Developing Models on Alveo U200

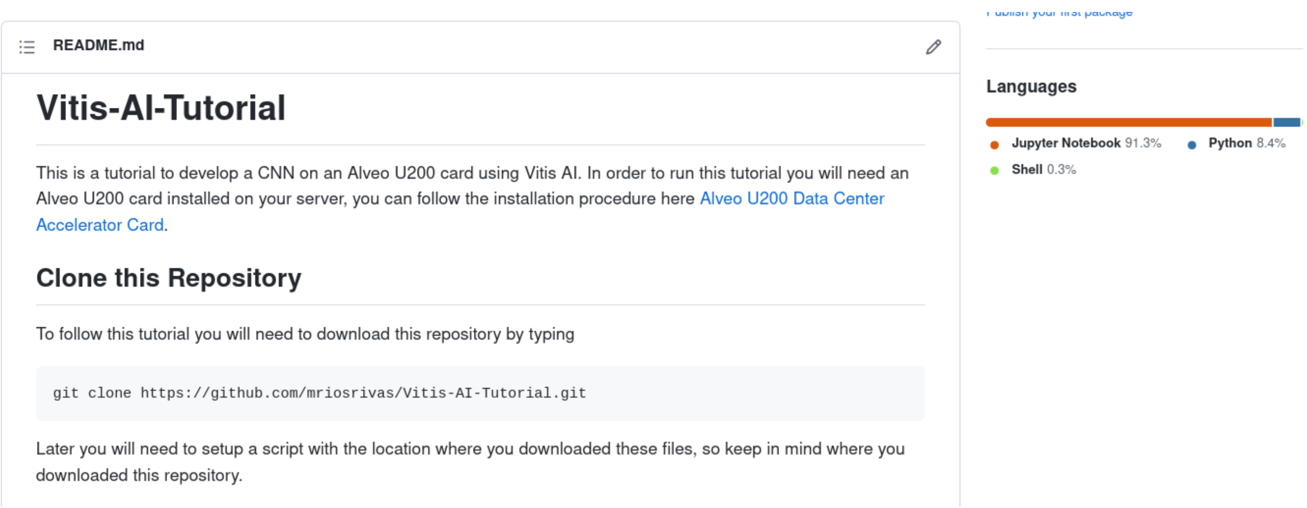
Manuel Rios

GitHub Repository

Clone GitHub Repository

You can download all the source material from here:

```
git clone https://github.com/mriosrivas/Vitis-AI-Tutorial.git
```

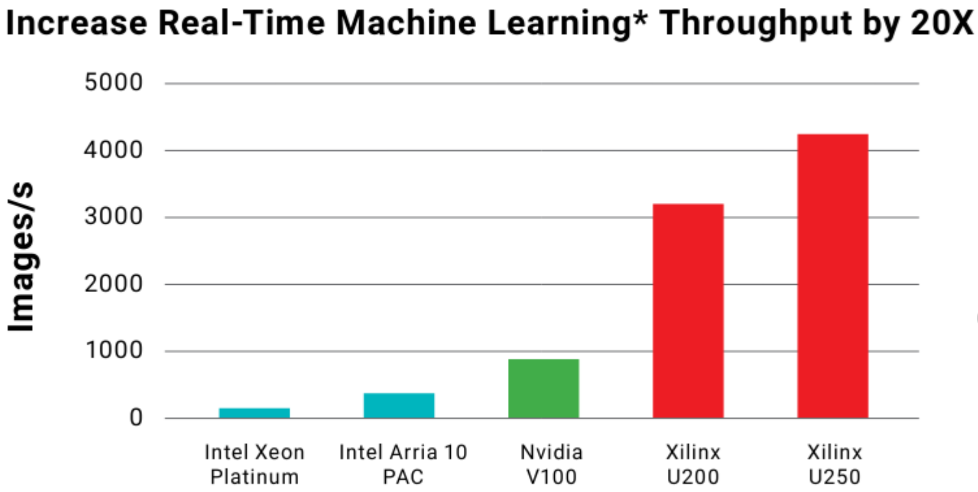


Alveo U200

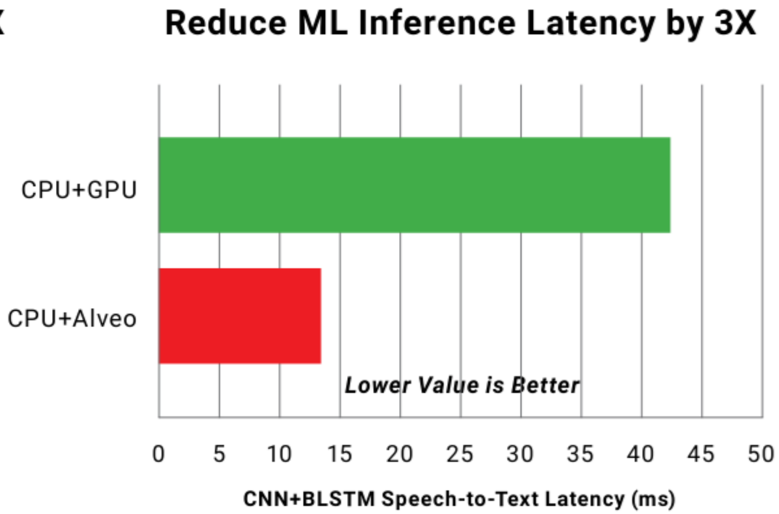
Alveo U200

- Xilinx 16nm UltraScale™ architecture
- Adaptable to any workload
  - Database Search & Analytics
  - Financial Computing
  - Machine Learning
  - Storage Compression
  - Video Processing/Transcoding
  - Genomics

Alveo U200



*GoogleNet V1: Accelerating DNNs with Xilinx Alveo Accelerator Cards White Paper



CPU+GPU: Nvidia P4 + Xeon CPU E5-2690 v4 @2.60GHz (56 Cores)
CPU+Alveo: Alveo U200 or U250 + Xeon CPU E5-2686 v4 @2.3GHz (8 Cores)

Alveo U200 -DPUCADF8H

The Xilinx® **Deep Learning Processor Unit** (DPU) is a series of soft IP dedicated for convolutional neural networks acceleration.

DPUCADF8H is a high throughput CNN inference IP for Alveo cards.

The IP is optimized for **high-resolution image networks** and featured with high efficiency.

It runs with a set of efficiently optimized instructions and it can support most **convolutional neural networks**, such as VGG, ResNet, GoogLeNet, YOLO, SSD, MobileNet, and FPN.

Alveo U200 -DPUCADF8H

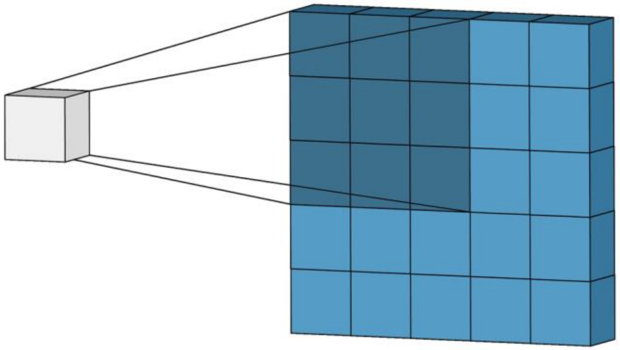
Highlights of DPUCADF8H functionality include:

- Convolution, dilated convolution, and deconvolution
- Maximum and average pooling
- Element wise sum
- ReLU
- Data split and concat
- Data reorganization
- Fully connected layer
- Batch normalization

Alveo U200 -DPUCADF8H

Highlights of DPUCADF8H functionality include:

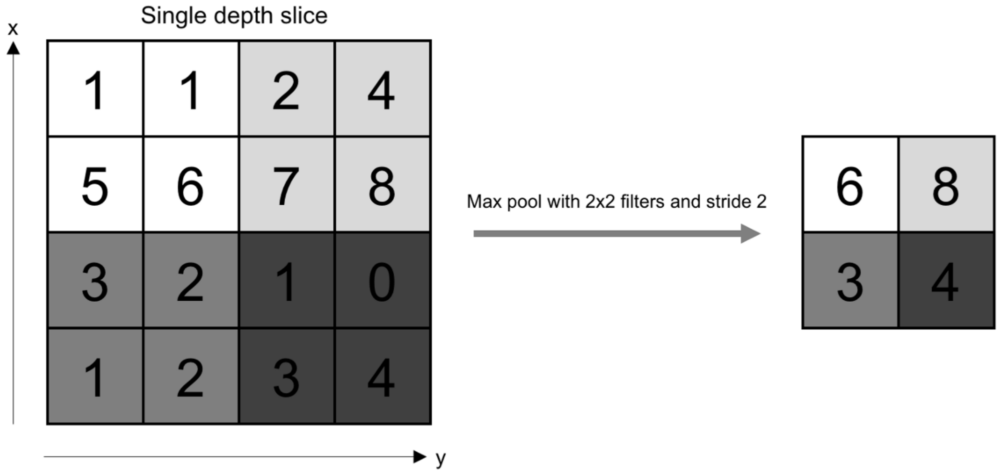
- **Convolution, dilated convolution, and deconvolution**
- Maximum and average pooling
- Element wise sum
- ReLU
- Data split and concat
- Data reorganization
- Fully connected layer
- Batch normalization



Alveo U200 -DPUCADF8H

Highlights of DPUCADF8H functionality include:

- Convolution, dilated convolution, and deconvolution
- **Maximum and average pooling**
- Element wise sum
- ReLU
- Data split and concat
- Data reorganization
- Fully connected layer
- Batch normalization



