DXNN Compiler (DX-COM) User Manual

August 2025 - Version 2.0.0

DEEPX.ai

ⓒ *Copyright 2025 DEEPX All Rights Reserved.*

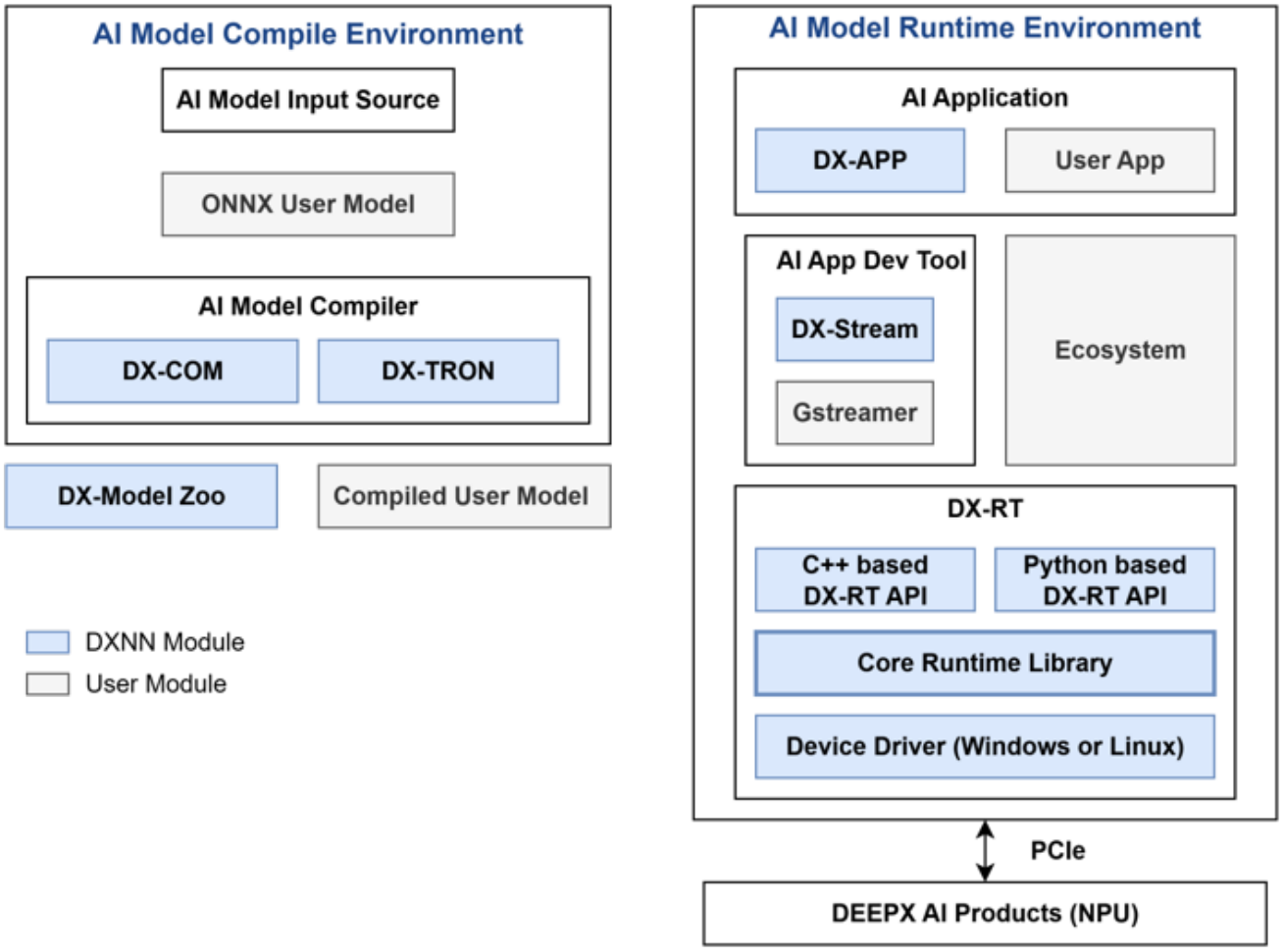
Table of contents

ⓒ Copyright 2025 DEEPX All Rights Reserved.

1. DXNN SDK Overview

This chapter provides an overview of the DEEPX SDK architecture and explains each core component and its role in the AI development workflow.

1.1 DEEPX SDK Architecture



DEEPX SDK is an all-in-one software development platform that streamlines the process of compiling, optimizing, simulating, and deploying AI inference applications on DEEPX NPUs (Neural Processing Units). It provides a complete toolchain, from AI model creation to runtime deployment, optimized for edge and embedded systems, enabling developers to build high-performance AI applications with minimal effort.

DX-COM is the compiler in the DEEPX SDK that converts a pre-trained ONNX model and its associated configuration JSON file into a hardware-optimized .dxnn binary for DEEPX NPUs. The ONNX file contains

ⓒ Copyright 2025 DEEPX All Rights Reserved.

the model structure and weights, while the JSON file defines pre/post-processing settings and compilation parameters. DX-COM provides a fully compiled .dxnn file, optimized for low-latency and high-efficient inference on DEEPX NPU.

