	Segmentation
Cityscapes	Segmentation
WiderFace	Face Detection
COCOPose	Pose Estimation
COCOPoseTopDown	Pose Estimation (Top-Down)
COCOPersonSeg	Person Segmentation
NYUDepthv2	Depth Estimation
LFW	Face Recognition
BSD68 / BSD100 / CBSD68	Super Resolution / Denoising
LOL	Low-light Enhancement
ADE20K	Segmentation
DOTAv1	Oriented Object Detection
OxfordIIITPet	Segmentation
Market1501	Person Re-ID
PETA	Pedestrian Attribute

© Copyright 2026 DEEPX All Rights Reserved.

Name	Task
`HandKeypoints`	Hand Landmark
`HandKeypointsDetection`	Hand Detection

Listing Registered Types

```
from dx_modelzoo.preprocessing import PREPROCESSING_REGISTRY
print(PREPROCESSING_REGISTRY.list())
# ['resize', 'centercrop', 'convertcolor', 'div', 'normalize', ...]
```

5.3.6 Adding Custom Types

The same decorator pattern shown in Registration applies to any registry (`PREPROCESSING_REGISTRY` , `POSTPROCESSING_REGISTRY` , `DATASET_REGISTRY` , `EVALUATOR_REGISTRY`). For more details, see the Custom Model Guide and Custom Dataset Guide.

5.3.7 Auto-Import via `custom_ops.py`

Model-specific postprocessing types don't need to be in the core `postprocessing/` package. Instead, they can be defined in a `custom_ops.py` file within the model's directory.

How It Works

```
sequenceDiagram
    participant CLI as dxmz eval
    participant MB as ModelBuilder
    participant FS as FileSystem
    participant R as POSTPROCESSING_REGISTRY

    CLI->>MB: ModelBuilder("RTMPose_T.yaml")
    MB->>FS: Check model_dir/custom_ops.py
    FS-->>MB: File exists
    MB->>MB: importlib.import_module(custom_ops)
    Note over MB,R: @register decorators execute
    MB->>R: POSTPROCESSING_REGISTRY now has "rtmpose_simcc_decode"
    MB->>R: get("rtmpose_simcc_decode")
    R-->>MB: RTMPoseSimCCDecode class
```


© Copyright 2026 DEEPX All Rights Reserved.

Key Details

- **Location**: `custom_ops.py` must be in the same directory as the model YAML
- **Import timing**: Imported during `ModelBuilder.__init__()`, before YAML validation
- **Module naming**: Uses SHA1 hash of the file path to avoid module name collisions
- **Idempotent**: If the same file is already loaded, the import is skipped

```
# ModelBuilder._import_custom_ops() (simplified)
def _import_custom_ops(self) -> None:
    custom_ops_path = self.model_dir / "custom_ops.py"
    if custom_ops_path.exists():
        module_key = hashlib.sha1(str(custom_ops_path.resolve()).encode()).hexdigest()[:12]
        module_name = f"custom_ops_{self.model_dir.name}_{module_key}"
        spec = importlib.util.spec_from_file_location(module_name, custom_ops_path)
        module = importlib.util.module_from_spec(spec)
        spec.loader.exec_module(module)
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

6. API Reference

6.1 CLI Reference

DX-ModelZoo performs all operations through the `dxmz` CLI tool. It is built on Typer.

6.1.1 Installation Verification

Confirm the CLI is installed and accessible:

```
dxmz --help
```

6.1.2 Command Overview

DX-ModelZoo provides the following commands:

Command	Description
`eval`	Model evaluation
`compile`	NPU compilation
`benchmark`	Multi-model benchmarking
`list`	List available models
`info`	Model details
`create`	Create a new model YAML interactively

6.1.3 `dxmz eval`

Evaluates one or more models on their datasets and outputs metrics.

```
dxmz eval <model> [<model2> ...] --profile <profile> [options]
```


© Copyright 2026 DEEPX All Rights Reserved.

Arguments

Argument	Description	Example
`model`	One or more model names or YAML paths	`resnet50_224x224`, `./my_model.yaml`

Options

Option	Description	Default
`--profile`	Runtime profile	(required)
`--data-root`	Dataset root directory (overrides `DATA_ROOT`)	Environment variable
`--model-root`	Model file root directory (overrides `MODEL_ROOT`)	Environment variable
`--model-path`	Explicit path to the model file	`None` (falls back to `artifacts.path` in YAML)
`--save`	Save results as JSON	`False`
`--seed`	Random seed for reproducible evaluation	`42`

Examples

```
# Basic evaluation
dxmz eval resnet50_224x224 --profile onnx

# Multiple models
dxmz eval resnet50_224x224 mobilenetv3-large_224x224 --profile onnx

# Specify data/model paths
dxmz eval resnet50_224x224 --profile onnx \
  --data-root /data/datasets \
  --model-root /data/models

# Explicit model file path + save results
dxmz eval resnet50_224x224 --profile q-lite \
  --model-path /path/to/resnet50.dxnn \
  --save

# Pipeline model with stage-specific model paths
```


© Copyright 2026 DEEPX All Rights Reserved.

```
dxmz eval ocr_pipeline.yaml --profile onnx \
  --model-path "det=/path/det.onnx,rec=/path/rec.onnx"
```

6.1.4 dxmz compile

Compiles a model for the NPU.

```
dxmz compile <model> --profile <profile> [options]
```

Arguments

Argument	Description	Example
`model`	Model name or YAML path	`yolov8-n_640x640`

Options

Option	Description	Default
`--profile`	Compilation profile (`c-lite` , `q-lite` , `q-pro`)	(required)
`--output`	Output directory or file path	Auto-generated
`--model-path`	Source ONNX model path	`None` (falls back to `artifacts.path` in YAML)
`--data-root`	Calibration data path	Environment variable
`--model-root`	Model file root	Environment variable

Examples

```
# Basic compilation
dxmz compile resnet50_224x224 --profile q-lite

# Specify output directory
dxmz compile resnet50_224x224 --profile q-pro --output ./compiled/

# NPU compilation with q-lite calibration
dxmz compile yolov8-n_640x640 --profile q-lite

# Pipeline model with stage paths
```


© Copyright 2026 DEEPX All Rights Reserved.

```
dxmz compile ocr_pipeline.yaml --profile q-lite \
  --model-path "det=/path/det.onnx,rec=/path/rec.onnx"
```

6.1.5 dxmz benchmark

Benchmarks multiple models in batch with optional parallel execution.

```
dxmz benchmark --profile <profile> [options]
```

Options

Option	Description	Default
`--profile`	Runtime profile	(required)
`--models-dir`	Directory containing model YAMLs	Builtin models
`--data-root`	Dataset root	Environment variable
`--model-root`	Model file root	Environment variable
`--domain`	Domain filter (e.g., `cv`)	All
`--task`	Task filter (e.g., `image_classification`)	All
`--devices`	Comma-separated device IDs for parallel execution	`None` (sequential execution)
`--save`	Save results to JSON file	`False`

Examples

```
# Benchmark all models
dxmz benchmark --profile onnx

# Benchmark specific task
dxmz benchmark --profile onnx --task image_classification

# Parallel on multiple NPU devices
dxmz benchmark --profile q-lite --devices 0,1,2,3

# Save results
dxmz benchmark --profile q-lite --save
```


© Copyright 2026 DEEPX All Rights Reserved.

Saved output uses `result/benchmark_<profile>_<timestamp>.json`.

6.1.6 dxmz list

Displays the list of available models. By default, an interactive tree browser is launched.

```
dxmz list [options]
```

Options

Option	Description	Default
`--domain`	Domain filter	All
`--task`	Task filter	All
`--all`, `-a`	Output as plain text instead of the interactive UI	`False`

Examples

```
# Interactive model browser
dxmz list

# Full list in plain text
dxmz list --all

# Filter by specific task
dxmz list -a --task object_detection
```

Interactive Browser

Running `dxmz list` without any options launches a TUI (Text User Interface) tree browser. Use the arrow keys to navigate and press Enter to view model details.

6.1.7 dxmz info

Displays detailed information about a specific model.

```
dxmz info <model>
```


© Copyright 2026 DEEPX All Rights Reserved.

Arguments

Argument	Description	Example
`model`	Model name or YAML path	`resnet50_224x224`

Examples

```
dxmz info resnet50_224x224
```

Sample output:

```
Name: resnet50_224x224
Inputs: [{'name': 'input', 'shape': [1, 3, 224, 224], 'dtype': 'float32'}]
Profiles: ['onnx', 'c-lite', 'q-lite', 'q-pro']
Dataset: {'type': 'ILSVRC2012', 'eval_path': '${DATA_ROOT}/ILSVRC2012/val'}
```

6.1.8 `dxmz create`

Interactively create a new model YAML configuration file.

```
dxmz create
```

Launches a step-by-step wizard that asks for model name, task, input shape, preprocessing, and generates the YAML file.

6.1.9 Environment Variables

Key environment variables used by the CLI:

Variable	Description	CLI Override
`DATA_ROOT`	Dataset root directory	`--data-root`
`MODEL_ROOT`	Model file root directory	`--model-root`
`DXMZ_MODEL_URL`	Model auto-download base URL	None
`DXNN_DEVICES`	DxRuntime device specification	None

© Copyright 2026 DEEPX All Rights Reserved.

Environment Variable Precedence

When CLI options (`--data-root` , `--model-root`) are specified, they take precedence over environment variables.

6.1.10 Profile Types

The following runtime profiles are available across commands:

Profile	Target	Description
`onnx`	ONNX Runtime	CPU/GPU inference
`c-lite`	dxnn	Compile-Lite (opt_level=0, minimal optimization)
`q-lite`	dxnn	Quantize-Lite (EMA calibration)
`q-pro`	dxnn	DXNN profile with `calibration.use_q_pro: true`

© Copyright 2026 DEEPX All Rights Reserved.

6.2 ModelBuilder

`ModelBuilder` is the core class in DX-ModelZoo that parses YAML configuration files and creates all components required at runtime.

Module: `dx_modelzoo.loader.model_builder`

6.2.1 Method Map

```
graph LR
    A[YAML Config] --> MB[ModelBuilder]
    MB --> B[build_preprocessing]
    MB --> C[build_postprocessing]
    MB --> D[build_session]
    MB --> E[build_dataset]
    MB --> F[build_evaluator]
    MB --> G[run_eval]
```

6.2.2 Constructor

```
ModelBuilder(yaml_path: Union[str, Path], resolve_env: bool = True)
```

Parameter	Type	Description
`yaml_path`	`str | Path`	Path to the YAML configuration file
`resolve_env`	`bool`	Whether to resolve environment variables (default: `True`)