Alveo U200 -DPUCADF8H

Highlights of DPUCADF8H functionality include:

- Convolution, dilated convolution, and deconvolution
- Maximum and average pooling
- **Element wise sum**
- ReLU
- Data split and concat
- Data reorganization
- Fully connected layer
- Batch normalization

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} + \begin{bmatrix} 1 & 2 \end{bmatrix} = \begin{pmatrix} 1 & 3 \\ 3 & 5 \end{pmatrix}$$

Add to each column

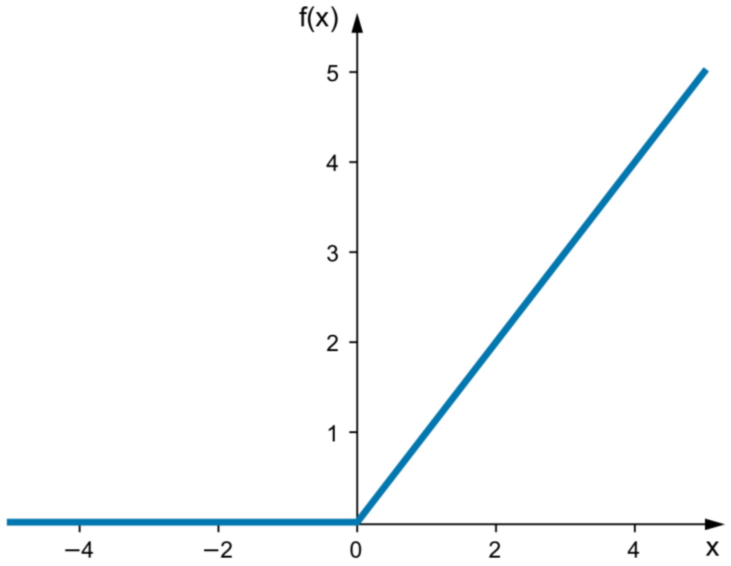
$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$$

Add to each row

Alveo U200 -DPUCADF8H

Highlights of DPUCADF8H functionality include:

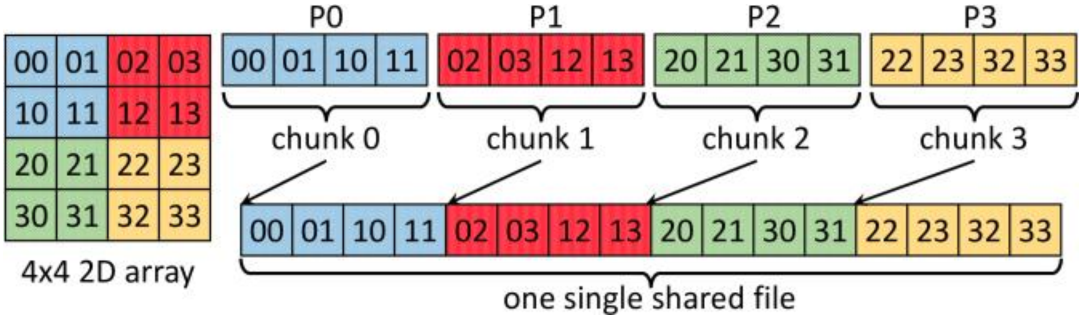
- Convolution, dilated convolution, and deconvolution
- Maximum and average pooling
- Element wise sum
- **ReLU**
- Data split and concat
- Data reorganization
- Fully connected layer
- Batch normalization



Alveo U200 -DPUCADF8H

Highlights of DPUCADF8H functionality include:

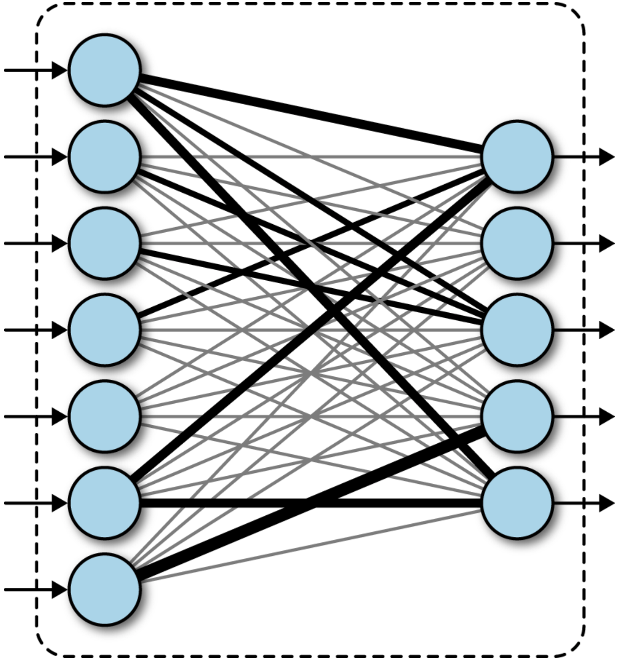
- Convolution, dilated convolution, and deconvolution
- Maximum and average pooling
- Element wise sum
- ReLU
- **Data split and concat**
- **Data reorganization**
- Fully connected layer
- Batch normalization



Alveo U200 -DPUCADF8H

Highlights of DPUCADF8H functionality include:

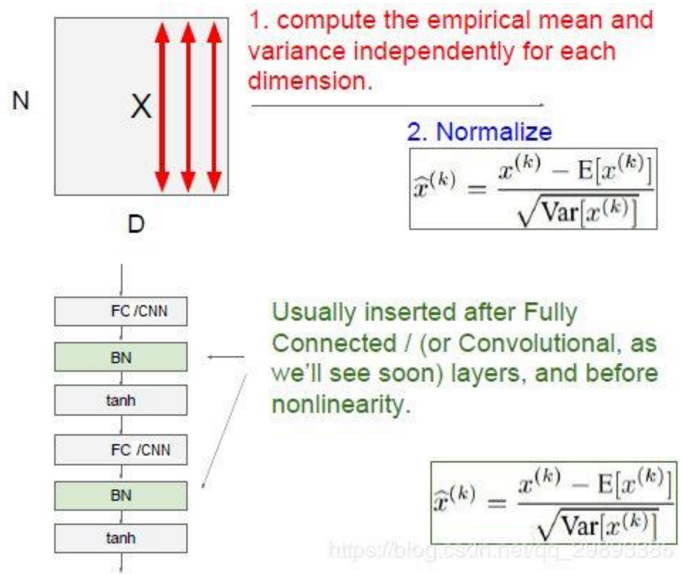
- Convolution, dilated convolution, and deconvolution
- Maximum and average pooling
- Element wise sum
- ReLU
- Data split and concat
- Data reorganization
- **Fully connected layer**
- Batch normalization



Alveo U200 -DPUCADF8H

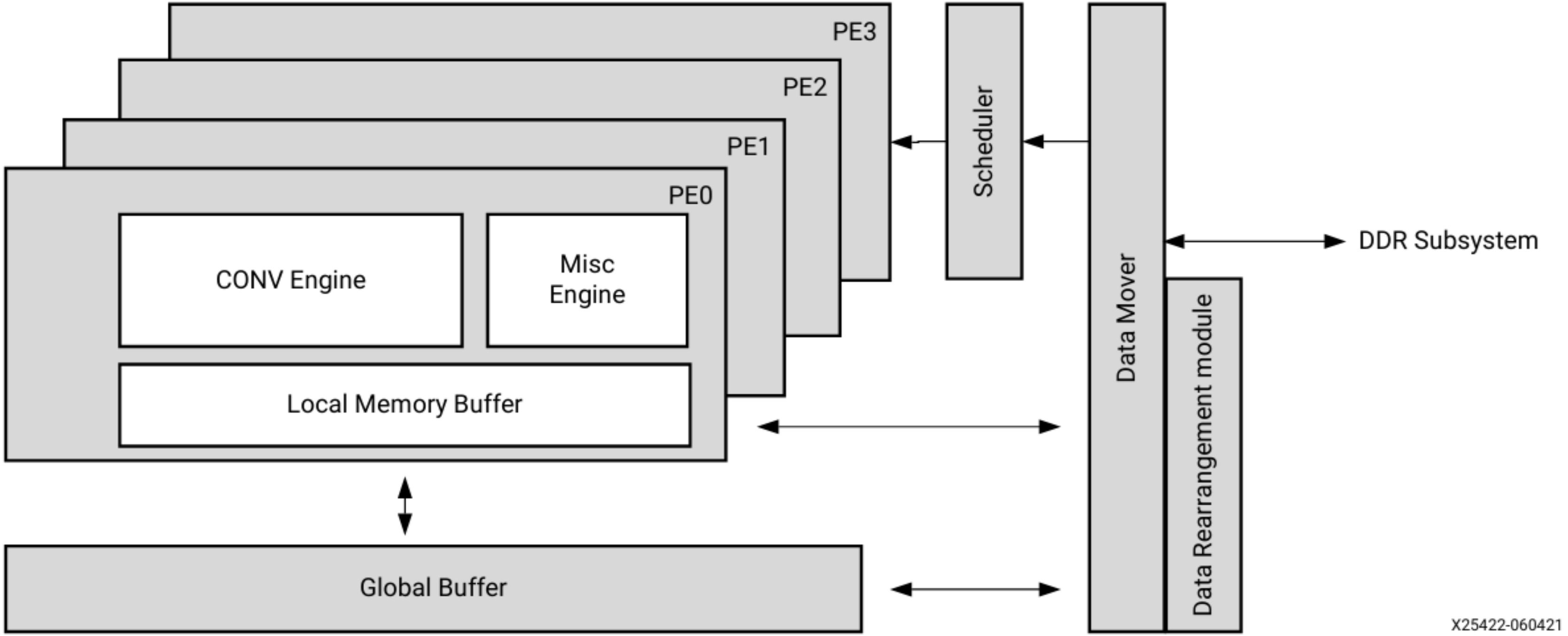
Highlights of DPUCADF8H functionality include:

- Convolution, dilated convolution, and deconvolution
- Maximum and average pooling
- Element wise sum
- ReLU
- Data split and concat
- Data reorganization
- Fully connected layer
- **Batch normalization**



Alveo U200 -DPUCADF8H

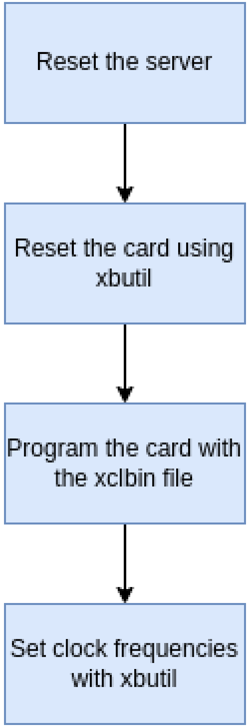
Figure 1: **DPUCADF8H Top-Level Block Diagram**



Server Setup

Frequency Setup of Alveo U200 Card

- If we let the Alveo card to work with the **default settings**, it will **overheat**.
- Temperature higher than **91°C will shutdown** the card.
- To solve this issue we **change** the **clock frequency**.



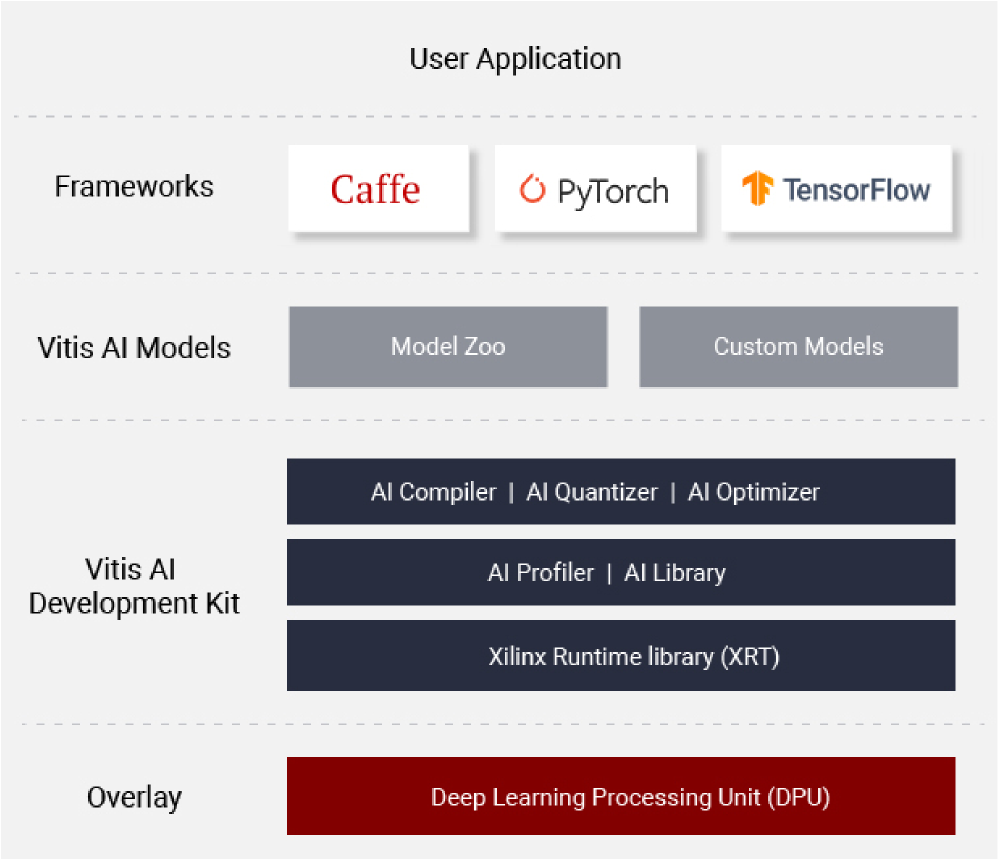
Frequency Setup of Alveo U200 Card

```
# Run this in the host machine (not in docker)

# Reset the card with id 0000:17:00.1
xbutil reset -d 0000:17:00.1

# Program the card with the xclbin file
xbutil program -d 0000:17:00.1 -u
/opt/xilinx/overlaybins/DPUCADF8H/dpdpuv3_wrapper.hw.xilinx_u200_gen3x16
_xdma_1_202110_1.xclbin

# Set the clock frequency
xbutil --legacy clock -d 0000:17:00.1 -f 70 -g 70
```

Vitis AI

Vitis AI

Development stack for AI inference **on Xilinx hardware** platforms, including both edge devices and Alveo cards.



Setup Vitis AI

1. **Clone Vitis AI** repository

```
git clone --recurse-submodules https://github.com/Xilinx/Vitis-AI
cd Vitis-AI
```

1. **Pull Docker image**

```
docker pull xilinx/vitis-ai-cpu:latest
```

Note: These two steps are **already done in the server**.

Setup Vitis AI

3. Edit **docker_run.sh** to add your project as a volume.

```
docker_run_params=$(cat <<-END
    -v /dev/shm:/dev/shm \
    -v /opt/xilinx/dsa:/opt/xilinx/dsa \
    -v /opt/xilinx/overlaybins:/opt/xilinx/overlaybins \
    -e USER=$user -e UID=$uid -e GID=$gid \
    -e VERSION=$VERSION \
    -v $DOCKER_RUN_DIR:/vitis_ai_home \
    -v $HERE:/workspace \
    -v /home/manuel/tutorial:/tutorial \
    -w /workspace \
    --rm \
    --network=host \
    ${DETACHED} \
    ${RUN_MODE} \
    $IMAGE_NAME \
    $DEFAULT_COMMAND
END
)
```

Run Docker and setup environment

1. **Run** your **Docker** container

```
cd Vitis-AI
./docker_run.sh
```

1. **Activate** the **Tensorflow 2 environment**

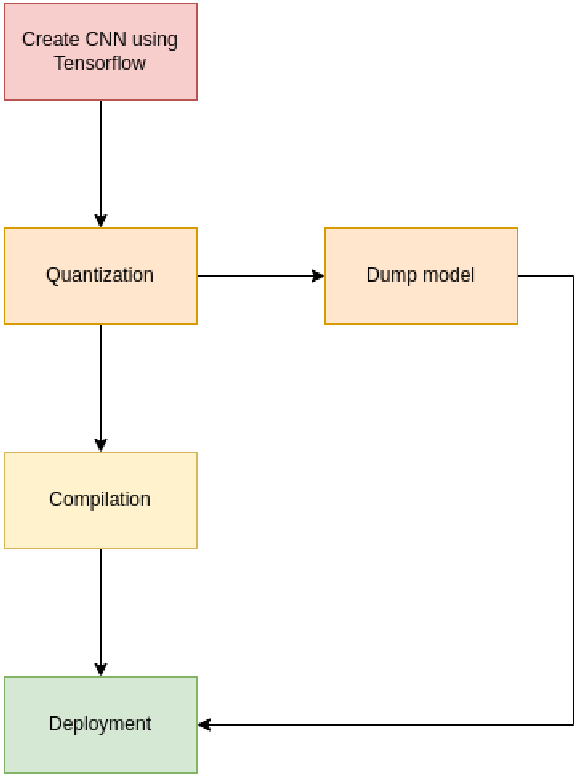
```
conda activate vitis-ai-tensorflow2
```

1. **Source** the Xilinx **environment variables**

```
source /workspace/setup/alveo/setup.sh DPUCADF8H
```