DX-RT is the runtime software responsible for executing ,dxnn models on DEEPX NPU hardware. DX-RT directly interacts with the DEEPX NPU through firmware and device drivers, using PCIe interface for high-speed data transfer between the host and the NPU, and provides C/C++ and Python APIs for application-level inference control. DX-RT offers a complete runtime environment, including model loading, I/O buffer management, inference execution, and real-time hardware monitoring.

DX ModelZoo is a curated collection of pre-trained neural network models optimized for DEEPX NPU, designed to simplify AI development for DEEPX users. It includes pre-trained ONNX models, configuration JSON files, and pre-compiled DXNN binaries, allowing developers to rapidly test and deploy applications. DX ModelZoo also provides benchmark tools for comparing the performance of quantized INT8 models on DEEPX NPU with full-precision FP32 models on CPU or GPU.

DX-Stream is a custom GStreamer plugin that enables real-time streaming data integration into AI inference applications on DEEPX NPU. It provides a modular pipeline framework with configurable elements for preprocessing, inference, and postprocessing, tailored to vision AI work. DX-Stream allows developers to build flexible, high-performance applications for use cases such as video analytics, smart cameras, and edge AI systems.

DX-APP is a sample application that demonstrates how to run compiled models on actual DEEPX NPU using DX-RT. It includes ready-to-use code for common vision tasks such as object detection, face recognition, and image classification. DX-APP helps developers quickly set up the runtime environment and serves as a template for building and customizing their own AI applications.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

2. Installation and Configuration

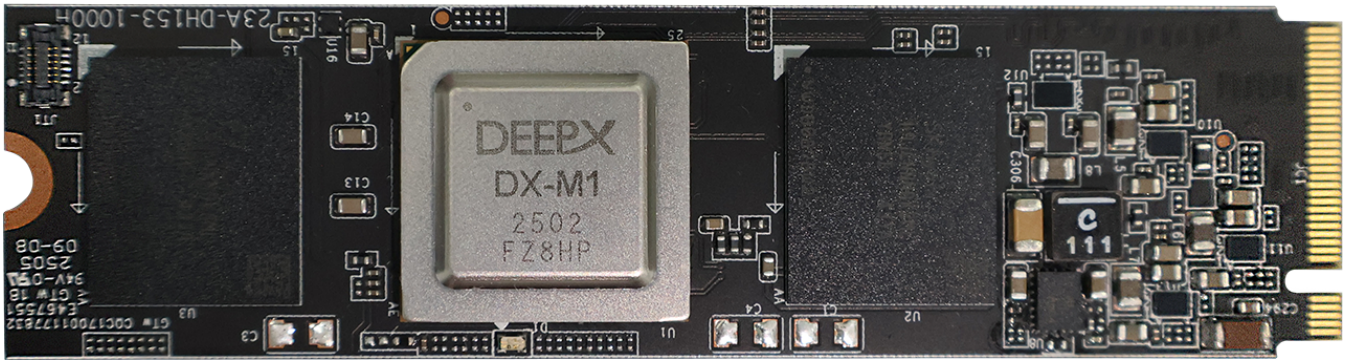
2.1 System Requirements of DX-COM

This section describes the hardware and software requirements for running **DX-COM**.

Hardware and Software Requirements

- **CPU:** amd64(x86_64)
- P.S. aarch64(arm64) is **NOT** supported
- **RAM:** ≥ 16 GB
- **Storage:** ≥ 8 GB available disk space
- **OS:** Ubuntu 20.04 / 22.04 / 24.04 (x64)
- P.S. Ubuntu 18.04 OS is **NOT** supported
- **LDD:** ≥ 2.28

Note. To check your LDD version, run `ldd --version` in the terminal.



ⓒ Copyright 2025 DEEPX All Rights Reserved.

2.2 Installation of DX-COM

This section provides instructions for installing required libraries and setting up **DX-COM** on supported Ubuntu distributions.

Install the Required Libraries

Before installing **DX-COM**, ensure the following libraries are installed.

- `libgl1-mesa-glx` : Provides OpenGL runtime support for graphical operations
- `dlibglib2.0-0` : Core utility library used by many GNOME and GTK applications

- Run the following command to install the required libraries.