```
from dx_modelzoo.loader.model_builder import ModelBuilder

builder = ModelBuilder("src/dx_modelzoo/models/cv/image_classification/resnet/
resnet50_224x224.yaml")
```

6.2.3 Properties

Property	Type	Description
`name`	`str`	Model name (`config["name"]`)

© Copyright 2026 DEEPX All Rights Reserved.

Property	Type	Description
inputs	list[dict]	Input tensor definitions
config	dict	Complete parsed YAML configuration
yaml_path	Path	Path to the YAML file
model_dir	Path	Parent directory of the YAML file
is_pipeline	bool	True if YAML defines a multi-model pipeline
pipeline_stages	list[str]	Ordered list of pipeline stage names
appendix	dict	Model-specific overrides (npu_skip, label_preprocessing)

```
print(builder.name)      # "resnet50_224x224"
print(builder.inputs)    # [{"name": "input", "shape": [1,3,224,224], ...}]
```

6.2.4 Methods

get_profile

```
def get_profile(self, profile_name: str, stage: Optional[str] = None) -> dict
```

Returns the profile configuration dictionary for the given profile name.

Parameter	Type	Description
profile_name	str	Profile name
stage	str | None	Pipeline stage name (for multi-stage pipeline YAMLs)

```
profile = builder.get_profile("onnx")
# {"target": "onnx", "runtime": {"device": "gpu", "batch_size": 1}}
```


© Copyright 2026 DEEPX All Rights Reserved.

build_preprocessing

```
def build_preprocessing(
    self,
    profile_name: str,
    modal: Optional[str] = None,
    input_dtype: Optional[str] = None,
    stage: Optional[str] = None,
) -> PreprocessingPipeline
```

Builds the preprocessing pipeline for the given profile, from the YAML's `preprocessing` section.

Parameter	Type	Description
`profile_name`	`str`	Profile name (used to determine NPU mode)
`modal`	`str | None`	Modality selection for multimodal models
`input_dtype`	`str | None`	Input dtype override
`stage`	`str | None`	Pipeline stage name (for multi-stage pipeline YAMLs)

```
preprocessing = builder.build_preprocessing("onnx")
# -> PreprocessingPipeline with .steps == [Resize(...), CenterCrop(...), ...], npu_mode=False

# NPU uint8 input → arithmetic operations skipped
preprocessing_npu = builder.build_preprocessing("q-lite")
# -> PreprocessingPipeline with .steps == [Resize(...), CenterCrop(...)], npu_mode=True
```

Automatic NPU Mode Detection

`target: dxnn` enables NPU mode. Arithmetic operations are skipped when runtime input dtype is `uint8`; they remain for `float32`. Override this behavior with `appendix.npu_skip_arithmetic`.

build_postprocessing

```
def build_postprocessing(self, stage: Optional[str] = None) -> PostprocessingPipeline
```


© Copyright 2026 DEEPX All Rights Reserved.

Creates a pipeline from the `postprocessing` section of the YAML. Returns the same pipeline regardless of the profile.

Parameter	Type	Description
`stage`	`str | None`	Pipeline stage name (for multi-stage pipeline YAMLs)

```
postprocessing = builder.build_postprocessing()
# -> PostprocessingPipeline with .steps == [TopK(k=[1, 5])]
```

build_session

```
def build_session(
    self,
    profile_name: str,
    model_path: Optional[str] = None,
    device: Optional[Union[int, str, list]] = None,
    stage: Optional[str] = None,
) -> SessionBase
```

Builds the inference session for the given profile (ONNX Runtime or DxRuntime).

Parameter	Type	Description
`profile_name`	`str`	Profile name
`model_path`	`str | None`	Direct path to the model file
`device`	`int | str | list | None`	Runtime device override
`stage`	`str | None`	Pipeline stage name

```
session = builder.build_session("onnx")
# OnnxRuntimeSession("/path/to/resnet50_224x224.onnx")

session = builder.build_session("q-lite")
# DxRuntimeSession("/path/to/resnet50_224x224.dxnn")
```


© Copyright 2026 DEEPX All Rights Reserved.

Automatic Download

If the `DXMZ_MODEL_URL` environment variable is set, the model file is automatically downloaded when it is not available locally.

build_dataset

```
def build_dataset(
    self,
    data_dir: Optional[str] = None,
) -> DatasetBase
```

Builds the dataset instance from the YAML's `dataset` section.

Parameter	Type	Description
`data_dir`	`str | None`	Data directory override

```
dataset = builder.build_dataset()
# ImageNetDataset("/data/ILSVRC2012/val")

dataset = builder.build_dataset(data_dir="/custom/data/path")
```

build_evaluator

```
def build_evaluator(
    self,
    session: SessionBase,
    dataset: DatasetBase,
    profile_name: Optional[str] = None,
) -> EvaluatorBase
```

Builds the evaluator that ties together the session, dataset, and postprocessing pipeline.

Parameter	Type	Description
`session`	`SessionBase`	Inference session
`dataset`	`DatasetBase`	Dataset

© Copyright 2026 DEEPX All Rights Reserved.

Parameter	Type	Description
profile_name	str | None	Profile name (for context configuration)

```
evaluator = builder.build_evaluator(session, dataset, "onnx")
```

run_eval

```
def run_eval(
    self,
    profile_name: str,
    data_dir: Optional[str] = None,
    model_path: Optional[str] = None,
    device: Optional[Union[int, str, list]] = None,
) -> dict
```

Runs the entire evaluation pipeline in a single call. Internally invokes build_session → build_dataset → build_evaluator → eval() in sequence.

Parameter	Type	Description
profile_name	str	Profile name
data_dir	str | None	Data directory
model_path	str | None	Path to the model file
device	int | str | list | None	Runtime device override

```
results = builder.run_eval("onnx")
# {
#     "model": "resnet50_224x224",
#     "metrics": [
#         {"name": "Top-1", "metric_value": 69.76},
#         {"name": "Top-5", "metric_value": 89.08}
#     ],
#     "fps": 624.7,
#     "elapsed_time": 82,
#     "start_time": "2026-07-23 14:30:00",
#     "profile": "onnx"
# }
```

6.2.5 Examples

© Copyright 2026 DEEPX All Rights Reserved.

Simple Evaluation

```
from dx_modelzoo.loader.model_builder import ModelBuilder

builder = ModelBuilder("resnet50_224x224.yaml")
results = builder.run_eval("onnx")
print(results["metrics"])
```

Step-by-Step Execution

```
from dx_modelzoo.loader.model_builder import ModelBuilder

builder = ModelBuilder("resnet50_224x224.yaml")

# Build individual components
session = builder.build_session("onnx")
dataset = builder.build_dataset()
preprocessing = builder.build_preprocessing("onnx")
postprocessing = builder.build_postprocessing()

# Attach preprocessing to dataset
dataset.preprocessing = preprocessing

# Build & run Evaluator
evaluator = builder.build_evaluator(session, dataset, "onnx")
evaluator.set_postprocessing(postprocessing)
results = evaluator.eval()

session.close()
```

Custom Model Path

```
builder = ModelBuilder("MyModel.yaml")
results = builder.run_eval(
    profile_name="onnx",
    model_path="/custom/path/model.onnx",
    data_dir="/custom/data",
)
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

6.3 Session

Session in DX-ModelZoo is a runtime interface for performing model inference. It supports two backends: ONNX Runtime and DxRuntime (NPU).

Module: `dx_modelzoo.session`

6.3.1 SessionBase

Abstract base class that all sessions inherit from.

```
class SessionBase(ABC):
    def __init__(self, path: str, device_count: int = 1, runtime_config: Optional[RuntimeConfig] = Non
e)

    @abstractmethod
    def run(self, inputs: np.ndarray) -> list[np.ndarray]
        """Synchronous inference"""

    def run_async(self, inputs: np.ndarray, **kwargs) -> int
        """Asynchronous inference request → returns job_id"""

    def wait(self, job_id: int) -> list[np.ndarray]
        """Wait for asynchronous inference result"""

    def close(self) -> None
        """Release session resources"""

    def __enter__(self): ...
    def __exit__(self, ...): ...
```

Attributes

Attribute	Type	Description
`path`	`str`	Path to the model file
`device_count`	`int`	Number of devices to use
`runtime_config`	`RuntimeConfig | None`	Runtime options (device, batch size, async mode)

Context Manager

© Copyright 2026 DEEPX All Rights Reserved.

```
with OnnxRuntimeSession("model.onnx") as session:
    output = session.run(input_tensor)
# close() is called automatically
```

6.3.2 OnnxRuntimeSession

Inference session based on ONNX Runtime.

Module: `dx_modelzoo.session.onnx_session`

```
class OnnxRuntimeSession(SessionBase):
    def __init__(self, path: str, runtime_config: Optional[RuntimeConfig] = None)
    def run(self, inputs: np.ndarray) -> list[np.ndarray]
    def run_async(self, inputs: np.ndarray, **kwargs) -> int
    def wait(self, job_id: int) -> list[np.ndarray]
```

Features

- Automatically detects CUDA availability for GPU inference
- `run_async` is based on `ThreadPoolExecutor`
- Configurable `batch_size` (YAML `profiles.onnx.runtime.batch_size`)

Usage Example

```
from dx_modelzoo.session.onnx_session import OnnxRuntimeSession

session = OnnxRuntimeSession("resnet18.onnx")

# Synchronous inference
import numpy as np
input_tensor = np.random.randn(1, 3, 224, 224).astype(np.float32)
outputs = session.run(input_tensor)
# outputs: list[np.ndarray]

session.close()
```

6.3.3 DxRuntimeSession

Inference session based on the DeepX NPU runtime.

Module: `dx_modelzoo.session.dx_session`

```
class DxRuntimeSession(SessionBase):
    def __init__(self, path: str, runtime_config: Optional[RuntimeConfig] = None)
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    def run(self, inputs: np.ndarray) -> list[np.ndarray]
    def run_async(self, inputs: np.ndarray, **kwargs) -> int
    def wait(self, job_id: int) -> list[np.ndarray]
```

Attributes

Attribute	Type	Description
`dtype`	`list[dtype]`	Model input data types (extracted from InferenceEngine)

Features

- Uses the `dx_engine` library
- Native support for asynchronous inference
- Device selection: `runtime_config.device` or the `DXNN_DEVICES` environment variable
- Fixed `batch_size=1`

Usage Example

```
from dx_modelzoo.session.dx_session import DxRuntimeSession
from dx_modelzoo.session.runtime_config import DxnnRuntimeConfig

# Use device 0
session = DxRuntimeSession("model.dxnn", runtime_config=DxnnRuntimeConfig(device=0))

# Asynchronous inference
job_id = session.run_async(input_tensor)
outputs = session.wait(job_id)

session.close()
```


© Copyright 2026 DEEPX All Rights Reserved.

Device Selection

```
# Specify via runtime_config
session = DxRuntimeSession("model.dxnn", runtime_config=DxnnRuntimeConfig(device=0))
session = DxRuntimeSession("model.dxnn", runtime_config=DxnnRuntimeConfig(device=[0, 1]))
# Multi-device

# Specify via environment variable
# export DXNN_DEVICES=0,1
session = DxRuntimeSession("model.dxnn")
```

6.3.4 Session Factory

A factory function that automatically creates the appropriate session based on profile information.