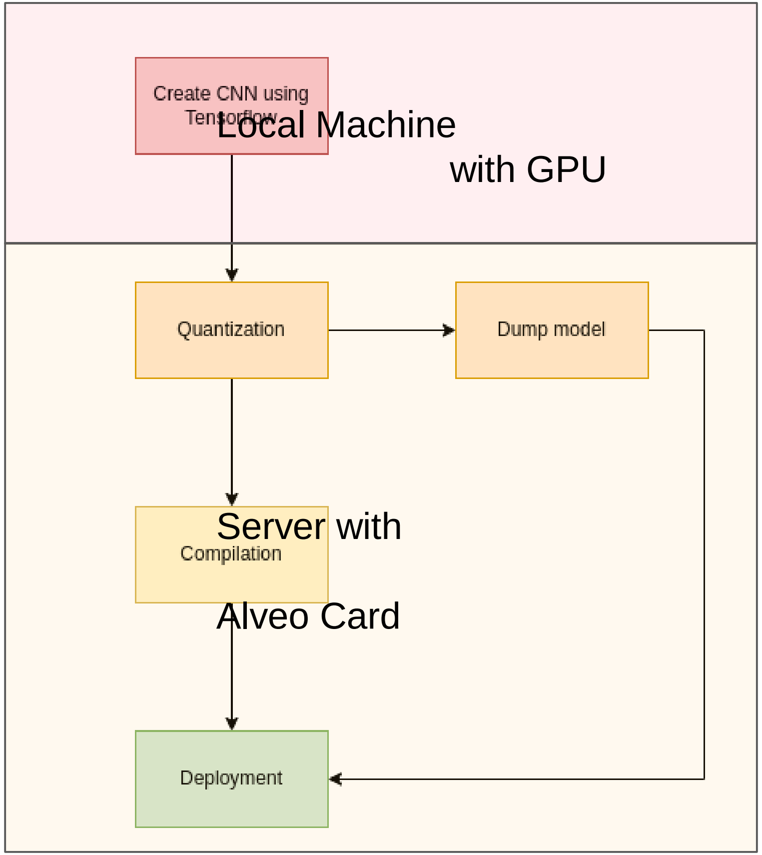
Workflow

Workflow

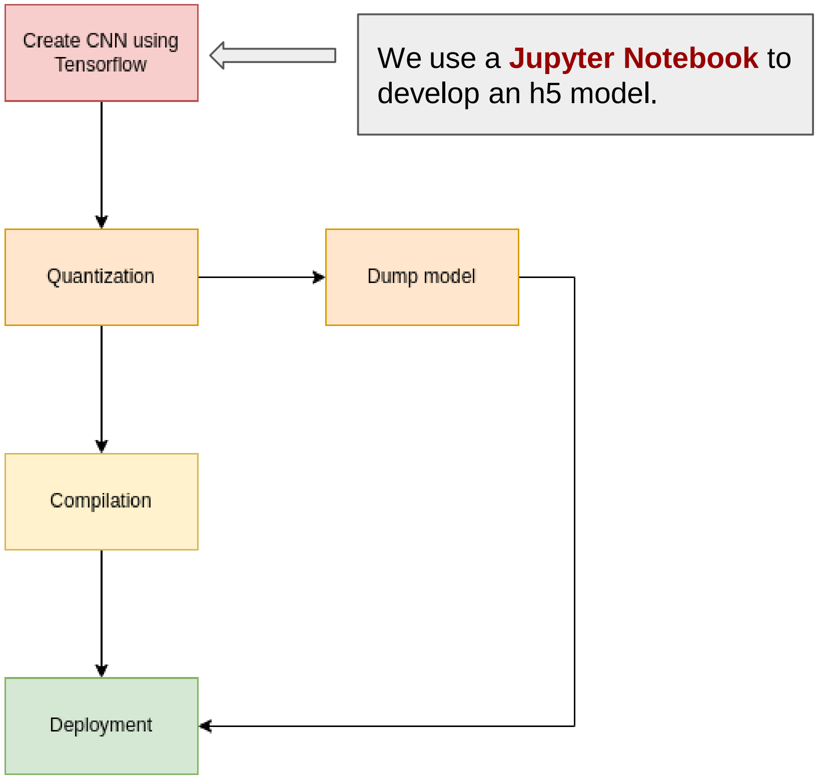


Create CNN using Tensorflow
Quantization
Dump model
Compilation
Deployment

Workflow

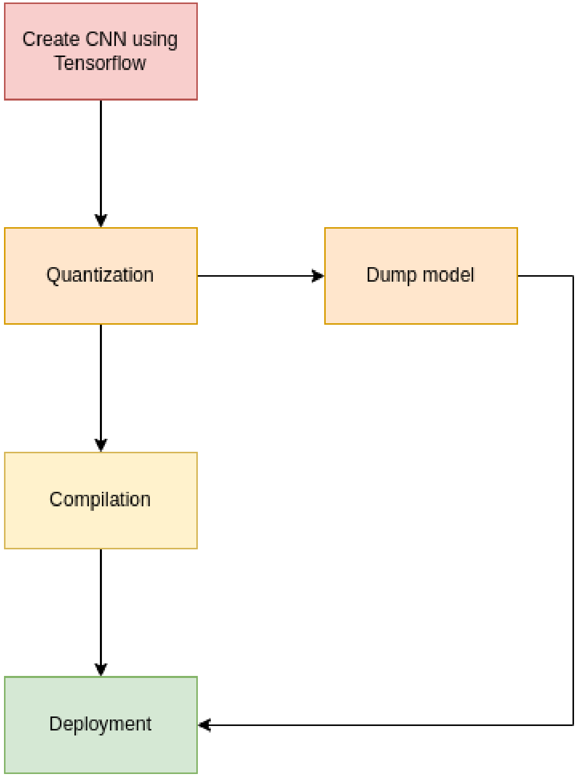

Create CNN using Tensorflow
Local Machine
with GPU
Quantization
Dump model
Server with
Compilation
Alveo Card
Deployment

Workflow

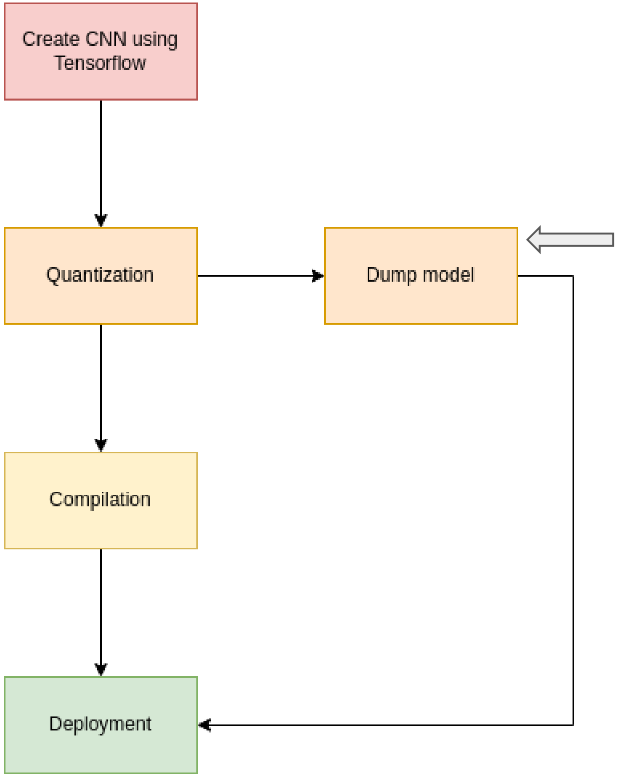

Create CNN using Tensorflow
We use a **Jupyter Notebook** to develop an h5 model.
Quantization
Dump model
Compilation
Deployment

Workflow

A script called **quantize.py** is used to transform our model into integer number precision.

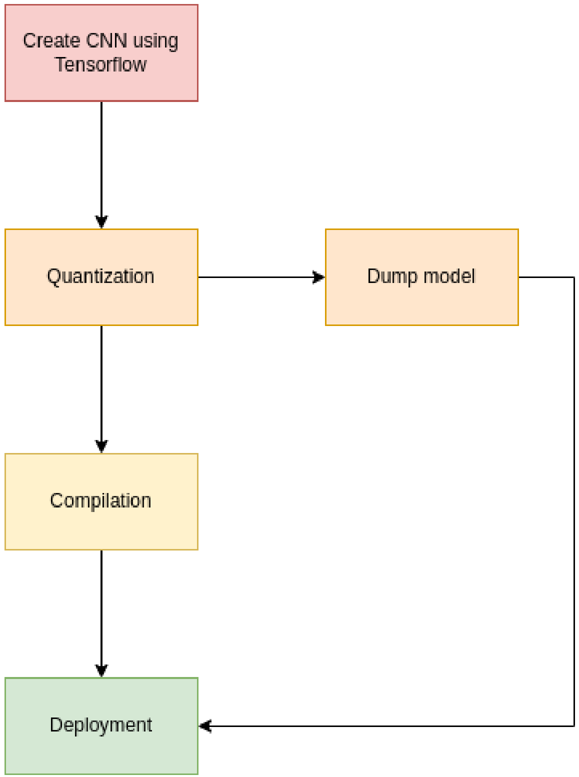


Workflow



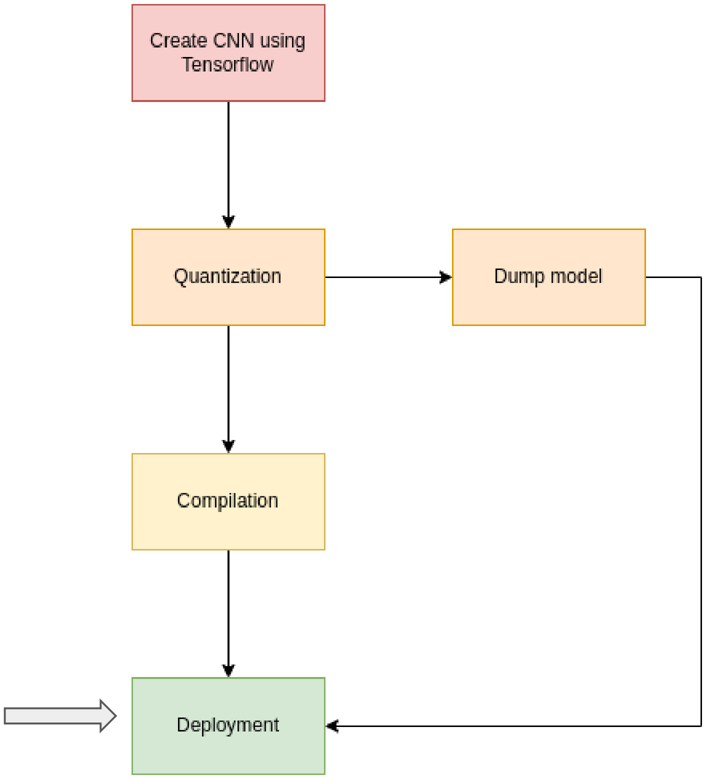
A script called **dump.py** will dump our simulation results for later comparison.

Workflow



A script called **compile.sh** is used to transform our quantized model into an xmodel for the DPU.