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

Install DX-COM

DX-COM supports the Target OS of Ubuntu 20.04, Ubuntu 22.04, and Ubuntu 24.04.
- P.S. Ubuntu 18.04 is **NOT** supported

- After downloading the compiler archive, extract it using the following command.

```
tar xfz dx_com_M1_vx.x.x.tar.gz
```

After extraction, the directory `dx_com/` will contain the compiler executables, sample ONNX models, JSON configuration files, a sample `Makefile` for compilation.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

2.3 Source File Structure of DX-COM

The source structure of **DX-COM** is organized as follows.

```
dx_com
├── calibration_dataset       # Dataset used to optimize model accuracy <br>
├── dx_com
│ ├── cv2/                   # Third party shared libraries (e.g., OpenCV) <br>
│ ├── google/                # Third party shared libraries (e.g., protobuf) <br>
│ ├── numpy/                 # Third party  shared libraries (e.g., NumPy) <br>
│ ├── ...                    # Other dependencies <br>
│ └── dx_com                 # Core compiler implementation <br>
├── sample
│ ├── MobilenetV1.json       # Sample configuration file <br>
│ └── MobilenetV1.onnx       # Sample ONNX model
└── Makefile                  # Build script for compiling the sample model  <br>
```

`calibration_dataset`

This directory contains the calibration dataset used for compiling the included sample model as an example. It is used to calibrate the model's input range for quantization purposes. If the calibration dataset does not reflect the training or field data, it may significantly degrade model accuracy.

`dx_com`

This directory contains executable files and shared libraries used to generate NPU command sets from ONNX models. It includes core compiler logic and third-party dependencies such as OpenCV, NumPY, and Protobuf.

`sample`

This folder provides example files to demonstrate how to compile an ONNX model using **DX-COM**.

- **ONNX File** (`.onnx`): An ONNX model used as input for NPU command generation.
- **Config File** (`.json`): A JSON configuration file that includes parameters such as quantization methods, image processing settings, and other options. Refer to **Chapter 2.5. JSON File Configuration**.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

2.4 ONNX File Configuration

This section describes how to convert a PyTorch model to the ONNX format using the `torch.onnx.export()` function.

pytorch to ONNX Converison Example

You can export a PyTorch model to ONNX format as follows.

Example

```
import torch
import torch.nn as nn

# 1. Define or load the PyTorch model
class SimpleModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Linear(10, 5)    # Input 10, Output 5

  def forward(self, x):
    return self.linear(x)

model = SimpleModel()
model.eval()                # Set to inference mode (Affect Dropout, BatchNorm, etc.)

# 2. Create a dummy input tensor with the same shape and type as the model input
# This is used to trace the model's computational graph, not for the actual inference
batch_size = 1              # batch size must be 1
dummy_input = torch.randn(batch_size, 10)  # Create input matching the model's input
shape (Batch, Features)

# 3. Export the model to ONNX format
onnx_file_path = "simple_model.onnx"

torch.onnx.export(
  model,                  # PyTorch model object to export
  dummy_input,            # Dummy input used for tracing (tuple is possible)
  onnx_file_path,         # Output ONNX file path
  export_params=True,     # If True, saves model parameter (weight) into the ONNX file
  opset_version=11,       # ONNX opset version (11 or higher recommended)
  input_names=['input'],  # Name of the ONNX model input tensor
  output_names=['output']  # Name of the ONNX model output tensor
)
```


ⓒ Copyright 2025 DEEPX All Rights Reserved.

Key Parameter of `torch.onnx.export()`

- `model` : PyTorch model object to export
- `dummy_input` : Input values to model's `forward()` method
- `onnx_file_path` : Output ONNX file path
- `export_params` : If True, includes weights in the ONNX file
- `opset_version` : ONNX opset version (11 or higher recommended)
- `input_names` : Name of the input tensor(s)
- `output_names` : Name of the output tensor(s)

Note.

- `model.eval()` : Set the model to "eval()" mode before exporting
- `batch size` : Batch size **must** be 1

ⓒ Copyright 2025 DEEPX All Rights Reserved.

2.5 JSON File Configuration

This chapter describes various parameters required for compiling an ONNX model using the **DX_COM**. It includes input specifications, calibration methods, data preprocessing settings, and optional parameters for advanced compilation schemes.
These parameters are defined in a JSON file, which serves as a blueprint for how the compiler interprets and processes the input model.

Required Parameters

DEEPX provides the following required parameters for configuring JSON files. These parameters **must** be defined to successfully compile an ONNX model.

Inputs

Defines the input name and shape of the ONNX model.

> ⚠ **Model Input Restrictions**
>
> • The batch size **must** be fixed to 1.
> • **Only** a single input is supported.
> • Input name **must** exactly match ONNX model definition.

Example

```
{
  "inputs": {
    "input.1": [1, 3, 512, 512]
  }
}
```

In this example, `"input.1"` is the name of the input tensor and its shape is `[1, 3, 512, 512]`, where:

- `1`: batch size (**must be 1**)
- `3`: number of channels (e.g., RGB)
- `512 x 512`: image height and width

Calibration Method

ⓒ Copyright 2025 DEEPX All Rights Reserved.

Defines the calibration method used during quantization. It is essential for maintaining model accuracy after compilation by determining appropriate activation ranges.

Available Methods

- `ema`:
  Uses exponential moving average of activation values.
  Recommended for improved post-quantization accuracy.
- `minmax`:
  Uses the minimum and maximum activation values to determine quantization range.