Module: `dx_modelzoo.session.factory`

```
def create_session(
    model_or_profile: str,
    *,
    builder: Optional[object] = None,
    runtime_config: Optional[RuntimeConfig] = None,
) -> SessionBase
```

Parameter	Type	Description
`model_or_profile`	`str`	Model file path or profile name
`builder`	`object | None`	ModelBuilder instance (for profile resolution)
`runtime_config`	`RuntimeConfig | None`	Runtime options such as device, batch size, and async mode

How It Works

```
flowchart TD
    A[create_session] --> B{File extension?}
    B -->|.onnx| C[OnnxRuntimeSession]
    B -->|.dxnn| D[DxRuntimeSession]
    B -->|Profile name| E{profile.target?}
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    E -->|onnx| C
    E -->|dxnn| D
```

Automatic Download

If the `DXMZ_MODEL_URL` environment variable is set, the model file is automatically downloaded when it is not available locally:

```
export DXMZ_MODEL_URL=https://models.example.com/modelzoo
```

```
# Downloads automatically if file is not found
session = create_session("onnx", builder=builder)
# → Downloads https://models.example.com/modelzoo/onnx/resnet50_224x224.onnx
```

> **Network Required**
>
> Automatic download only works when `DXMZ_MODEL_URL` is set and the model file is not available locally. In offline environments, model files must be prepared in advance.

6.3.5 Sync vs Async Comparison

© Copyright 2026 DEEPX All Rights Reserved.

The following compares synchronous and asynchronous inference:

Sync **Async**

```
session = OnnxRuntimeSession("model.onnx")
output = session.run(input_tensor)  # Blocks until result is returned
```

- Simple and intuitive
- Runs only one inference at a time

```
session = DxRuntimeSession("model.dxnn")

# Inference requests (non-blocking)
job1 = session.run_async(input1)
job2 = session.run_async(input2)

# Wait for results
output1 = session.wait(job1)
output2 = session.wait(job2)
```

- High throughput through pipeline optimization
- Overlaps data loading and inference on the NPU

6.3.6 Runtime Configuration

Runtime options are defined in `dx_modelzoo.session.runtime_config` and are built from the YAML `profiles.<name>.runtime` block before a session is created.

`RuntimeConfig`

Base dataclass shared by all backends.

```
@dataclass
class RuntimeConfig:
    device: Any = None
    batch_size: int = 1
    async_mode: Optional[bool] = None
    start_method: str = "auto"
```

Field	Description
`device`	

© Copyright 2026 DEEPX All Rights Reserved.

Field	Description
	Device selection (`cpu` / `gpu` for ONNX, device IDs for DXNN)
`batch_size`	Evaluation batch size carried from YAML runtime config
`async_mode`	Explicit async override from YAML `runtime.asyn c`
`start_method`	Multiprocessing start method for the DataLoader: `auto` , `forkserver` , or `spawn`

`RuntimeConfig.use_async` resolves `async_mode` to the backend default when the YAML omits `runtime.async` .

OnnxRuntimeConfig

ONNX Runtime backend config.

```
@dataclass
class OnnxRuntimeConfig(RuntimeConfig):
    ASYNC_DEFAULT = False
```

- Default async mode: `False`
- Used by `OnnxRuntimeSession`

DxnnRuntimeConfig

DXNN backend config.

```
@dataclass
class DxnnRuntimeConfig(RuntimeConfig):
    ASYNC_DEFAULT = True
    buffer_count: Optional[int] = None
    use_ort: bool = True
```

Field	Description
`buffer_count`	Engine I/O buffer count from YAML `runtime.buffer_count`

© Copyright 2026 DEEPX All Rights Reserved.

Field	Description
`use_ort`	Run unsupported ops on ONNX Runtime via YAML `runtime.use_ort`

- Default async mode: `True`
- Used by `DxRuntimeSession`

YAML Mapping

```yaml
profiles:
  onnx:
    target: onnx
    runtime:
      device: gpu
      batch_size: 4
      async: false

  q-lite:
    target: dxnn
    runtime:
      device: 0
      async: true
      buffer_count: 6
      use_ort: true
```

This maps to:

- `target: onnx` → `OnnxRuntimeConfig`
- `target: dxnn` → `DxnnRuntimeConfig`
- `runtime.async` → `async_mode`
- `runtime.buffer_count` / `runtime.use_ort` → DXNN-only fields

6.3.7 Environment Variables

`DXMZ_MODEL_URL` and `DXNN_DEVICES` are relevant to session creation — see CLI Reference for the full list of environment variables.

6.3.8 Integration with ModelBuilder

Typically, sessions are not created directly but through `ModelBuilder`:

© Copyright 2026 DEEPX All Rights Reserved.

```python
from dx_modelzoo.loader.model_builder import ModelBuilder

builder = ModelBuilder("resnet50_224x224.yaml")

# Automatically creates the appropriate session for the profile
session = builder.build_session("onnx")      # → OnnxRuntimeSession
session = builder.build_session("q-lite")    # → DxRuntimeSession

# Specify model file directly
session = builder.build_session("onnx", model_path="/path/to/model.onnx")
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

6.4 DataLoader

DX-ModelZoo provides a pure Python/NumPy DataLoader for efficient batch loading and preprocessing of datasets.

Module: `dx_modelzoo.common.dataloader`

6.4.1 DatasetBase

An abstract base class that all datasets must inherit from.

```
class DatasetBase(ABC):
    def __init__(self, data_dir: str)

    @property
    def preprocessing(self) -> Optional[Callable]
        """Preprocessing pipeline getter"""

    @preprocessing.setter
    def preprocessing(self, value: Callable) -> None
        """Preprocessing pipeline setter"""

    @abstractmethod
    def __len__(self) -> int
        """Return the dataset size"""

    @abstractmethod
    def __getitem__(self, idx: int) -> tuple
        """Return a sample by index"""

    @abstractmethod
    def validate(self) -> None
        """Validate dataset files and directory structure"""
```

Attributes

Attribute	Type	Description
`data_dir`	`str`	Path to the data directory
`_preprocessing`	`Callable | None`	Preprocessing pipeline (set by the Evaluator)

© Copyright 2026 DEEPX All Rights Reserved.

`DatasetBase.preprocessing` raises `ValueError` until an Evaluator assigns a preprocessing pipeline. Custom datasets can inspect `_preprocessing` before applying it in `__getitem__`.

Usage Example

```
from dx_modelzoo.common.dataloader import DatasetBase

class MyDataset(DatasetBase):
    def __init__(self, data_dir: str):
        super().__init__(data_dir)
        self.files = sorted(Path(data_dir).glob("*.jpg"))

    def validate(self) -> None:
        # Required abstract method — without it the class cannot be instantiated
        self.ensure_exists(self.data_dir, "Place *.jpg files under this directory.")
        if not self.files:
            raise RuntimeError(f"No *.jpg files found under {self.data_dir}")

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        img = cv2.imread(str(self.files[idx]))
        if self.preprocessing:
            img = self.preprocessing(img)
        return img, idx
```

> **Warning**
>
> `__len__`, `__getitem__` **and** `validate` are abstract. Omitting `validate` raises `TypeError: Can't instantiate abstract class MyDataset without an implementation for abstract method 'validate'`.

6.4.2 DataLoader

An iterator that traverses a dataset in batches.

```
class DataLoader:
    def __init__(
        self,
        dataset: DatasetBase,
        batch_size: int = 1,
        shuffle: bool = False,
        num_workers: int = 0,
        collate_fn: Optional[Callable] = None,
        prefetch_factor: int = 2,
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    start_method: str = "auto",
)
```

Parameters

Parameter	Type	Default	Description
`dataset`	`DatasetBase`	(required)	Dataset instance
`batch_size`	`int`	`1`	Batch size
`shuffle`	`bool`	`False`	Whether to shuffle the data
`num_workers`	`int`	`0`	Number of parallel loading workers (0 = main process)
`collate_fn`	`Callable | None`	`None`	Batch collation function (None = default numpy collate)
`prefetch_factor`	`int`	`2`	Number of batches to prefetch per worker
`start_method`	`str`	`"auto"`	Multiprocessing method: `auto`, `forkserver`, or `spawn`

Methods

```
def __len__(self) -> int
    """Return the total number of batches"""

def __iter__(self) -> Iterator
    """Batch iterator"""
```

Usage Example

```
from dx_modelzoo.common.dataloader import DataLoader

loader = DataLoader(
    dataset=my_dataset,
    batch_size=4,
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    shuffle=False,
    num_workers=4,
    prefetch_factor=2,
)

print(len(loader))  # Total number of batches

for batch in loader:
    # Default collate does not stack tuple samples:
    #   batch_size == 1 -> the sample itself, e.g. (img, label)
    #   batch_size > 1  -> list of samples, e.g. [(img, label), ...]
    for img, label in batch:
        # img: np.ndarray (3, 224, 224), label: int
        results = session.run(np.expand_dims(img, 0))
```

Default collate behavior

Only `Mapping` samples are merged into batched numpy arrays. Tuple samples are returned as a Python `list` (or the single sample when `batch_size=1`), so integer labels stay `int` inside a `list` — they are **not** converted to `np.ndarray`. Pass a custom `collate_fn` if you need stacked arrays.

Single / Multi-Process Modes

Single Process (num_workers=0)	Multi-Process (num_workers>0)

```
loader = DataLoader(dataset, batch_size=4, num_workers=0)
```

- Loads data sequentially in the main process
- Useful for debugging
- May become a bottleneck for I/O-bound workloads

```
loader = DataLoader(dataset, batch_size=4, num_workers=4, prefetch_factor=2)
```

- Loads data in parallel across separate processes
- Use `prefetch_factor` to control how many batches are prepared in advance
- Maximizes throughput for large-scale datasets

6.4.3 `make_dataloader`

© Copyright 2026 DEEPX All Rights Reserved.

A factory function for conveniently creating a DataLoader.