Workflow



A script called **inference.py** will perform inference using the developed model.

1. Create CNN Using Tensorflow

- Create and train a **cat-dog classifier**.
- Use **Tensorflow 2.8.0** and **Keras**.
- Save the model as an **H5 file** for later use.

1. Create CNN Using Tensorflow

Basic architecture:

- Convolutional Neural Network
- Max Pooling
- Batch Normalization
- Dense Layers

Optimizer:

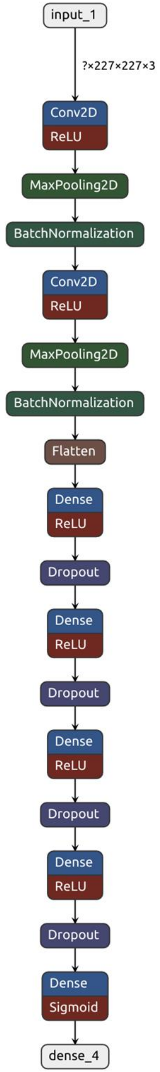
- RMSprop

Loss:

- Binary Cross Entropy

Metric:

- Binary Accuracy



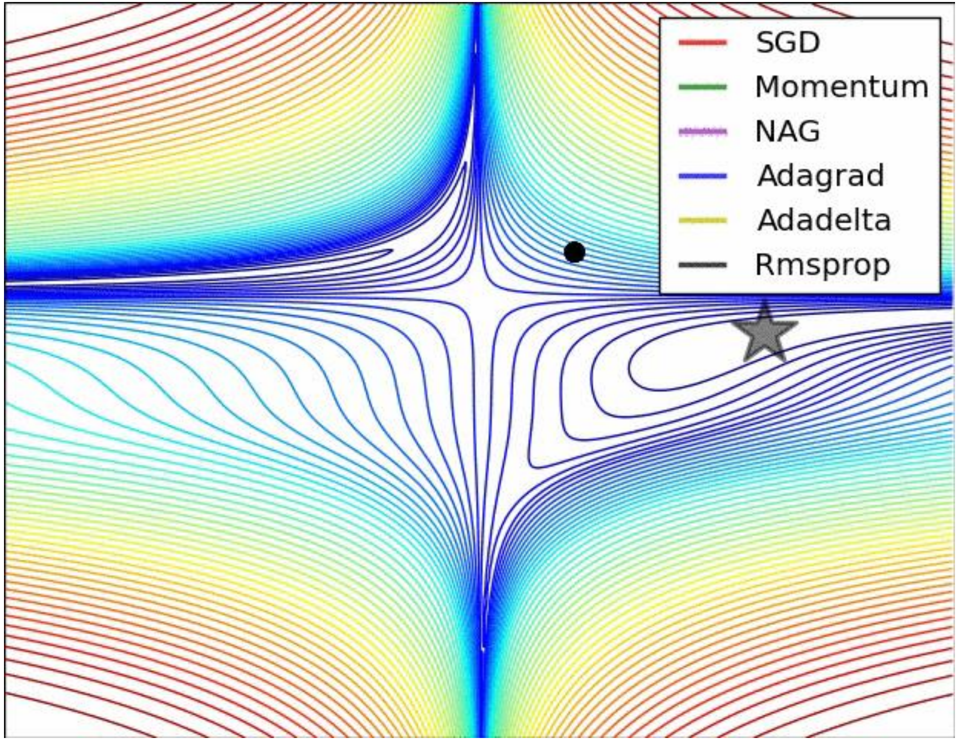
1. Create CNN Using Tensorflow

Basic architecture:
- Convolutional Neural Network
- Max Pooling
- Batch Normalization
- Dense Layers

Optimizer:
- RMSprop

Loss:
- Binary Cross Entropy

Metric:
- Binary Accuracy



1. Create CNN Using Tensorflow

Basic architecture:
- Convolutional Neural Network
- Max Pooling
- Batch Normalization
- Dense Layers

Optimizer:
- RMSprop

Loss:
- Binary Cross Entropy

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Metric:
- Binary Accuracy

1. Create CNN Using Tensorflow

Basic architecture:

- Convolutional Neural Network
- Max Pooling
- Batch Normalization
- Dense Layers

Optimizer:

- RMSprop

Loss:

- Binary Cross Entropy

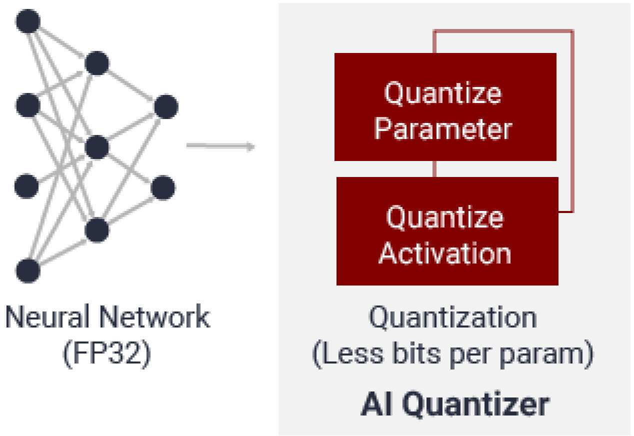
Metric:

- Binary Accuracy

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

2. Quantize model

Quantization is the process of converting a **32-bit floating point model** into an **INT8 representation**.



2. Quantize model

There are **two** different **approaches to quantize a deep learning model**:

- **Post-training quantization** (PTQ)
  - Convert a **pre-trained float model** into a quantized model with little degradation in model accuracy.
  - A representative **dataset** is needed to **run a few batches of inference** on the float model.
- **Quantization aware training** (QAT)
  - Models the quantization errors in both the **forward and backward passes** during model quantization.

2. Post-training quantization (PTQ)

```
## Load model trained in Keras (Tensorflow 2)
float_model=tf.keras.models.load_model('float_model.h5')

## Quantize the model
quantizer = vitis_quantize.VitisQuantizer(float_model)


quantized_model = quantizer.quantize_model(calib_dataset=train_ds,
                                           calib_batch_size=4,
                                           replace_sigmoid=True,
                                           input_shape="?,227,227,3",
                                           weight_bit=8,
                                           bias_bit=8)



## Save quantized model
quantized_model.save('quantized_model.h5')
```

2. Post-training quantization (PTQ)

```
## Evaluate model
quantized_model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
                        metrics= ['binary_accuracy'])

quantized_model.evaluate(val_ds)
```

3. Dump Model

- When deploying a model, **debugging becomes difficult**, therefore **dumping** the simulation **results can help**.
- Dumping results can be:
    - **Inputs** and **outputs**
    - **Weights** and **bias**
- Results can be **integer and floating** point for reference.

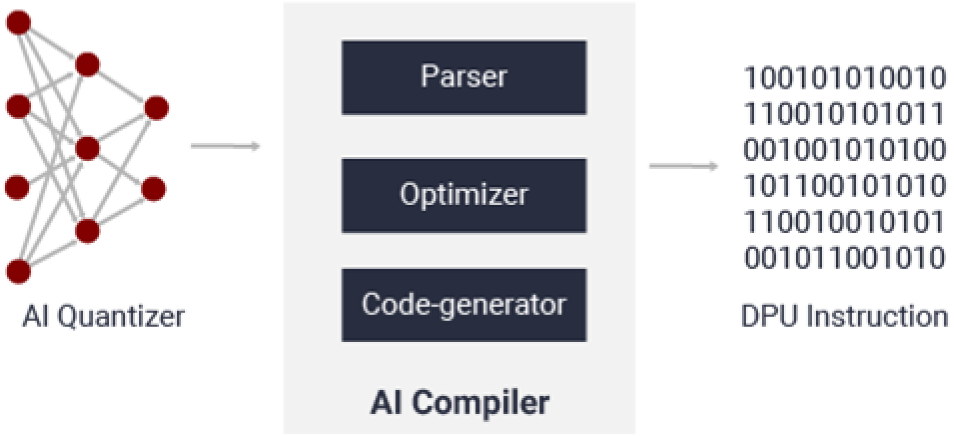
3. Dump Model

```
vitis_quantize.VitisQuantizer.dump_model(model=quantized_model,
                                         dataset=dump_ds,
                                         dump_float=True,
                                         output_dir='./dump_model')
```

4. Compile model

Vitis™ AI compiler (VAI_C):

- Interface to a **compiler family** targeting the optimization of **neural-network computations to different DPUs**.
- **Maps a network** model **to** a highly optimized **DPU instruction sequence**.