Example

```
{
  "calibration_method": "ema"
}
```

In this example, the ema method is selected to compute more stable and accurate quantization thresholds.

Calibration Number

Defines the number of steps used during calibration. A higher number may improve quantization accuracy by better estimating activation ranges, but may also increase compile time.
To minimize the accuracy degradation, it is recommended to try different values, such as 1, 5, 10, 100, or 1000, and determine the value that yields the best accuracy for your model.

Example

```
{
  "calibration_num": 100
}
```

In this example, 100 samples from the calibration dataset will be used to determine activation ranges for quantization.

Calibration Data Loader

Defines the dataset loader used during the calibration process. This parameter specifies the dataset location, accepted file types, and preprocessing steps to be applied before feeding data into the model.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

Parameter

- `dataset_path` : The directory path where the calibration dataset is located.
- `file_extensions` : A list of allowed file extensions for dataset files (case-sensitive). Only files with these extensions can be used.
- `preprocessings` : Defines preprocessing steps applied to the calibration dataset. These steps should match the preprocessing used during inference to ensure consistency.

Example

```
{
  "default_loader": {
    "dataset_path": "/datasets/ILSVRC2012",
    "file_extensions": ["jpeg", "png", "jpg", "JPEG"],
    "preprocessings": [...]
  }
}
```

Input Preprocessing Operations in `default_loader`

The following preprocessing operations can be applied to input data for calibration or inference. TThese operations help standardize input formats and ensure consistency between calibration and deployment.

`convertColor`

Changes the color channel order of the input images. It is useful when the input image format (e.g., BGR or RGB) differs from what the model expects.

Parameter

- `form` : Defines the type of color space conversion.
- Supported values: `["RGB2BGR", "BGR2RGB", "BGR2GRAY", "BGR2YCrCb"]`

Example

```
{
  "preprocessings": [
  {
    "convertColor": {
      "form": "BGR2RGB"
      }
    }
  ]
}
```


ⓒ Copyright 2025 DEEPX All Rights Reserved.

In this example, the color format is converted from BGR to RGB before being passed to the model.

`resize`

Resizes the input image to a specified target size.This operation is commonly used to match the model's expected input dimensions.

Parameter

- `mode`

  Defines the backend used for resizing.

  `default` : Uses OpenCV's resize function.

  `torchvision` : Uses PIL's resize function.

- `size`

  Defines the target output image size.

  Accepts list, tuple, or integer.

  Cannot be used with `width` and `height` .

- `width, height`

  Defines target width and height.

  Cannot be used with `size` .

- `interpolation` (Optional)

  Defines the interpolation method during resizing.

  Default: `"LINEAR"`

Mode	Supported Interpolation Methods
`default`	`LINEAR` , `NEAREST` , `CUBIC` , `AREA` , `LANCZOS4`
`torchvision`	`BILINEAR` , `NEAREST` , `BICUBIC` , `LANCZOS`

ⓒ Copyright 2025 DEEPX All Rights Reserved.

Example for `default` mode (OpenCV)

```
{
  "preprocessings": [
    {
      "resize": {
        "mode": "default",
        "width": 320,
        "height": 320,
        "interpolation": "LINEAR" # LINEAR, NEAREST, AREA, CUBIC, LANCZOS4
      }
    }
  ]
}
```

Example for `torchvision` mode (PIL-based)

```
{
  "preprocessings": [
    {
      "resize": {
        "mode": "torchvision",
        "size": 250,
        "interpolation": "BILINEAR" # NEAREST, BILINEAR, BICUBIC, LANCZOC

      }
    }
  ]
}
```

`centercrop`

Crops the central region of the input image to the specified dimension. The crop is automatically centered based on the original image size.

Parameter

- `width` : The width of the crop region (in pixels)
- `height` : The height of the crop region (in pixels)

Example

```
{
  "preprocessings": [
    {
      "centercrop": {
        "width": 224,
        "height": 224
      }
```


ⓒ Copyright 2025 DEEPX All Rights Reserved.

```
    }
  ]
}
```

In this example, a 224 × 224 region is cropped from the center of the input image.

`transpose`

Rearranges the dimensions of the input data based on the specified axis order. It is commonly used to convert between data formats.

Paramater

- `axis` : A list specifying the new order of axes

Example

```
{
  "preprocessings": [
    {
      "transpose": {
        "axis": [0, 2, 3, 1]
      }
    }
  ]
}
```

In this example, the tensor dimensions are rearranged to follow the `[batch, height, width, channels]` (NHWC) format.

`expandDim`

Adds a new dimension to the input tensor at the specified axis. It is commonly used to insert a batch or channel dimension when required by the model.

Parameter

- `axis` : The axis index where the new dimension should be inserted

Example

```
{
  "preprocessings": [
    {
      "expandDim": {
        "axis": 0
      }
    }
```


ⓒ Copyright 2025 DEEPX All Rights Reserved.

```
    ]
}
```

In this example, a new dimension is added at axis 0, typically used to add a batch dimension to an image tensor.

normalize

Normalizes the input data by applying mean and standard deviation values for each channel. This is commonly used to standardize input values before feeding them into a model.