```
def make_dataloader(
    dataset: DatasetBase,
    batch_size: int = 1,
    shuffle: bool = False,
    num_workers: int = 0,
    collate_fn: Optional[Callable] = None,
    start_method: str = "auto",
) -> DataLoader
```

```
from dx_modelzoo.common.dataloader import make_dataloader

loader = make_dataloader(dataset, batch_size=8, num_workers=4)
```

6.4.4 Batch Collation

The default collate function stacks NumPy arrays along the batch axis:

```
def _numpy_collate(batch: list[tuple]) -> tuple:
    """Merge a list of samples into a batch tensor

    [(img1, label1), (img2, label2), ...]
    → (np.stack([img1, img2, ...]), np.array([label1, label2, ...]))
    """
```

You can also provide a custom collate function:

```
def my_collate(batch):
    images = [item[0] for item in batch]
    labels = [item[1] for item in batch]
    # Handle variable-size images, etc.
    return images, labels

loader = DataLoader(dataset, batch_size=4, collate_fn=my_collate)
```

6.4.5 Differences from PyTorch

The following table summarizes the key differences:

Feature	DX ModelZoo DataLoader	PyTorch DataLoader
Dependencies	Pure Python/NumPy (DataLoader), torchvision for preprocessing fusion	Requires PyTorch

© Copyright 2026 DEEPX All Rights Reserved.

Feature	DX ModelZoo DataLoader	PyTorch DataLoader
Tensor type	`np.ndarray`	`torch.Tensor`
Default collate	`np.stack`	`torch.stack`
pin_memory	❌	☑
sampler	❌	☑
drop_last	❌	☑
Multi-process	Python `multiprocessing`	Custom implementation

Design Philosophy

DX-ModelZoo is designed for lightweight deployment in NPU environments. The DataLoader is implemented in pure Python/NumPy. PyTorch is NOT required for the DataLoader itself, though torchvision is used by the preprocessing pipeline for dx_com fusion compatibility.

6.4.6 Integration with Evaluator

In most cases, you do not need to create a DataLoader directly. `EvaluatorBase` internally calls `make_loader()` to create one automatically:

```
class EvaluatorBase:
    def make_loader(self) -> DataLoader:
        return DataLoader(
            self.dataset,
            batch_size=self.batch_size,
            shuffle=False,
            num_workers=self.workers,
        )
```

If direct usage is required (e.g., for a custom evaluation loop):

```
from dx_modelzoo.loader.model_builder import ModelBuilder
from dx_modelzoo.common.dataloader import make_dataloader

builder = ModelBuilder("resnet50_224x224.yaml")
dataset = builder.build_dataset()
preprocessing = builder.build_preprocessing("onnx")
dataset.preprocessing = preprocessing

loader = make_dataloader(dataset, batch_size=8, num_workers=4)
```


© Copyright 2026 DEEPX All Rights Reserved.

```
for images, labels in loader:
    output = session.run(images)
    # Custom processing ...
```


ⓒ Copyright 2026 DEEPX All Rights Reserved.

6.5 Datasets

API reference for registered dataset classes in DX-ModelZoo, including constructors, return formats, and resources for supported datasets.

Module: `dx_modelzoo.dataset`

6.5.1 Overview

All datasets inherit from `DatasetBase` and are registered via `DATASET_REGISTRY`. Each class is registered with `@DATASET_REGISTRY.register` — if no explicit key is provided, the class name is used as the registry key.

```
from dx_modelzoo.dataset import DATASET_REGISTRY

# Look up a dataset class by registry key
dataset_cls = DATASET_REGISTRY.get("ILSVRC2012")
dataset = dataset_cls(data_dir="/path/to/data")
```

See DataLoader for `DatasetBase` interface details.

6.5.2 Summary

The following table lists all registered dataset classes:

Registry Key	Class	Task	Module
`AFLW20003D`	`AFLW20003D`	3D Face Landmark	`dx_modelzoo.dataset.aflw20003d`
`ADE20K`	`ADE20K`	Semantic Segmentation	`dx_modelzoo.dataset.ade20k`
`BDD100K`	`BDD100K`	Panoptic Driving Perception / Vehicle Detection	`dx_modelzoo.dataset.bdd100k`
`BSD100`	`BSD100`	Super Resolution	`dx_modelzoo.dataset.bsd`
`BSD68`	`BSD68`	Image Denoising	`dx_modelzoo.dataset.bsd`

© Copyright 2026 DEEPX All Rights Reserved.

Registry Key	Class	Task	Module
CBSD68	CBSD68	Image Denoising	dx_modelzoo.dataset.bsd
CamVid	CamVid	Semantic Segmentation	dx_modelzoo.dataset.camvid
COCO	COCO	Instance Segmentation	dx_modelzoo.dataset.coco
COCOPose	COCOPose	Pose Estimation	dx_modelzoo.dataset.coco
COCOPoseTopDown	COCOPoseTopDown	Pose Estimation (Top-Down)	dx_modelzoo.dataset.coco
COCOPersonSeg	COCOPersonSeg	Person Segmentation	dx_modelzoo.dataset.coco
CelebA	CelebA	Face Attribute	dx_modelzoo.dataset.celeba
Cityscapes	Cityscapes	Semantic Segmentation	dx_modelzoo.dataset.cityscapes
DOTAv1	DOTAv1	Oriented Object Detection	dx_modelzoo.dataset.dotav1
Distinctions646	Distinctions646	Image Matting	dx_modelzoo.dataset.distinctions646
HandKeypoints	HandKeypoints	Hand Landmark	dx_modelzoo.dataset.hand_keypoints
HandKeypointsDetection	HandKeypointsDetection	Hand Detection	dx_modelzoo.dataset.hand_keypoints
HOPE	HOPE	Object Pose Estimation	dx_modelzoo.dataset.hope
HPatches	HPatches	Keypoint Detection	dx_modelzoo.dataset.hpatches
ILSVRC2012	ILSVRC2012	Image Classification	dx_modelzoo.dataset.imagenet

© Copyright 2026 DEEPX All Rights Reserved.

Registry Key	Class	Task	Module
ImageNetText	ImageNetText	Image Retrieval	dx_modelzoo.dataset.imagenet
KITTI	KITTI	3D Object Detection	dx_modelzoo.dataset.kitti
LFW	LFW	Face Recognition	dx_modelzoo.dataset.lfw
LOL	LOL	Low-Light Enhancement	dx_modelzoo.dataset.lol
Market1501	Market1501	Person Re-ID	dx_modelzoo.dataset.market1501
MVTecAD	MVTecADDataset	Anomaly Detection	dx_modelzoo.dataset.mvtec
NYUDepthv2	NYUDepthv2	Depth Estimation	dx_modelzoo.dataset.nyu
Objectron	Objectron	3D Object Detection / Pose	dx_modelzoo.dataset.objectron
OxfordIIITPet	OxfordIIITPet	Semantic Segmentation	dx_modelzoo.dataset.oxford_iiit_pet
PETA	PETA	Pedestrian Attribute	dx_modelzoo.dataset.peta
PascalVOC2007	PascalVOC2007	Object Detection	dx_modelzoo.dataset.voc
PascalVOC2012	PascalVOC2012	Semantic Segmentation	dx_modelzoo.dataset.voc
WiderFace	WiderFace	Face Detection	dx_modelzoo.dataset.widerface

6.5.3 Classification

ImageNet ILSVRC2012

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Module: `dx_modelzoo.dataset.imagenet`

Registry key: `ILSVRC2012`

Task: Image Classification

CONSTRUCTOR

```
class ILSVRC2012(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
`data_dir`	`str`	(required)

`__getitem__` RETURN FORMAT

Index	Element
`[0]`	`img`
`[1]`	`label`

RESOURCES

- **Download**: ImageNet ILSVRC2012
- **License**: Academic/Research use only

ImageNet Text Prompts

Module: `dx_modelzoo.dataset.imagenet`

Registry key: `ImageNetText`

Task: Image Retrieval

Notes: Converts ImageNet class prompts into CLIP text-transformer inputs used as text queries against a precomputed image-embedding gallery.

6.5.4 Object Detection

DOTA v1

© Copyright 2026 DEEPX All Rights Reserved.

Module: `dx_modelzoo.dataset.dotav1`

Registry key: `DOTAv1`

Task: Oriented Object Detection

CONSTRUCTOR

```
class DOTAv1(DatasetBase):
    def __init__(self, data_dir: str, split: str = 'val')
```

Parameter	Type	Default
`data_dir`	`str`	(required)
`split`	`str`	`'val'`

`__getitem__` RETURN FORMAT

Index	Element
`[0]`	`img`
`[1]`	`origin_img.shape`
`[2]`	`self.ids[idx]`

RESOURCES

- **Download**: DOTA v1
- **License**: Research use only

PascalVOC2007

Module: `dx_modelzoo.dataset.voc`

Registry key: `PascalVOC2007`

Task: Object Detection

CONSTRUCTOR

© Copyright 2026 DEEPX All Rights Reserved.

```
class PascalVOC2007(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

`__getitem__` RETURN FORMAT

Index	Element
[0]	self.preprocessing(image)
[1]	image.shape
[2]	idx

DEPENDENCIES

- faster_coco_eval

RESOURCES

- **Download**: PascalVOC2007
- **License**: "VOC2007" challenge data

BDD100K

Module: dx_modelzoo.dataset.bdd100k

Registry key: BDD100K

Task: Panoptic Driving Perception / Vehicle Detection

Notes: Provides aligned validation samples for YOLOPv2-style evaluation: vehicle boxes plus drivable-area and lane masks.

KITTI

Module: dx_modelzoo.dataset.kitti

Registry key: KITTI

© Copyright 2026 DEEPX All Rights Reserved.

Task: 3D Object Detection

Notes: Converts LiDAR point clouds into a Bird's-Eye-View (BEV) map for SFA3D-style evaluation. Ground-truth 3D boxes are read from `label_2` and projected into LiDAR coordinates.

CONSTRUCTOR

```
class KITTI(DatasetBase):
    def __init__(self, data_dir: str, split: str = "val")
```

Parameter	Type	Default
data_dir	str	(required)
split	str	"val"

__getitem__ RETURN FORMAT

Index	Element
[0]	self.preprocessing(bev)
[1]	bev.shape
[2]	sample_id

RESOURCES

- **Download**: KITTI 3D Object Detection
- **License**: Research use only

6.5.5 Segmentation

COCO 2017

Module: dx_modelzoo.dataset.coco

Registry key: COCO

Task: Instance Segmentation

© Copyright 2026 DEEPX All Rights Reserved.

CONSTRUCTOR

```
class COCO(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

__getitem__ RETURN FORMAT

Index	Element
[0]	img
[1]	origin_shape
[2]	int(self.ids[idx])

DEPENDENCIES

- faster_coco_eval

RESOURCES

- **Download**: COCO 2017
- **License**: Annotations under CC BY 4.0; images remain under their original owners' rights and Flickr Terms of Use. See COCO Terms of Use.