4. Compile model

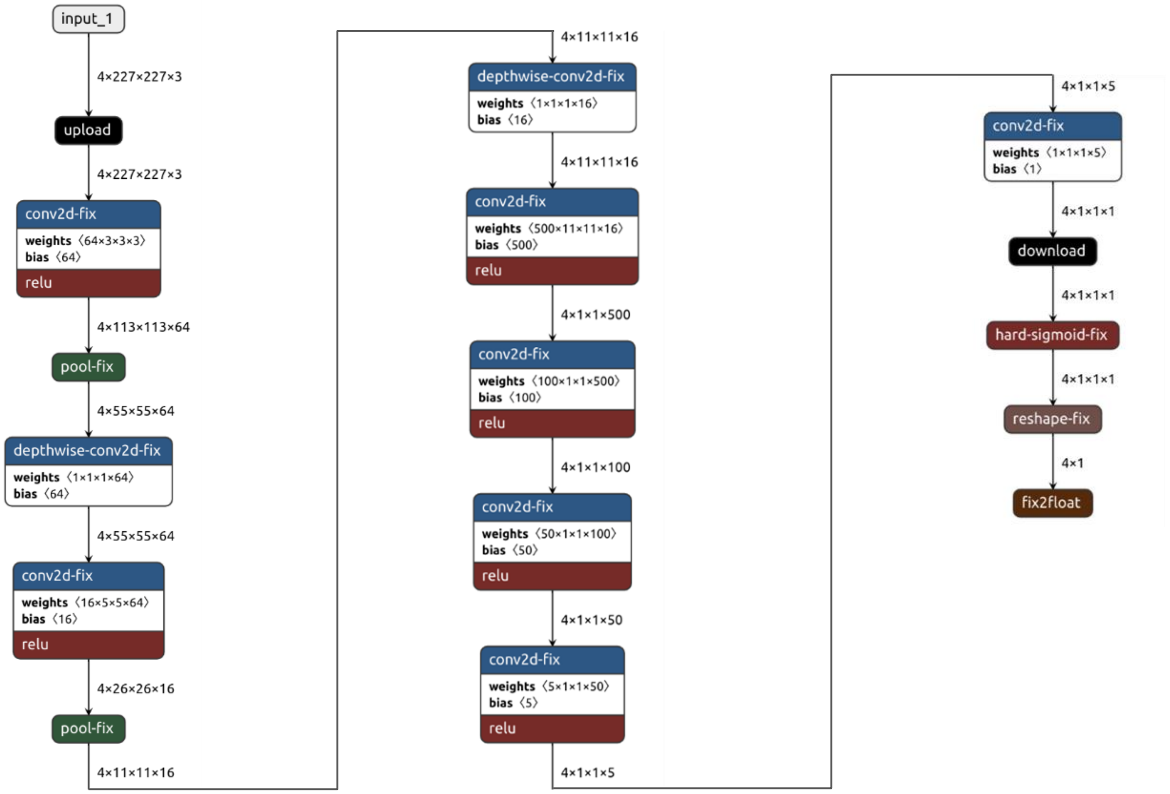
```
vai_c_tensorflow2 \
    --model quantized_model.h5\
    --arch /opt/vitis_ai/compiler/arch/DPUCADF8H/U200/arch.json \
    --output_dir compile_model \
    --net_name deploy \
    --options '{"input_shape": "4,227,227,3"}'
```

4. Compile model

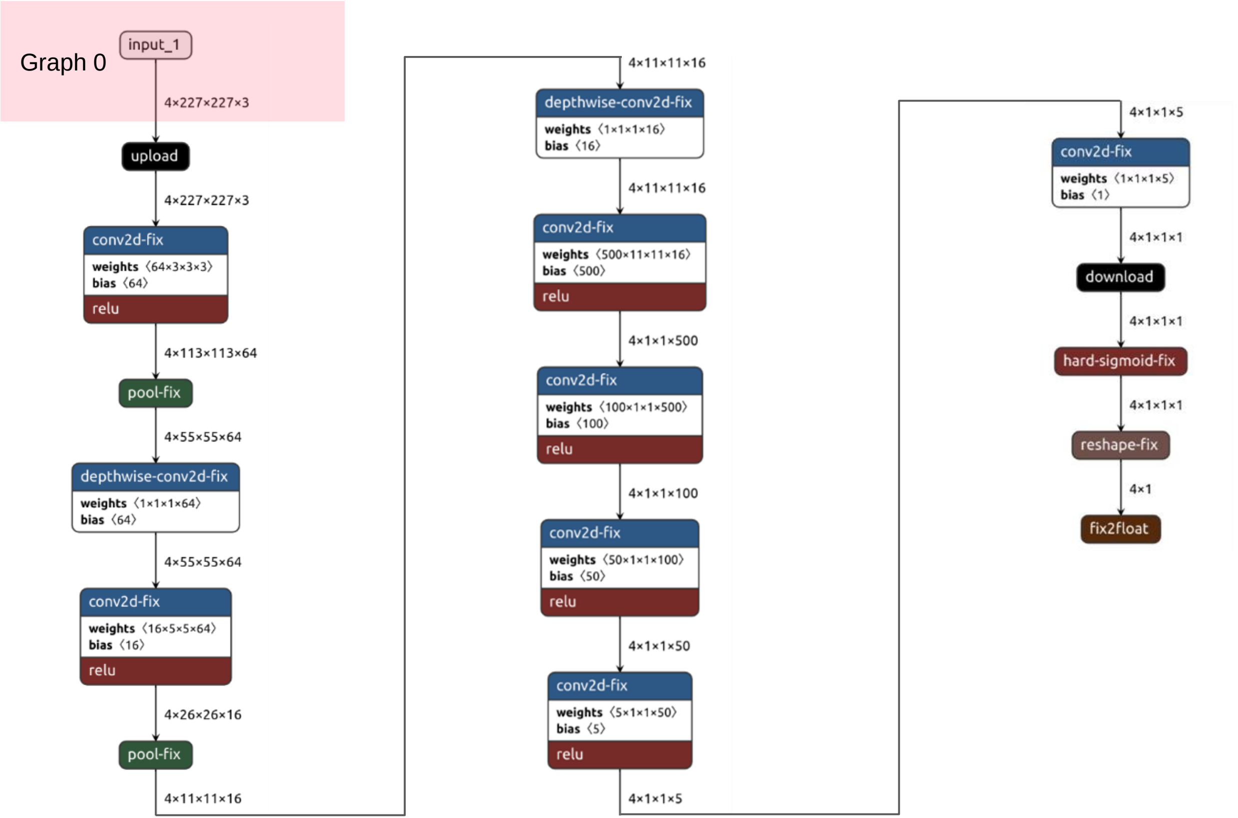
We can plot the model graph using

```
xdputil xmodel deploy.xmodel -s xmodel_graph.png
```