> **Warning**
>
> This preprocessing is accelerated by NPU. Thus, it **must not** be applied during NPU runtime.

Parameter

- `mean` : A list of mean values for each channel (e.g., R, G, B)
- `std` : A list of standard deviation values for each channel

Example

```
{
  "preprocessings": [
    {
      "normalize": {
        "mean": [0.486, 0.486, 0.486],
        "std": [0.229, 0.229, 0.229]
      }
    }
  ]
}
```

In this example, the same mean and standard deviation values are applied to all three channels, typically used for normalized RGB images.

mul

Multiplies the input data by a specified constant value. It is commonly used for scaling input pixel values.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

> **Warning**
>
> This preprocessing is accelerated by NPU. Thus, it **must not** be applied during NPU runtime.

Parameter

- `x` : The constant value to multiply with the input data

Example

```
{
  "preprocessings": [
    {
      "mul": {
        "x": 255
      }
    }
  ]
}
```

In this example, the input data is scaled by a factor of 255.

`add`

Adds a constant value to the input data. This operation is commonly used to adjust the input data by a fixed offset, such as shifting pixel values.

> **Warning**
>
> This preprocessing is accelerated by NPU. Thus, it **must not** be applied during NPU runtime.

Parameter

- `x` : The constant value to be added to each element of the input data

Example

```
{
  "preprocessings": [
    {
      "add": {
        "x": 255
      }
    }
```


ⓒ Copyright 2025 DEEPX All Rights Reserved.

```
    ]
}
```

In this example, the value 255 is added to each element of the input data.

`subtract`

Subtracts a constant value from the input data. This operation is commonly used to adjust pixel intensity values or normalize data by removing a fixed offset.

> **Warning**
>
> This preprocessing is accelerated by NPU. Thus, it **must not** be applied during NPU runtime.

Parameter

- `x` : The constant value to subtract from each element of the input data

Example

```
{
  "preprocessings": [
    {
      "subtract": {
        "x": 255
      }
    }
  ]
}
```

In this example, the value 255 is subtracted from each element of the input data.

`div`

Divides the input data by a specified constant. It is commonly used to scale pixel values into a normalized range such as [0, 1].

> **Warning**
>
> This preprocessing is accelerated by NPU. Thus, it **must not** be applied during NPU runtime.

Parameter

- `x` : The constant value to divide each element of the input data by (i.e., the divisor)

ⓒ Copyright 2025 DEEPX All Rights Reserved.

Example

```
{
  "preprocessings": [
    {
      "div": {
        "x": 255
      }
    }
  ]
}
```

In this example, all input values are divided by 255.

Custom Loader

If the model's input is not an image, you can use a custom loader to provide the input data during calibration. The user **must** provide a Python script that defines a custom dataset class.

Guidelines for Writing a Dataset Class
Your custom dataset class **must** implement the following methods.

- `__init__()`

  All constructor arguments **must** be optional, and have default values.

- `__len__()`

  Must return the number of samples in the dataset.

- `__getitem__()`

  Must return the data sample at the given index.

  The returned data **must** have a shape of either CHW (3-dimensional) or C. (1-dimensional).

  The batch dimension (N) is automatically added by the system and is always set to 1.

Recommendation

When using a custom loader, it is recommended to implement preprocessing logic directly within the dataset class, rather than relying on the preprocessing settings in the JSON configuration file.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

This approach offers the following benefits

- Keeps data loading and transformation self-contained
- Improve maintainability and debuggability
- Provide flexibility for non-image input type

Example of `CustomDataset(Dataset)`

```
  import pandas as pd
  import numpy as np
  from PIL import Image

  from torchvision import transforms
  from torch.utils.data import Dataset

  class CustomDataset(Dataset):
      def __init__(self, csv_path="./custom_loader_example/data/mnist_in_csv.csv",
height=28, width=28):
        """
          Custom dataset example for reading data from csv

          Args: (should use default values for custom dataloader)
              csv_path (string): path to csv file
              height (int): image height
              width (int): image width
              transform: pytorch transforms for transforms and tensor conversion
        """
          self.data = pd.read_csv(csv_path)
          # self.labels = np.asarray(self.data.iloc[:, 0]) # not used for calibration
          self.height = height
          self.width = width
          self.transform = transforms.Compose([transforms.ToTensor()])

      def __len__(self):
          return len(self.data.index)

      def __getitem__(self, index):
          # Read each 784 pixels and reshape the 1D array ([784]) to 2D array ([28,28])
          img_as_np = np.asarray(self.data.iloc[index][1:]).reshape(self.height,
self.width).astype('uint8')

         # Convert image from numpy array to PIL image, mode 'L' is for grayscale
          img_as_img = Image.fromarray(img_as_np)
          img_as_img = img_as_img.convert('L')

          # Transform image to tensor
          img_as_tensor = self.transform(img_as_img)

          # returned data shape should be CHW (3-dimensional) or C(1-dimensional)
          return img_as_tensor
```


ⓒ Copyright 2025 DEEPX All Rights Reserved.