Cityscapes

Module: dx_modelzoo.dataset.cityscapes

Registry key: Cityscapes

Task: Semantic Segmentation

CONSTRUCTOR

© Copyright 2026 DEEPX All Rights Reserved.

```
class Cityscapes(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

`__getitem__` RETURN FORMAT

Index	Element
[0]	img
[1]	label

CLASS CONSTANTS

- num_class = 19

RESOURCES

- **Download**: Cityscapes
- **License**: Academic/Research use only

ADE20K

Module: dx_modelzoo.dataset.ade20k

Registry key: ADE20K

Task: Semantic Segmentation

Notes: SceneParse150 / ADEChallengeData2016 dataset with num_class = 150. Validation labels are remapped from dataset IDs to model indices.

Oxford-IIIT Pet

Module: dx_modelzoo.dataset.oxford_iiit_pet

Registry key: OxfordIIITPet

Task: Semantic Segmentation

© Copyright 2026 DEEPX All Rights Reserved.

CONSTRUCTOR

```
class OxfordIIITPet(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

__getitem__ RETURN FORMAT

Index	Element
[0]	img
[1]	label

CLASS CONSTANTS

- num_class = 3

RESOURCES

- **Download**: Oxford-IIIT Pet
- **License**: CC BY-SA 4.0 (free for research and commercial use)

PascalVOC2012

Module: dx_modelzoo.dataset.voc

Registry key: PascalVOC2012

Task: Semantic Segmentation

CONSTRUCTOR

```
class PascalVOC2012(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

© Copyright 2026 DEEPX All Rights Reserved.

`__getitem__` RETURN FORMAT

Index	Element
[0]	img
[1]	label

CLASS CONSTANTS

- num_class = 21

DEPENDENCIES

- faster_coco_eval

RESOURCES

- **Download**: PascalVOC2012
- **License**: "VOC2007" challenge data

CamVid

Module: dx_modelzoo.dataset.camvid

Registry key: CamVid

Task: Semantic Segmentation

CONSTRUCTOR

```
class CamVid(DatasetBase):
    def __init__(self, data_dir: str, split: str = "val")
```

Parameter	Type	Default
data_dir	str	(required)
split	str	"val"

© Copyright 2026 DEEPX All Rights Reserved.

`__getitem__` RETURN FORMAT

Index	Element
[0]	self.preprocessing(image)
[1]	label

CLASS CONSTANTS

- num_class = 11

RESOURCES

- **Download**: CamVid
- **License**: Research/non-commercial use only

6.5.6 Pose / Landmark

AFLW2000-3D

Module: dx_modelzoo.dataset.aflw20003d

Registry key: AFLW20003D

Task: 3D Face Landmark

CONSTRUCTOR

```
class AFLW20003D(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

`__getitem__` RETURN FORMAT

Index	Element
[0]	face_crop

© Copyright 2026 DEEPX All Rights Reserved.

Index	Element
[1]	gt_landmarks
[2]	bbox_size
[3]	yaw_deg
[4]	idx

CLASS CONSTANTS

- NUM_KEYPOINTS = 68

DEPENDENCIES

- scipy

RESOURCES

- **Download**: AFLW2000-3D
- **License**: Research use only

COCOPose

Module: dx_modelzoo.dataset.coco

Registry key: COCOPose

Task: Pose Estimation

CONSTRUCTOR

```
class COCOPose(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

© Copyright 2026 DEEPX All Rights Reserved.

`__getitem__` RETURN FORMAT

Index	Element
[0]	img
[1]	origin_shape
[2]	int(self.ids[idx])

DEPENDENCIES

- faster_coco_eval

RESOURCES

- **Download**: COCOPose
- **License**: Annotations under CC BY 4.0; images remain under their original owners' rights and Flickr Terms of Use. See COCO Terms of Use.

COCOPoseTopDown

Module: dx_modelzoo.dataset.coco

Registry key: COCOPoseTopDown

Task: Pose Estimation (Top-Down)

Notes: Each sample is a single-person crop from a COCO validation image, using GT bounding boxes with 25% padding. Keypoints are transformed into crop coordinates.

CONSTRUCTOR

```
class COCOPoseTopDown(DatasetBase):
    def __init__(
        self,
        data_dir: str,
        bbox_pad_ratio: float = 0.25,
        square_crop: bool = False,
        model_input_w: int = 192,
```


© Copyright 2026 DEEPX All Rights Reserved.

```
    model_input_h: int = 256,
)
```

Parameter	Type	Default
data_dir	str	(required)
bbox_pad_ratio	float	0.25
square_crop	bool	False
model_input_w	int	192
model_input_h	int	256

`__getitem__` RETURN FORMAT

Index	Element
[0]	crop (preprocessed)
[1]	gt_bbox
[2]	crop_params
[3]	kpts_crop
[4]	visibility
[5]	img_id
[6]	ann_id

DEPENDENCIES

- faster_coco_eval

RESOURCES

- **Download**: COCO 2017
- **License**: Annotations under CC BY 4.0; images remain under their original owners' rights and Flickr Terms of Use. See COCO Terms of Use.

HOPE

© Copyright 2026 DEEPX All Rights Reserved.

Module: `dx_modelzoo.dataset.hope`

Registry key: `HOPE`

Task: Object Pose Estimation

Notes: HOPE-Image validation dataset for single-object 6-DoF pose evaluation. Supports DOPE-style projected cuboid keypoints and object geometry.

Objectron

Module: `dx_modelzoo.dataset.objectron`

Registry key: `Objectron`

Task: 3D Object Detection / Object Pose Estimation

Notes: Reads native TFRecord test shards directly and returns 9-point 2D keypoints for cuboid-based evaluation.

HPatches

Module: `dx_modelzoo.dataset.hpatches`

Registry key: `HPatches`

Task: Keypoint Detection

Notes: Sequence dataset with reference/target image pairs and homographies for repeatability and matching evaluation.

Hand Keypoints

Module: `dx_modelzoo.dataset.hand_keypoints`

Registry key: `HandKeypoints`

Task: Hand Landmark

CONSTRUCTOR

ⓒ Copyright 2026 DEEPX All Rights Reserved.

```
class HandKeypoints(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

`__getitem__` RETURN FORMAT

Index	Element
[0]	crop
[1]	gt_kpts.astype(np.float32)
[2]	idx

CLASS CONSTANTS

- NUM_KEYPOINTS = 21

RESOURCES

- **License**: Custom dataset

Hand Keypoints Detection

Module: dx_modelzoo.dataset.hand_keypoints

Registry key: HandKeypointsDetection

Task: Hand Detection

Notes: Whole-image hand detection view of the hand-keypoints dataset — returns the full preprocessed image with all hand bounding boxes for AP evaluation (unlike HandKeypoints, which crops a single rotated hand).

CONSTRUCTOR

© Copyright 2026 DEEPX All Rights Reserved.

```
class HandKeypointsDetection(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

`__getitem__` RETURN FORMAT

Index	Element
[0]	image (preprocessed)
[1]	boxes (normalized xyxy)
[2]	(h, w)
[3]	img_path

RESOURCES

- **License**: Custom dataset

6.5.7 Face

CelebA

Module: dx_modelzoo.dataset.celeba

Registry key: CelebA

Task: Face Attribute

CONSTRUCTOR

```
class CelebA(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

© Copyright 2026 DEEPX All Rights Reserved.

`__getitem__` RETURN FORMAT

Index	Element
[0]	img
[1]	labels
[2]	idx

CLASS CONSTANTS

- NUM_ATTRIBUTES = 40

RESOURCES

- **Download**: CelebA
- **License**: Non-commercial research only

LFW (Labeled Faces in the Wild)

Module: dx_modelzoo.dataset.lfw

Registry key: LFW

Task: Face Recognition

CONSTRUCTOR

```
class LFW(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

`__getitem__` RETURN FORMAT

Index	Element
[0]	img1
[1]	img2

© Copyright 2026 DEEPX All Rights Reserved.

Index	Element
[2]	label

RESOURCES

- **Download**: LFW (Labeled Faces in the Wild)
- **License**: Research use only

WiderFace

Module: dx_modelzoo.dataset.widerface

Registry key: WiderFace

Task: Face Detection

CONSTRUCTOR

```
class WiderFace(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

__getitem__ RETURN FORMAT

Index	Element
[0]	self.preprocessing(img)
[1]	img.shape
[2]	file_path

DEPENDENCIES

- scipy

© Copyright 2026 DEEPX All Rights Reserved.

RESOURCES

- **Download**: WiderFace
- **License**: Non-commercial research only

6.5.8 Person

PETA (PEdesTrian Attribute)

Module: `dx_modelzoo.dataset.peta`

Registry key: `PETA`

Task: Pedestrian Attribute

CONSTRUCTOR

```
class PETA(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
`data_dir`	`str`	(required)

`__getitem__` RETURN FORMAT

Index	Element
`[0]`	`img`
`[1]`	`labels`
`[2]`	`idx`

DEPENDENCIES

- `scipy`

RESOURCES

- **Download**: PETA (PEdesTrian Attribute)
- **License**: Research use only

© Copyright 2026 DEEPX All Rights Reserved.

COCOPersonSeg

Module: `dx_modelzoo.dataset.coco`

Registry key: `COCOPersonSeg`

Task: Person Segmentation

Notes: Builds a binary person-vs-background mask (0=bg, 1=person) by rasterizing the union of all COCO `person` instance segmentations.

CONSTRUCTOR

```
class COCOPersonSeg(DatasetBase):
    def __init__(
        self,
        data_dir: str,
        inputs=None,
        ann_file: str | None = None,
        person_only: bool = True,
        **kwargs,
    )
```

Parameter	Type	Default
`data_dir`	`str`	(required)
`inputs`	`list[dict] | None`	`None`
`ann_file`	`str | None`	`None`
`person_only`	`bool`	`True`

`__getitem__` RETURN FORMAT

Index	Element
`[0]`	`image` (preprocessed)
`[1]`	`label` (binary mask)

CLASS CONSTANTS

- `num_class = 2`

© Copyright 2026 DEEPX All Rights Reserved.

DEPENDENCIES

- `faster_coco_eval`

RESOURCES

- **Download**: COCO 2017
- **License**: Annotations under CC BY 4.0; images remain under their original owners' rights and Flickr Terms of Use. See COCO Terms of Use.

6.5.9 Image Restoration

BSD100

Module: `dx_modelzoo.dataset.bsd`

Registry key: `BSD100`

Task: Super Resolution

CONSTRUCTOR

```
class BSD100(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
`data_dir`	`str`	(required)

`__getitem__` RETURN FORMAT

Index	Element
`[0]`	`lr_image`
`[1]`	`hr_image`

RESOURCES

- **Download**: BSD100
- **License**: BSD license (free for research and commercial use)

© Copyright 2026 DEEPX All Rights Reserved.

BSD68

Module: `dx_modelzoo.dataset.bsd`

Registry key: `BSD68`

Task: Image Denoising

CONSTRUCTOR

```
class BSD68(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
`data_dir`	`str`	(required)

`__getitem__` RETURN FORMAT

Index	Element
`[0]`	`self.preprocessing(origin_img)`
`[1]`	`origin_img`

RESOURCES

- **Download**: BSD68
- **License**: BSD license (free for research and commercial use)

CBSD68

Module: `dx_modelzoo.dataset.bsd`

Registry key: `CBSD68`

Task: Image Denoising

CONSTRUCTOR

© Copyright 2026 DEEPX All Rights Reserved.

```
class CBSD68(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

__getitem__ RETURN FORMAT

Index	Element
[0]	self.preprocessing(origin_img)
[1]	origin_img

RESOURCES

- **Download**: CBSD68
- **License**: BSD license (free for research and commercial use)

LOL (Low-Light)

Module: dx_modelzoo.dataset.lol

Registry key: LOL

Task: Low-Light Enhancement

CONSTRUCTOR

```
class LOL(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

© Copyright 2026 DEEPX All Rights Reserved.

`__getitem__` RETURN FORMAT

Index	Element
[0]	low_img
[1]	high_img

RESOURCES

- **Download**: LOL (Low-Light)
- **License**: Research use only

6.5.10 Depth

NYU Depth v2

Module: dx_modelzoo.dataset.nyu

Registry key: NYUDepthv2

Task: Depth Estimation

CONSTRUCTOR

```
class NYUDepthv2(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
data_dir	str	(required)

`__getitem__` RETURN FORMAT

Index	Element
[0]	self.preprocessing(rgb)
[1]	self.label_preprocessing(depth)

© Copyright 2026 DEEPX All Rights Reserved.