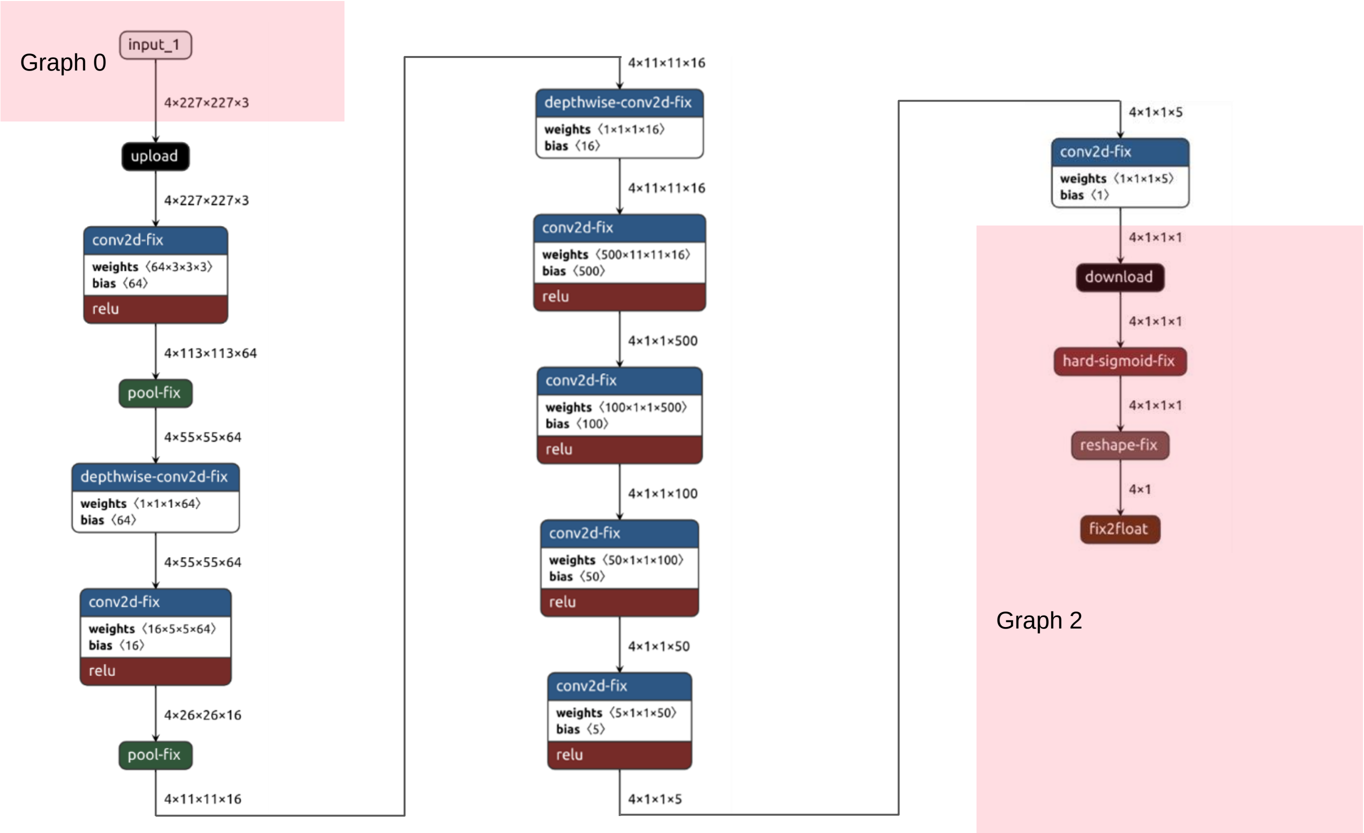
5. Deployment: Graph



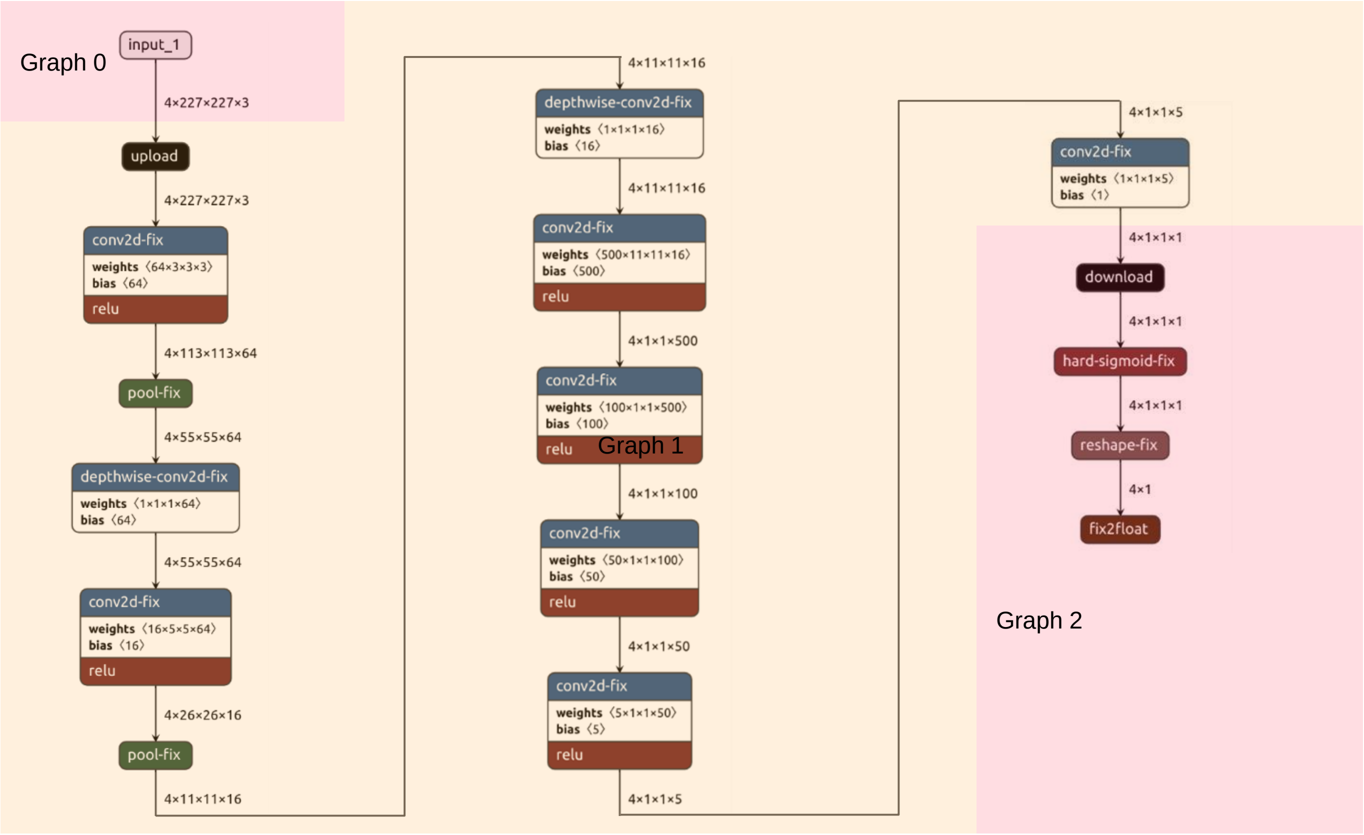
5. Deployment: Sub-Graphs



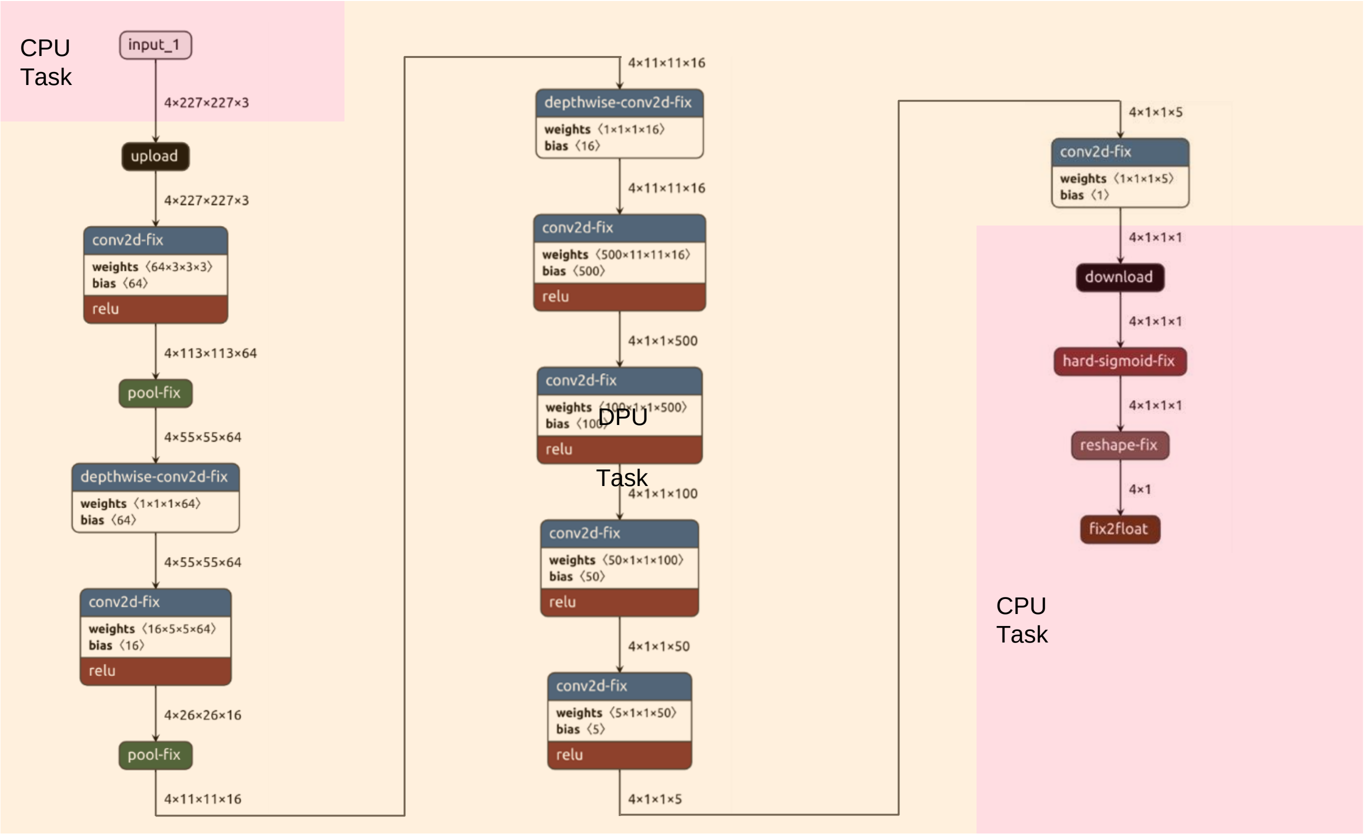
5. Deployment: Sub-Graphs



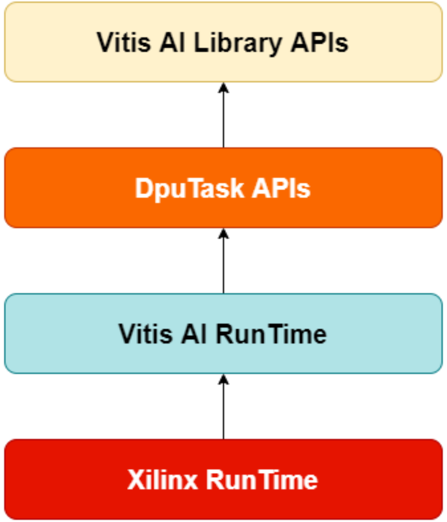
5. Deployment: Sub-Graphs



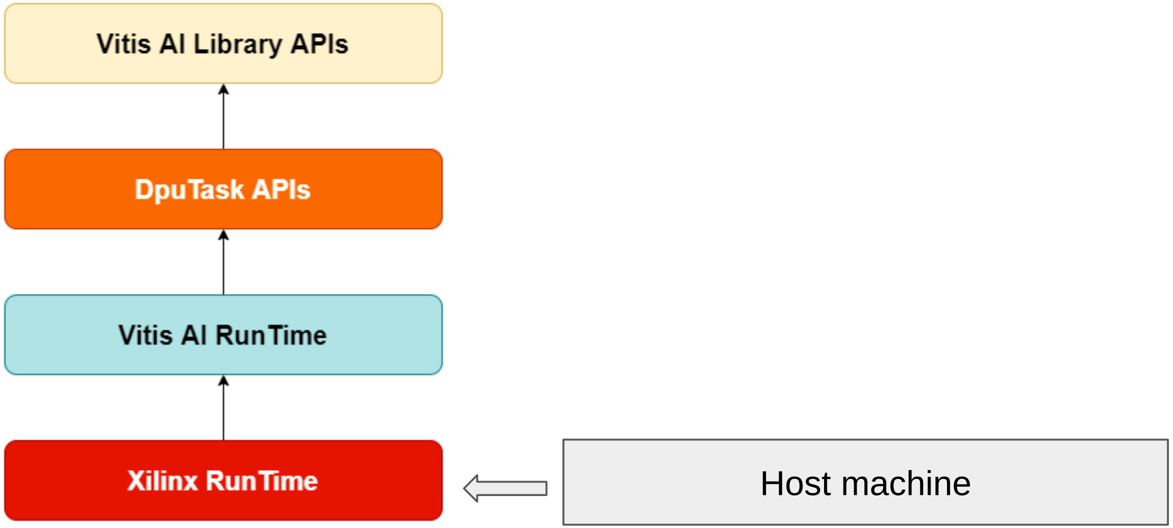