Using a Custom Loader in JSON configuration

To use a custom loader during calibration, you **must** specify the python dataset class in the JSON configuration.

If the Python script file is named `dataset_module.py` and the dataset class is `CustomDataset`, the JSON configuration should be as follows.

```
{
  "custom_loader": {
    "package": “dataset_module.CustomDataset”
  }
}
```

The Python script file **must** be in the same directory as the dx_com executable before execution.

Example

```
 dx_com
├── calibration_dataset
├── dataset_module.py
├── dx_com
│ ├── cv2/
│ ├── google/
│ ├── numpy/
│ ├── ...
│ └── dx_com
├── sample
│ ├── MobilenetV1.json
│ └── MobilenetV1.onnx
└── Makefile
```

See also: Download the Custom Dataloader Guide” file for more detailed instruction.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

2.6 Execution of DX-COM

With the ONNX and JSON files prepared, you can run the DXNN compiler to generate the `.dxnn` output file.

Notice to Execute

Output Results Are Not Deterministic
The generated output `(.dxnn)` may vary depending on the system environment, such as CPU, OS, and other hardware factors.

Calibration Data Type
By default, Calibration Data must consist of image files. Supported image formats are JPEG, PNG, and others. If you need to support other data types, **Custom Dataloader** in **Secion 2.5 must** be implemented.

No Support for Multiple-Input ONNX Models
ONNX models with multiple input tensors are currently not supported. Only models with a single input can be compiled.

How to Compile

Use the following commands to generate the NPU Command Set and weights from a target ONNX model and configuration file.

COMMAND FORMAT

```
dx_com -m <MODEL_PATH> -c <CONFIG_PATH> -o <OUTPUT_DIR>
```

- `-m or --model_path MODEL_PATH` : Path to the ONNX Model file `(*.onnx)` .
- `-c or --config_path CONFIG_PATH` : Path to the Model Configuration JSON file `(*.json)` .
- `-o or --output_dir OUTPUT_DIR` : Directory to save the compiled model data.
- `-i or --info` (optional): Print internal module version information and exit.
- `-v or --version` (optional): Print compiler module version and exit.
- `--shrink` (optional): Generate a minimal output by including only the data essential for running on the NPU

ⓒ Copyright 2025 DEEPX All Rights Reserved.

If you want to reduce the file size of the compiled output, use the --shrink option. This ensures the output file contains only the components required for running the NPU execution, excluding debug and intermediate files.

Note. Despite using the same ONNX file in the same PC environment, dx_com may produce different outputs due to internal kernel behavior during optimization and compilation.

Command Examples

Basic Command

```
./dx_com/dx_com \
-m sample/MobilenetV1.onnx \
-c sample/MobilenetV1.json \
-o output/mobilenetv1
```

Using `--shrink` Option

```
./dx_com/dx_com \
-m sample/MobilenetV1.onnx \
-c sample/MobilenetV1.json \
-o output/mobilenetv1 \
--shrink
```

This will generate a minimal output containing only the components essential for NPU execution.

Compile Sample Model with Makefile

You can compile the sample model using the provided Makefile.

```
make mv1
```

This command will compile the `mobilenetv1` sample and generate the output in `./output/mobilenetv1`.

Note. `dx_com` supports ONNX models with batch size fixed to 1 only. If the input shape of the ONNX model is defined as `(batch_size, C, H, W)`, you must overwrite the batch size to 1 before compilation.

```
pip install onnxsim

python3 -m onnxsim EfficientNet.onnx EfficientNet_sim.onnx --overwrite-input-shape
1,3,224,224
```

This will create a new ONNX model `(EfficientNet_sim.onnx)` with the batch size set to 1.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

Output File Structure

After a successful compilation, the DXNN Compiler generates a single output file named

```
{onnx_name}.dxnn
```

This file contains the compiled model and is used as input for both **DX-SIM** and **DX-RT** processes.

`Example Output Directory Structure`

After compiling a sample model using the dx_com tool, your project directory may look like this.

```
dx_com
├── calibration_dataset                  # Dataset used for calibration
...
├── dx_com                               # DXNN Compiler executable and libraries
...
├── output
│   └── mobilenetv1
│       └── MobilenetV1.dxnn             # Compiled output file
├── sample                                # Sample ONNX and JSON files
└── Makefile                              # Build script for compiling sample model
```


ⓒ Copyright 2025 DEEPX All Rights Reserved.