DEPENDENCIES

- `h5py`

RESOURCES

- **Download**: NYU Depth v2
- **License**: Research use only

6.5.11 Other

Market-1501

Module: `dx_modelzoo.dataset.market1501`

Registry key: `Market1501`

Task: Other

CONSTRUCTOR

```
class Market1501(DatasetBase):
    def __init__(self, data_dir: str)
```

Parameter	Type	Default
`data_dir`	`str`	(required)

`__getitem__` RETURN FORMAT

Index	Element
`[0]`	`img`
`[1]`	`idx`

RESOURCES

- **Download**: Market-1501
- **License**: Research use only

Distinctions-646

© Copyright 2026 DEEPX All Rights Reserved.

Module: `dx_modelzoo.dataset.distinctions646`

Registry key: `Distinctions646`

Task: Image Matting

Notes: Composites foreground/alpha pairs onto PASCAL VOC 2012 backgrounds in memory; no derived `val/` directory is required.

CONSTRUCTOR

```
class Distinctions646(DatasetBase):
    def __init__(
        self,
        data_dir: str,
        split: str = "val",
        background_dir: str | None = None,
    )
```

Parameter	Type	Default
`data_dir`	`str`	(required)
`split`	`str`	`"val"`
`background_dir`	`str | None`	`None` (resolves to the sibling PascalVOC dataset under `data_dir`)

`__getitem__` RETURN FORMAT

Index	Element
`[0]`	`self.preprocessing(composite)`
`[1]`	`alpha` (float32, normalized `[0, 1]`)

RESOURCES

- **Download**: Distinctions-646 (PaddleSeg PP-Matting) + PASCAL VOC 2012 (backgrounds)
- **License**: Follow Distinctions-646 and PASCAL VOC 2012 terms

MVTec AD

© Copyright 2026 DEEPX All Rights Reserved.

Module: `dx_modelzoo.dataset.mvtec`

Registry key: `MVTecAD`

Task: Anomaly Detection

Notes: Yields one test image per index with its image-level label (0=good, 1=anomalous), category name, and ground-truth mask path. Also exposes `train_files()` for building a PatchCore memory bank / EfficientAD statistics.

CONSTRUCTOR

```
class MVTecADDataset(DatasetBase):
    def __init__(
        self,
        data_dir: str,
        category: Optional[Union[str, List[str]]] = None,
        max_train_images: int = 0,
    )
```

Parameter	Type	Default
`data_dir`	`str`	(required)
`category`	`str | list[str] | None`	`None` (all categories)
`max_train_images`	`int`	`0` (use all)

`__getitem__` RETURN FORMAT

Index	Element
`[0]`	`img` (preprocessed)
`[1]`	`label` (0=good, 1=anomalous)
`[2]`	`category`
`[3]`	`mask_path`

RESOURCES

- **Download**: MVTec AD
- **License**: Non-commercial research/education use only

© Copyright 2026 DEEPX All Rights Reserved.

6.6 Evaluator

The Evaluator in DX-ModelZoo is an abstract class that encapsulates model evaluation logic. It performs inference through a session and computes metrics.

Module: `dx_modelzoo.evaluator`

6.6.1 EvaluatorBase

The abstract base class that all evaluators inherit from.

Constructor

```
class EvaluatorBase(ABC):
    def __init__(
        self,
        session: SessionBase,
        dataset: DatasetBase,
        workers: int = 12,
        batch_size: int = 1,
    )
```

Parameter	Type	Description
`session`	`SessionBase`	Inference session
`dataset`	`DatasetBase`	Evaluation dataset
`workers`	`int`	Number of DataLoader workers (default: 12)
`batch_size`	`int`	Batch size (default: 1)

Properties and Configuration Methods

```
@property
def postprocessing(self) -> Callable
    """Currently configured postprocessing pipeline"""

def set_preprocessing(self, preprocessing) -> None
    """Set the preprocessing pipeline on the dataset"""

def set_postprocessing(self, postprocessing) -> None
    """Set the postprocessing pipeline"""
```


© Copyright 2026 DEEPX All Rights Reserved.

```
def make_loader(self) -> DataLoader
    """Create a DataLoader instance"""
```

Context properties (set externally):

Property	Type	Description
model_name	str	Model name
dataset_name	str	Dataset name
profile_name	str	Profile name

Running Evaluation

```
def eval(self) -> dict
```

Runs evaluation and returns the results. Automatically selects Sync or Async mode based on whether the session supports asynchronous execution.

```
results = evaluator.eval()
# {
#     "model": "resnet50_224x224",
#     "metrics": [{"name": "Top-1", "metric_value": 69.76}, ...],
#     "fps": 624.7,
#     "elapsed_time": 82
# }
```

6.6.2 Abstract Methods (Must Be Implemented by Subclasses)

Concrete evaluator subclasses must implement the following 5 methods:

init_metrics

```
@abstractmethod
def init_metrics(self) -> Any
```

Initializes the metrics state. Called once before the evaluation loop begins.

```
# Example: Classification
def init_metrics(self):
    return {"correct_top1": 0, "correct_top5": 0, "total": 0}
```


© Copyright 2026 DEEPX All Rights Reserved.

`extract_inputs`

```
@abstractmethod
def extract_inputs(self, batch_data: Any) -> Any
```

Extracts model inputs from batch data. Converts the batch returned by the DataLoader into the format expected by the session.

```
# Example: Extract only images from an (images, labels) batch
def extract_inputs(self, batch_data):
    images, labels = batch_data
    return images
```

`process_batch_result`

```
@abstractmethod
def process_batch_result(
    self,
    batch_data: Any,
    output: Any,
    metrics_state: Any,
) -> Any
```

Processes the result of a single batch and updates the metrics state.

> ⚠ **`output` is already postprocessed**
>
> `EvaluatorBase` calls `postprocess_output()` **before** `process_batch_result()` and passes the result in as `output`. Calling `self.postprocessing(output)` again applies the pipeline twice, which corrupts metrics for non-idempotent steps (or raises on an unexpected input shape).
>
> The per-batch order is: `extract_inputs` → `run_inference` → `postprocess_output` → `process_batch_result`.

```
# Example: Update Top-1/Top-5 accuracy
def process_batch_result(self, batch_data, output, metrics_state):
    images, labels = batch_data
    predictions = output  # already postprocessed

    for pred, label in zip(predictions, labels):
        if pred["top1"] == label:
            metrics_state["correct_top1"] += 1
        if label in pred["top5"]:
            metrics_state["correct_top5"] += 1
```


© Copyright 2026 DEEPX All Rights Reserved.

```
        metrics_state["total"] += 1

    return metrics_state
```

compute_final_metrics

```
@abstractmethod
def compute_final_metrics(self, metrics_state: Any) -> dict
```

Computes the final metrics. Called once after the evaluation loop completes.

```
# Example
def compute_final_metrics(self, metrics_state):
    total = metrics_state["total"]
    return {
        "metrics": [
            {"name": "Top-1", "metric_value": metrics_state["correct_top1"] / total * 100},
            {"name": "Top-5", "metric_value": metrics_state["correct_top5"] / total * 100},
        ]
    }
```

format_progress_desc

```
@abstractmethod
def format_progress_desc(self, metrics_state: Any, current_fps: float) -> str
```

Generates the string to display in the progress bar.

```
# Example
def format_progress_desc(self, metrics_state, current_fps):
    total = metrics_state["total"] or 1
    acc = metrics_state["correct_top1"] / total * 100
    return f"Top-1: {acc:.2f}% | {current_fps:.1f} fps"
```

6.6.3 Sync / Async Mode

© Copyright 2026 DEEPX All Rights Reserved.

The Evaluator automatically selects the mode based on the session type:

Sync Mode **Async Mode**

```
def _eval_sync(self):
    metrics = self.init_metrics()
    for batch in dataloader:
        inputs = self.extract_inputs(batch)
        output = self.session.run(inputs)       # Synchronous inference
        metrics = self.process_batch_result(batch, output, metrics)
    return self.compute_final_metrics(metrics)
```

- **Requirement:** OnnxRuntimeSession
- Processes batches sequentially

```
def _eval_async(self):
    metrics = self.init_metrics()
    # Sliding window approach
    job_id = self.session.run_async(inputs)
    output = self.session.wait(job_id)
    metrics = self.process_batch_result(batch, output, metrics)
    return self.compute_final_metrics(metrics)
```

- **Requirement:** DxRuntimeSession (NPU)
- Maximizes throughput through pipeline optimization
- Maximum in-flight inference requests: `min(device_count × 4, total_samples)`
- Loader prefetch depth: at most `workers × 4`

6.6.4 Built-in Evaluators

The following evaluators are available out of the box:

Registry Name	Task	Key Metrics
`image_classification`	Classification	Top-1, Top-5
`anomaly_detection`	Anomaly Detection	AUROC, F1
`object_detection`	Object Detection	mAP
`3d_object_detection`	3D Object Detection	3D AP
`oriented_object_detection`	Oriented Object Detection	mAP

© Copyright 2026 DEEPX All Rights Reserved.

Registry Name	Task	Key Metrics
instance_segmentation	Instance Segmentation	mAP
zero_shot_instance_segmentation	Zero-shot Instance Segmentation	mAP
face_detection	Face Detection	AP / FROC
face_landmark	Face Landmark	NME
face_recognition	Face Recognition	Accuracy
face_attribute	Face Attribute	Accuracy
hand_detection	Hand Detection	AP
hand_landmark	Hand Landmark	NME
depth_estimation	Depth Estimation	δ<1.25, RMSE
image_denoising	Image Denoising	PSNR
low_light_enhancement	Low-light Enhancement	PSNR, SSIM
pose_estimation	Pose Estimation	AP, AR
pose_estimation_topdown	Top-down Pose Estimation	AP, AR
person_attribute	Person Attribute	mA
person_segmentation	Person Segmentation	mIoU
person_reid	Person Re-identification	Rank-1, mAP
semantic_segmentation	Semantic Segmentation	mIoU
panoptic_driving_perception	Panoptic Driving Perception	Detection + segmentation metrics
visual_place_recognition	Visual Place Recognition	Retrieval metrics
image_retrieval	Image Retrieval	Top-1, Top-5 retrieval accuracy
image_matting	Image Matting	SAD, MSE
keypoint_detection	Keypoint Detection	Matching / repeatability metrics

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Registry Name	Task	Key Metrics
`mainbody_detection`	Mainbody Detection	AP
`super_resolution`	Super Resolution	PSNR
`zero_shot_image_classification`	Zero-shot Image Classification	Top-1
`object_pose_estimation`	Object Pose Estimation	ADD / ADD-S / PnP metrics

6.6.5 Design Principles

The Evaluator architecture follows a strict separation of concerns:

Layer	Responsibility	Example
Postprocessing (`custom_ops.py`)	Convert raw model output → standardized format	Decode heatmap → keypoints
Evaluator	Consume standardized data → compute metrics	Keypoints → COCO AP

```
graph LR
    A[Model Output] -->|Postprocessor| B[Standardized Data]
    B -->|Evaluator| C[Metrics]
```

Rules: - Evaluators must **not** contain model-specific decoding logic. - Model-specific transforms belong in `custom_ops.py` (registered with `POSTPROCESSING_REGISTRY`). - Common operations (NMS, TopK) remain in the shared `postprocessing/` module.