Error Message During Compiling

This section describes common error types that may occur during the compilation process, along with their descriptions and examples of typical causes.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

No	Error Type	Description & Conditions
1	NotSupportError	Triggered when using features unsupported by the compiler. Examples: multi-input models, dynamic input shape, cubic resize
2	ConfigFileError	Invalid or missing JSON configuration file. Examples: incorrect file path, malformed JSON syntax
3	ConfigInputError	Input definitions in the config file do not match the ONNX model. Examples: mismatched input name or shape
4	DatasetPathError	The dataset path specified in the configuration is invalid. Examples: path does not exist, or is not a directory
5	NodeNotFoundError	The ONNX model contains a node that is unsupported by the compiler.
6	OSError	The operating system is unsupported. Examples: OS is not Ubuntu
7	UbuntuVersionError	The installed Ubuntu version is outside the supported range.
8	LDDVersionError	The installed `ldd` version is unsupported.
9	RamSizeError	The system does not meet the minimum RAM requirements.
10	DiskSizeError	Available disk space is insufficient for compilation.
11	NotsupportedPaddingError	Padding configuration is unsupported. Examples: asymmetric padding in width and height
12	RequiredLibraryError	

ⓒ Copyright 2025 DEEPX All Rights Reserved.

No	Error Type	Description & Conditions
		Missing essential system libraries. Examples: `libgl1-mesa-glx` is not installed
13	DataNotFoundError	No valid input data found in the specified dataset path. Examples: empty folder, wrong file extensions
14	OnnxFileNotFound	The ONNX model file cannot be found or does not exist at the specified location.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

3. Building Models

This chapter describes the ONNX operations currently supported by DX-COM. When you build or export models to ONNX format, you **must** use only the supported operations to ensure successful compilation and optimal performance on our NPU.

3.1 Supported ONNX Operations

The following ONNX operators are supported by the compiler.

3.1.1 Common Conditions (Applicable to All Operation Types)

Tensor Shape Limitations

- **Width, height:** < 8,192
- **Channels:** < 32,768
- Dynamic shapes are not supported.

Broadcasting Restrictions

- In element-wise operations like Add, Div, Mul, and Sub, **channel-wise broadcasting** is not supported when the channel dimension size is greater than **1**.
- **Example:** A tensor with shape 1x24x24x1 (NHWC) cannot be broadcast to shape 1x24x24x32.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

3.1.2 Normal Operations

ⓒ Copyright 2025 DEEPX All Rights Reserved.

Operator	Supported Conditions
Add	Supported as: - Bias addition (e.g., as part of `Gemm` or `Conv`) - Element-wise addition - Used for input normalization - Constant scalar addition
ArgMax	Only supported if all of the following hold: - It is the final operation in the network - The preceding output is 2D or 4D - It operates along the channel dimension
AveragePool	- `kernel_shape` < 32 - `strides` < 32
BatchNormalization	No restrictions
Clip	Only supported as an activation function (e.g., ReLU6)
Concat	No restrictions
Constant	Only numeric constants are supported
ConstantOfShape	No restrictions
Conv	**Common constraints:** - `dilations` < 64 - `pads` < 64 - `strides` < 16 **Standard Conv:** - `kernel_shape` < 16 **Depth-wise Conv:** - `kernel_shape` ∈ {[3, 3], [5, 5]} - Only constant weights are supported
ConvTranspose	- `dilations` = [1, 1] - `pads` ≤ 14 - `strides` ∈ [2, 8] - `kernel_shape` < 16 - `group` = 1
Div	Supported as: - Constant scalar division - Input normalization

ⓒ Copyright 2025 DEEPX All Rights Reserved.

Operator	Supported Conditions
	- Part of `Softmax` - Part of `LayerNorm`
Dropout	Removed during inference
Erf	Only supported as part of `GELU`
Flatten	Only supported for reshaping `Conv` output into `Dense` input
Gemm	No restrictions
GlobalAveragePool	No restrictions
Identity	No restrictions
MatMul	No restrictions
MaxPool	- `kernel_shape` < 16 - `strides` < 16
Mul	Supported as: - Element-wise multiplication - Constant scalar multiplication - Input normalization
Pad	- Only `mode=constant` is supported - Must precede a `Pool` or `Conv` operation
ReduceMean	Only supported when reducing along: - Channel dimension - (Width, Height) dimensions
ReduceSum	Only supported when reducing along the channel dimension
Reshape	Supported as: - Squeeze-like transformations - As part of the attention mechanism
Resize	Only supported with the following attributes: - `coordinate_transformation_mode` = `pytorch_half_pixel` - `mode` $\in$ {`nearest`, `linear`} - Scale values $\in \mathbb{Z}$ (integers)
Shape	Cannot be used as a model output
Slice	Only supported when slicing along the height dimension

ⓒ Copyright 2025 DEEPX All Rights Reserved.

Operator	Supported Conditions
Softmax	Only supported if the size of the input along the specified `axis` is ≤ 4080
Split	Input tensor must have rank 4
Squeeze	No restrictions
Sub	Supported as: - Element-wise subtraction - Constant scalar subtraction - Input normalization
Transpose	Only supported as part of the attention mechanism

3.1.3 Deprecated Operations

The following operations are deprecated in ONNX and maintained here only for backward compatibility.
Their usage is discouraged in new models and may be removed in future versions.
Please use alternative operators where possible.