Standardized Data Formats

Task	Postprocessor Output
Feature Extraction	`(keypoints [N,2], descriptors [N,D])`
Pose Estimation (Top-Down)	`(keypoints [17,2], scores [17])`
Text Detection	`List[np.ndarray]` — polygons `[4,2]`
Text Recognition	`str` — decoded text
Face Detection (BlazeFace)	`np.ndarray [N, 5]` (x1, y1, x2, y2, score)

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Task	Postprocessor Output
Instance Segmentation	`np.ndarray [N, H, W]` binary masks
Anomaly Detection	`float` — anomaly score

6.6.6 Writing a Custom Evaluator

Register a custom evaluator with `EVALUATOR_REGISTRY` as follows:

```python
from dx_modelzoo.evaluator import EVALUATOR_REGISTRY, EvaluatorBase

@EVALUATOR_REGISTRY.register("my_evaluator")
class MyEvaluator(EvaluatorBase):
    def init_metrics(self):
        return {"total_error": 0.0, "count": 0}

    def extract_inputs(self, batch_data):
        inputs, targets = batch_data
        return inputs

    def process_batch_result(self, batch_data, output, metrics_state):
        _, targets = batch_data
        predictions = output  # already postprocessed by the base evaluator
        error = np.mean(np.abs(predictions - targets))
        metrics_state["total_error"] += error
        metrics_state["count"] += 1
        return metrics_state

    def compute_final_metrics(self, metrics_state):
        mae = metrics_state["total_error"] / metrics_state["count"]
        return {"metrics": [{"name": "MAE", "metric_value": mae}]}

    def format_progress_desc(self, metrics_state, current_fps):
        count = metrics_state["count"] or 1
        mae = metrics_state["total_error"] / count
        return f"MAE: {mae:.4f} | {current_fps:.1f} fps"
```

```yaml
evaluator:
  type: my_evaluator
```


© Copyright 2026 DEEPX All Rights Reserved.

Keep Evaluators Pure

If your evaluator needs to decode model-specific output formats, put that logic in a `custom_ops.py` file in the model directory instead. The evaluator should only consume the decoded result and compute metrics.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

6.7 Preprocessing

The preprocessing pipeline in DX-ModelZoo transforms input data into the format required by the model.

Module: `dx_modelzoo.preprocessing`

6.7.1 PreprocessingPipeline

A pipeline that sequentially executes preprocessing operations defined in YAML.

```
class PreprocessingPipeline(Compose):
    def __init__(
        self,
        steps_config: list[dict] | dict[str, list[dict]],
        npu_mode: bool = False,
    )
    def __call__(self, img: np.ndarray) -> np.ndarray
```

Parameter	Type	Description
`steps_config`	`list[dict]` | `dict[str, list[dict]]`	Step **configuration** dicts (`{"type": ..., **params}`), i.e. the YAML `preprocessing:` block. Use a dict keyed by input name for multi-input models
`npu_mode`	`bool`	NPU mode (skips `NPU_SKIP_DEFAULT` operations when True)

Attribute	Type	Description
`steps`	`list`	Instantiated preprocessing operation objects built from `steps_config`

© Copyright 2026 DEEPX All Rights Reserved.

> **Warning**
>
> The constructor argument is `steps_config` and takes **config dicts**, not operation instances. `steps` is the built result and is read-only output — passing objects (or using `steps=`) raises `TypeError` / `AttributeError`.

```
from dx_modelzoo.preprocessing import PreprocessingPipeline

pipeline = PreprocessingPipeline(
    steps_config=[
        {"type": "resize", "size": [256, 256]},
        {"type": "centercrop", "size": [224, 224]},
    ]
)
```

Usage Example

```
from dx_modelzoo.loader.model_builder import ModelBuilder

builder = ModelBuilder("resnet50_224x224.yaml")
pipeline = builder.build_preprocessing("onnx")

import cv2
img = cv2.imread("test.jpg")      # (H, W, 3) BGR uint8
result = pipeline(img)            # (1, 3, 224, 224) float32
```

6.7.2 NPU Skip Behavior

NPU_SKIP_DEFAULT

```
NPU_SKIP_DEFAULT = {"div", "normalize", "transpose", "expanddim", "mul", "add", "subtract"}
```

When `npu_mode=True`, operations whose types are included in this set are skipped for `uint8` runtime input. For `float32` input, arithmetic operations remain unless `appendix.npu_skip_arithmetic` overrides the behavior.

> **NPU Mode Caution**
>
> For `target: dxnn`, arithmetic operations are skipped when runtime input dtype is `uint8`; they remain for `float32` input. Set `appendix.npu_skip_arithmetic` to override this behavior. Spatial transforms such as `resize`, `centercrop`, and `convertcolor` still run on the CPU.

© Copyright 2026 DEEPX All Rights Reserved.

NPU Skip Summary

Type	NPU Skip	Reason
`resize`	✗	Spatial transform — requires CPU
`centercrop`	✗	Spatial transform — requires CPU
`convertcolor`	✗	Color conversion — requires CPU
`div`	`uint8` only	Handled internally by NPU for uint8 input
`normalize`	`uint8` only	Handled internally by NPU for uint8 input
`transpose`	`uint8` only	Layout conversion handled internally for uint8 input
`expanddim`	`uint8` only	Batch handling handled internally for uint8 input
`mul`	`uint8` only	Handled internally by NPU for uint8 input
`add`	`uint8` only	Handled internally by NPU for uint8 input
`subtract`	`uint8` only	Handled internally by NPU for uint8 input

6.7.3 Built-in Types

Type	Summary
`resize`	Resize image
`centercrop`	Center crop
`convertcolor`	Color conversion
`div`	Divide by scalar

ⓒ Copyright 2026 DEEPX All Rights Reserved.

Type	Summary
`normalize`	Mean/std normalization
`transpose`	Axis permutation
`expanddim`	Insert dimension
`mul`	Multiply by scalar
`add`	Add scalar
`subtract`	Subtract scalar
`bgr_to_y_channel`	Convert BGR to Y channel
`bgr_to_y_channel_uint8`	Convert BGR to uint8 Y channel
`totensor`	HWC uint8 `[0,255]` → CHW float32 `[0,1]`

`resize`

Resize an image.

```yaml
- type: resize
  size: [256, 256]        # [H, W]
  mode: default           # torchvision | default | pad | pycls
  interpolation: BILINEAR   # BILINEAR | BICUBIC | NEAREST | AREA
```

Parameter	Type	Default	Description
`size`	`list[int]`	(required)	Target size [H, W]
`mode`	`str`	`"default"`	Resize mode
`interpolation`	`str`	`"BILINEAR"`	Interpolation method

Mode	Backend	Behavior
`default`	OpenCV (`cv2.resize`)	Plain OpenCV resize to the target size
`torchvision`	PIL / torchvision	Matches `torchvision.transforms.Resize` semantics

© Copyright 2026 DEEPX All Rights Reserved.

Mode	Backend	Behavior
`pad`	OpenCV	Aspect-ratio preserving resize + padding (letterbox); tune with `align_side`, `scale_method`, `pad_location`, `pad_value`
`pycls`	OpenCV	pycls-style resize convention

> **Note**
>
> There is no `opencv` or `letterbox` mode. Use `default` for OpenCV resize and `pad` for letterbox-style resize. Any other value raises `ValueError: Invalid mode: ...`.

centercrop

Center crop.

```
- type: centercrop
  height: 224
  width: 224
```

Parameter	Type	Description
`height`	`int`	Crop height
`width`	`int`	Crop width

convertcolor

Color space conversion.

```
- type: convertcolor
  form: BGR2RGB
```

Parameter	Type	Description
`form`	`str`	Conversion format (`BGR2RGB`, `RGB2BGR`, `BGR2GRAY`, etc.)

div

© Copyright 2026 DEEPX All Rights Reserved.

Division (scaling).

```
- type: div
  x: 255
```

Parameter	Type	Description
x	float	Divisor value

Automatic NPU Skip

div is included in NPU_SKIP_DEFAULT and is skipped in NPU mode.

normalize

Mean/standard deviation normalization: (img - mean) / std

```
- type: normalize
  mean: [0.485, 0.456, 0.406]
  std: [0.229, 0.224, 0.225]
```

Parameter	Type	Description
mean	list[float]	Per-channel mean
std	list[float]	Per-channel standard deviation

transpose

Axis transposition (e.g., HWC → CHW).

```
- type: transpose
  axis: [2, 0, 1]
```

Parameter	Type	Description
axis	list[int]	Axis order

expanddim

© Copyright 2026 DEEPX All Rights Reserved.

Add a dimension (e.g., batch dimension).

```
- type: expanddim
  axis: 0
```

Parameter	Type	Description
axis	int	Axis position to insert

mul

Multiplication.

```
- type: mul
  x: 2.0
```

Parameter	Type	Description
x	float	Multiplier value

add

Addition.

```
- type: add
  x: 1.0
```

Parameter	Type	Description
x	float	Value to add

subtract

Subtraction.

```
- type: subtract
  x: 128.0
```

Parameter	Type	Description
x	float	Value to subtract

© Copyright 2026 DEEPX All Rights Reserved.

bgr_to_y_channel

Convert a BGR image to the Y channel (luminance). Primarily used for super-resolution models.

```
- type: bgr_to_y_channel
```

No parameters.

bgr_to_y_channel_uint8

BGR to Y channel conversion (uint8 output). For NPU models.

```
- type: bgr_to_y_channel_uint8
```

No parameters.

totensor

Convert HWC `uint8` input in `[0, 255]` to CHW `float32` in `[0, 1]`.

```
- type: totensor
```

No parameters.

Equivalent to `div(x=255)` + `transpose`, but registered as a single preprocessing step.

6.7.4 Custom Preprocessing

```python
from dx_modelzoo.preprocessing import PREPROCESSING_REGISTRY

@PREPROCESSING_REGISTRY.register("my_custom_op")
class MyCustomOp:
    def __init__(self, param: int = 10):
        self.param = param

    def __call__(self, img: np.ndarray) -> np.ndarray:
        # Custom logic
        return img
```

```yaml
preprocessing:
  - type: my_custom_op
    param: 20
```


© Copyright 2026 DEEPX All Rights Reserved.

NPU Skip for Custom Operations

Custom operations are not skipped in NPU mode by default, as they are not included in `NPU_SKIP_DEFAULT`. If your custom operation is an arithmetic operation that should be skipped on the NPU, you must add it to the set manually.

For detailed usage, refer to the Custom Models guide.

ⓒ Copyright 2026 DEEPX All Rights Reserved.

6.8 Postprocessing

The postprocessing pipeline in DX-ModelZoo transforms raw model outputs into structured, task-specific results.

Module: `dx_modelzoo.postprocessing`

6.8.1 PostprocessingPipeline

Sequentially executes postprocessing operations defined in a YAML configuration.

```
class PostprocessingPipeline:
    def __init__(self, steps_config: list[dict])
    def __call__(self, outputs, **kwargs) -> Any
```

Parameter	Type	Description
`steps_config`	`list[dict]`	Step **configuration** dicts (`{"type": ..., **params}`), i.e. the YAML `postprocessing:` block

Attribute	Type	Description
`steps`	`list`	Instantiated postprocessing operation objects built from `steps_config`

> **Warning**
>
> The constructor argument is `steps_config` and takes **config dicts**, not operation instances. Using `steps=` raises `TypeError`; passing built objects raises `AttributeError: 'TopK' object has no attribute 'get'`.

Context keyword arguments (`runtime_target` , and for detection models `origin_hw` / `input_hw`) are forwarded to every step — see Adding Custom Postprocessing.

```
from dx_modelzoo.postprocessing import PostprocessingPipeline

postprocessing = PostprocessingPipeline(steps_config=[{"type": "topk", "k": 5}])
```


© Copyright 2026 DEEPX All Rights Reserved.

Usage Example

```
from dx_modelzoo.loader.model_builder import ModelBuilder

builder = ModelBuilder("resnet50_224x224.yaml")
postprocessing = builder.build_postprocessing()

# Apply to model outputs
outputs = session.run(input_tensor)
result = postprocessing(outputs)
```

No NPU Skip

Unlike preprocessing, the postprocessing pipeline does not have NPU skip logic. All postprocessing operations are always executed.

6.8.2 Built-in Postprocessing Types

The shared built-in types below are the only operations registered directly in `dx_modelzoo.postprocessing.POSTPROCESSING_REGISTRY`: `identity`, `topk`, `nms`, and `segmentation_argmax`.

identity

Returns the input as-is without any transformation. Used for models that do not require postprocessing.

```
postprocessing:
  - type: identity
```

No parameters.

topk

Extracts the top-k predictions from a classification model.

© Copyright 2026 DEEPX All Rights Reserved.

```yaml
postprocessing:
  - type: topk
    k: [1, 5]
```

Parameter	Type	Description
k	list[int]	Number of top predictions to extract (e.g., [1, 5] → Top-1, Top-5)
skip_background	bool	Drop index 0 (background class) before ranking (default: False)

Input: list[np.ndarray] — Model output (logits/probabilities) **Output:** Top-k class indices, sorted by confidence descending

nms

Applies Non-Maximum Suppression to remove duplicate detections.

```yaml
postprocessing:
  - type: nms
    conf_thres: 0.001
    iou_thres: 0.7
    max_output_boxes: 300
```

Parameter	Type	Description
conf_thres	float	Confidence threshold to keep a detection (default: 0.001)
iou_thres	float	IoU threshold above which overlapping boxes are suppressed (default: 0.7)
max_output_boxes	int	Maximum number of boxes to keep per image (default: 300)
pad_resize	bool	Whether to rescale coordinates for letterbox/pad-resize preprocessing (default: True)

© Copyright 2026 DEEPX All Rights Reserved.

Input: `dict` — `{"boxes": [N,4] xyxy, "scores": [N], "class_ids": [N]}` from a model-specific decode step **Output:** `np.ndarray` of shape `[M, 6]` — `(x1, y1, x2, y2, score, class_id)`

Task: Object Detection

segmentation_argmax

Applies argmax along the channel axis to the output of a segmentation model.

```yaml
postprocessing:
  - type: segmentation_argmax
    layout: nchw
    target_size: [512, 512]
```

Parameter	Type	Description
`layout`	`str`	Input layout, `nchw` or `nhwc` (default: `nchw`)
`target_size`	`list[int] | None`	Optional `[H, W]` to resize the output via nearest interpolation (default: `None`)

Input: Logits of shape `(1, num_classes, H, W)` **Output:** Class index map of shape `(H, W)`

Task: Semantic Segmentation

6.8.3 Postprocessing Guide by Task

The following table maps common tasks to their typical postprocessing:

Task	Recommended Postprocessing	YAML Example
Image Classification	`topk`	`type: topk, k: [1, 5]`
Object Detection	model-specific decode + shared `nms`	`type: yolov8_decode` → `type: nms, conf_thres: 0.001, iou_thres: 0.7`
Semantic Segmentation	`segmentation_argmax`	`type: segmentation_argmax`
Instance Segmentation	model-specific decode	`type: yolact_decode`

© Copyright 2026 DEEPX All Rights Reserved.

Task	Recommended Postprocessing	YAML Example
Face Detection	model-specific decode	`type: retinaface_decode`
Pose Estimation	model-specific decode	`type: rtmpose_simcc_decode`
Super Resolution	`identity`	`type: identity`
Depth Estimation	`identity`	`type: identity`

6.8.4 Adding Custom Postprocessing

Register a custom postprocessing operation the same way as preprocessing:

```python
from dx_modelzoo.postprocessing import POSTPROCESSING_REGISTRY

@POSTPROCESSING_REGISTRY.register("my_postprocess")
class MyPostprocess:
    """Custom postprocessing operation"""

    def __init__(self, threshold: float = 0.5):
        self.threshold = threshold

    def __call__(self, outputs: list[np.ndarray], **kwargs) -> dict:
        # outputs: list of raw model outputs
        # kwargs carries pipeline context (see table below) — always accept **kwargs
        result = outputs[0]
        mask = result > self.threshold
        return {"predictions": result[mask]}
```

> ⚠ **`__call__` must accept `**kwargs`**
>
> `PostprocessingPipeline.__call__(outputs, **kwargs)` forwards its context to every step. A step defined as `__call__(self, outputs)` raises `TypeError: __call__() got an unexpected keyword argument 'runtime_target'`.

Keyword	Provided by	Meaning
`runtime_target`	all evaluators	`"onnx"` / `"npu"` — the active runtime
`origin_hw`	detection-style evaluators	`(H, W)` of the original image

© Copyright 2026 DEEPX All Rights Reserved.

Keyword	Provided by	Meaning
`input_hw`	detection-style evaluators	`(H, W)` of the model input

Steps producing coordinates (boxes, keypoints) must map model-input pixel space back to original image coordinates using `origin_hw` / `input_hw`:

```python
from dx_modelzoo.postprocessing.coord_scaler import unpad_and_scale

@POSTPROCESSING_REGISTRY.register("my_box_postprocess")
class MyBoxPostprocess:
    def __call__(self, outputs, origin_hw=None, input_hw=None, **kwargs):
        boxes = outputs[0]  # [M, 6] (x1, y1, x2, y2, score, class_id) in input space
        if origin_hw is not None and input_hw is not None:
            boxes[:, :4] = unpad_and_scale(boxes[:, :4], input_hw, origin_hw)
        return boxes
```

```yaml
postprocessing:
  - type: my_postprocess
    threshold: 0.7
```

Chaining Multiple Postprocessing Steps

Multiple postprocessing operations can be chained sequentially:

```yaml
postprocessing:
  - type: yolov8_decode      # Step 1: model-specific decode
  - type: nms                # Step 2: shared NMS
    conf_thres: 0.001
    iou_thres: 0.7
```

> **Postprocessing Order**
>
> Postprocessing operations are executed in the order they are defined in the YAML configuration. The output of each step becomes the input to the next, so ordering matters.

For detailed usage, refer to the Custom Models guide.

6.8.5 Model-Specific Postprocessing (`custom_ops.py`)

© Copyright 2026 DEEPX All Rights Reserved.

For model-specific decoding logic that doesn't belong in the shared postprocessing module, use a `custom_ops.py` file in the model's directory. Many model YAMLs reference names registered there instead of shared built-ins.

Auto-Discovery Mechanism

When `ModelBuilder` loads a model YAML, it automatically imports `custom_ops.py` from the same directory:

```
models/custom/MyModel/
├── MyModel_v1.yaml
└── custom_ops.py        ← auto-imported
```

The import happens before validation, so any `@POSTPROCESSING_REGISTRY.register()` calls in `custom_ops.py` are available for the YAML's `postprocessing` section.

Example: Model-Specific Decoder

models/cv/pose_estimation/RTMPose/custom_ops.py

```python
from dx_modelzoo.postprocessing import POSTPROCESSING_REGISTRY
import numpy as np

@POSTPROCESSING_REGISTRY.register("rtmpose_simcc_decode")
class RTMPoseSimCCDecode:
    """Decode SimCC heatmaps to COCO 17-keypoint format."""

    def __init__(self, split_ratio: float = 2.0, **kwargs):
        self.split_ratio = split_ratio

    def __call__(self, outputs, **kwargs):
        simcc_x, simcc_y = outputs[0], outputs[1]
        x_idx = simcc_x[0].argmax(axis=-1).astype(np.float32)
        y_idx = simcc_y[0].argmax(axis=-1).astype(np.float32)
        scores = (simcc_x[0].max(axis=-1) + simcc_y[0].max(axis=-1)) / 2.0
        kpts = np.stack([x_idx / self.split_ratio, y_idx / self.split_ratio], axis=-1)
        return kpts, scores
```

RTMPose_T.yaml (postprocessing section)

```yaml
postprocessing:
  - type: rtmpose_simcc_decode
    split_ratio: 2.0
```

Built-in Model-Specific Types

© Copyright 2026 DEEPX All Rights Reserved.

These types are registered via `custom_ops.py` in their respective model directories:

Type	Model	Input → Output
`superpoint_decode`	SuperPoint	65-ch heatmap → `(keypoints, descriptors)`
`rtmpose_simcc_decode`	RTMPose	SimCC bins → `(keypoints [17,2], scores [17])`
`mediapipe_pose_decode`	MediaPipePose	Landmarks → `(keypoints [17,2], scores [17])`
`blazeface_decode`	BlazeFace	Raw boxes → `[N, 5]` (xyxy + score)

When NOT to Use `POSTPROCESSING_REGISTRY`

Not all model-specific code belongs in the postprocessing registry:

- **Init-time utilities** (run once at startup, not per-sample): Place in a utility file.
- Example: `clip_utils.py` for building text embeddings at evaluation init time.
- **Common operations** (used across many models): Keep in `src/dx_modelzoo/postprocessing/`.
- Example: NMS, TopK, coordinate scaling.

ⓒ Copyright 2026 DEEPX All Rights Reserved.