Operator	Supported Conditions
Upsample	Only supported when scale values in the N and C dimensions are 1

ⓒ Copyright 2025 DEEPX All Rights Reserved.

3.1.4 Activation Functions

Operator	Supported Conditions
HardSwish	No restrictions
HardSigmoid	No restrictions
LeakyRelu	No restrictions
Mish	No restrictions
PRelu	No restrictions
Relu	No restrictions
Sigmoid	No restrictions
Silu (Swish)	No restrictions
Softplus	No restrictions
Tanh	No restrictions

Note: Operator support may vary depending on how operations are combined within a model. This document is intended as a general guideline. For validation of specific use cases, please contact our technical support team.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

4. Development Tools

4.1 Model Viewer_DX-TRON

Overview

DX-TRON is a graphical visualization tool for exploring .dxnn model files compiled with the DEEPX toolchain. It allows users to load and inspect model structures, view workload distribution between NPU and CPU through color-coded graphs. With **DX-TRON**, users can better understand model execution flow and improve overall performance.

Key Features

- Supports for .dxnn Files

  Load and visualize model files compiled with the DEEPX toolchain.
  - Visual Workload Representation

    Displays a color-coded breakdown of workload execution:
    - Red: Operations executed on the NPU
    - Blue: Operations executed on the CPU or host

• Model Navigation Controls

  Use the backward arrow in the bottom-left corner to return to the model overview screen at any time.

• Interactive Node Inspection

  Double-click any node within the graph to view detailed information about the associated operations.

Installation on Linux

This section explains how to install and run DX-TRON on Ubuntu using the .AppImage file.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

Step 1. Install Required Packages
DX-TRON requires several system libraries to run. Run the following commands to install dependencies.

```
./scripts/install_prerequisites.sh
```

NOTE. `libfuse2` is mandatory for AppImage execution and may not be installed by default on Ubuntu 22.04 or later.

Step 2. Prepare the AppImage File
Make the AppImage executable.

```
chmod +x DXTron-x.y.z.AppImage
```

Step 3. Run DX-TRON
Execute the AppImage.

```
./DXTron-x.y.z.AppImage
```

Once launched, a GUI window will appear, enabling DXNN model visualization.

WARNING. Running DX-TRON with `sudo` may require the `--no-sandbox` flag, but this is **not recommended** for security reasons.

Installation on Windows

This section explains how to install and launch DX-TRON on Windows using the provided setup file.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

Step 1. Installation File

- File Name: `DXTron Setup x.y.z.exe`

Step 2. Installation Steps

1) Save the provided .exe file to your PC.
2) Double-click the file to launch the installer.
3) When the User Account Control (UAC) prompt appears, click **Yes**.
4) Follow the on-screen instructions in the installation wizard.
5) After installation, a DX-TRON shortcut is created in the Start Menu and optionally on the Desktop.

The default installation path is as follows.

```
C:\Users\YourUsername\AppData\Local\Programs\DXTron\
```

Step 3. Launching DX-TRON

- Search for DX-TRON in the Start Menu and run it, or
- Double-click the Desktop shortcut (if created).

Troubleshooting

Step 1. Security Warnings on Windows

If Microsoft Defender SmartScreen blocks the installer:
1) Click More Info.
2) Select Run Anyway to continue installation.

Step 2. Program Fails to Launch

Perform the following checks in order.
1) Check AppImage Permissioni (Linux)
- Ensure the file is executable.

```
chmod +x DXTron-x.y.z.AppImage
```

2) Verify Required Libraries (Linux)
- Confirm all dependencies are installed.

3) Reinstall DX-TRON
- Uninstall and reinstall DX-TRON to ensure all files are properly installed.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

ⓒ Copyright 2025 DEEPX All Rights Reserved.

5. Change Log

5.1 v2.0.0 (August 2025)

- Re-enabled support for the following operators:
- `Softmax`
- `Slice`
- Newly added support for the following operator:
- `ConvTranspose`
- Partial support for Vision Transformer (ViT) models:
- Verified with the following OpenCLIP models:
- ViT-L-14, ViT-L-14-336, ViT-L-14-quickgelu
- RN50x64, RN50x16
- ViT-B-16, ViT-B-32-256, ViT-B-16-quickgelu
- Compatibility with DX-RT versions earlier than v3.0.0 is not guaranteed.
- The `DXQ` option has been removed and will be reintroduced in a future release.
- `PPU(Post-Processing Unit)` is no longer supported, and there are no current plans to reinstate it.

5.2 v1.60.1 (June 2025)

- Internal bug fixes.
- Added support for:
- `-v` option: Displays **DX-COM module version**
- `-i` option: Displays **internal module information**
  → For usage, see: Command Format
- The following operators were deprecated and are scheduled to be re-supported in a future release:
- `Softmax`
- `Slice`

ⓒ Copyright 2025 DEEPX All Rights Reserved.