5. Deployment: Sub-Graphs



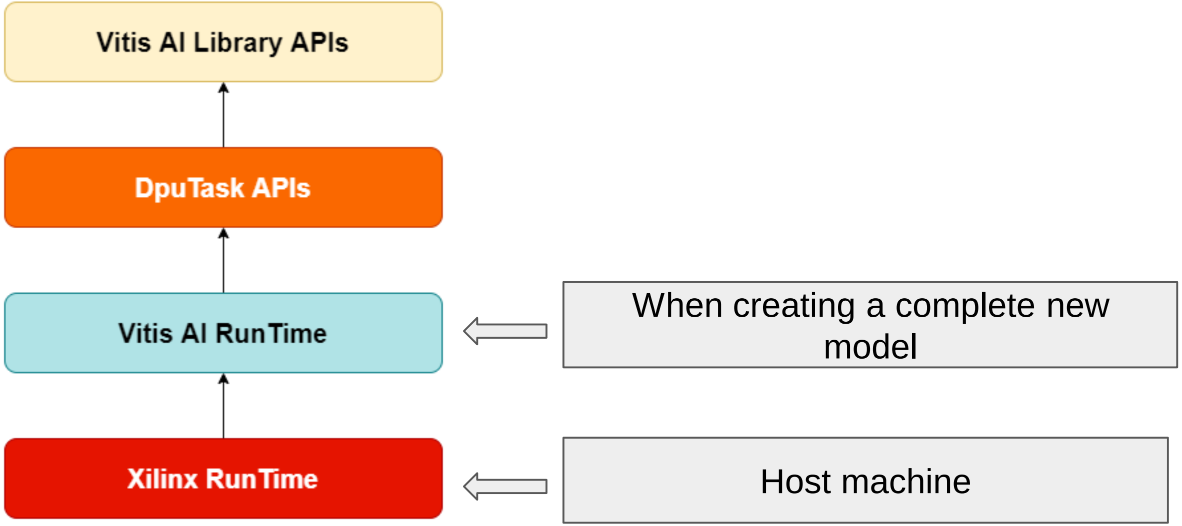
5. Deployment: API Levels



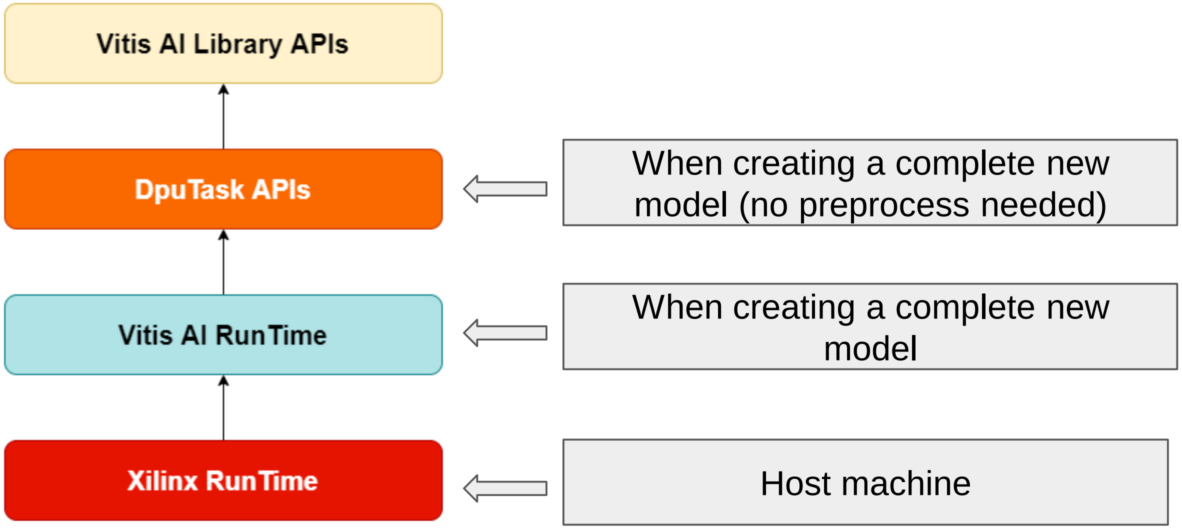
5. Deployment: API Levels



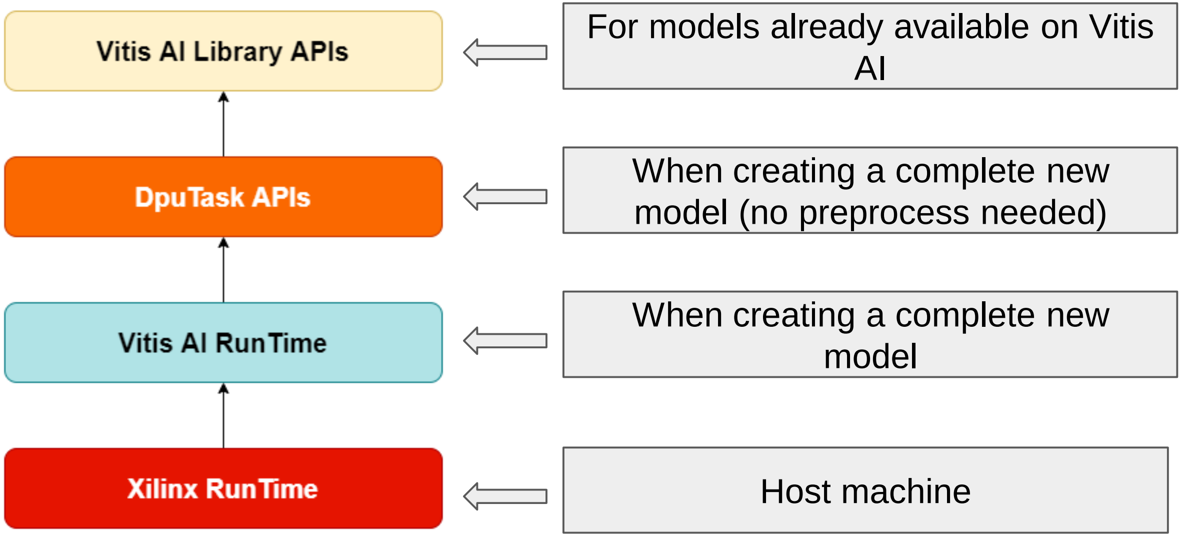
5. Deployment: API Levels



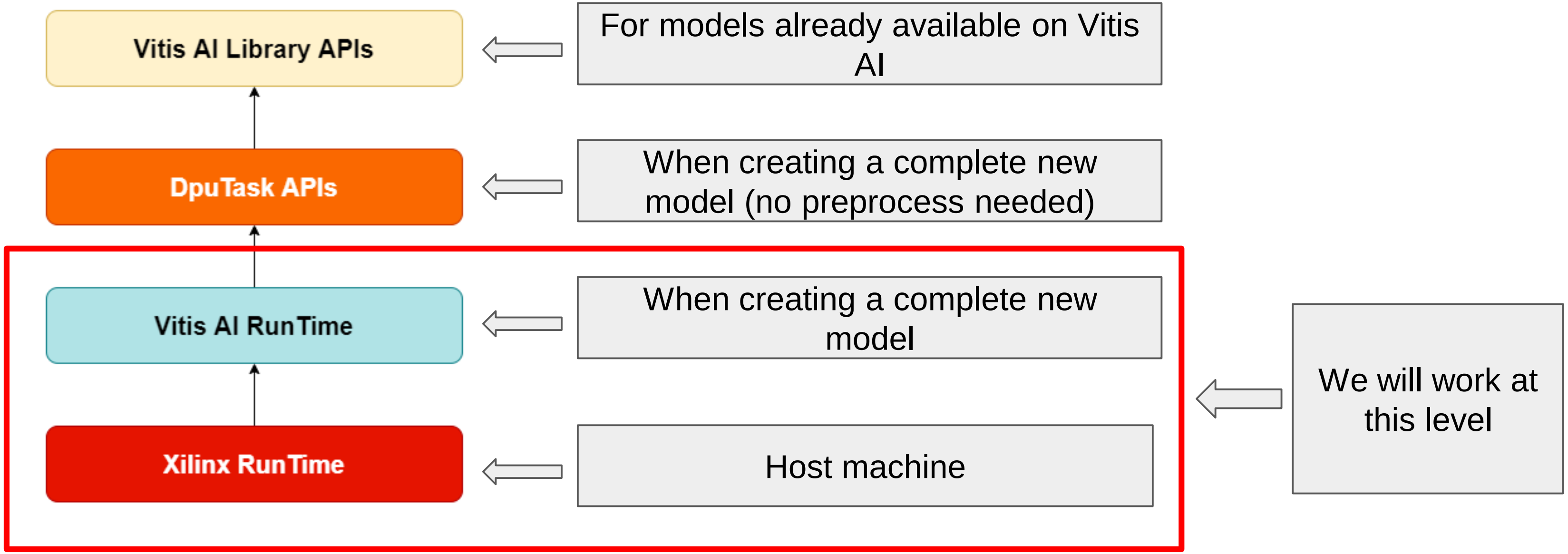
5. Deployment: API Levels



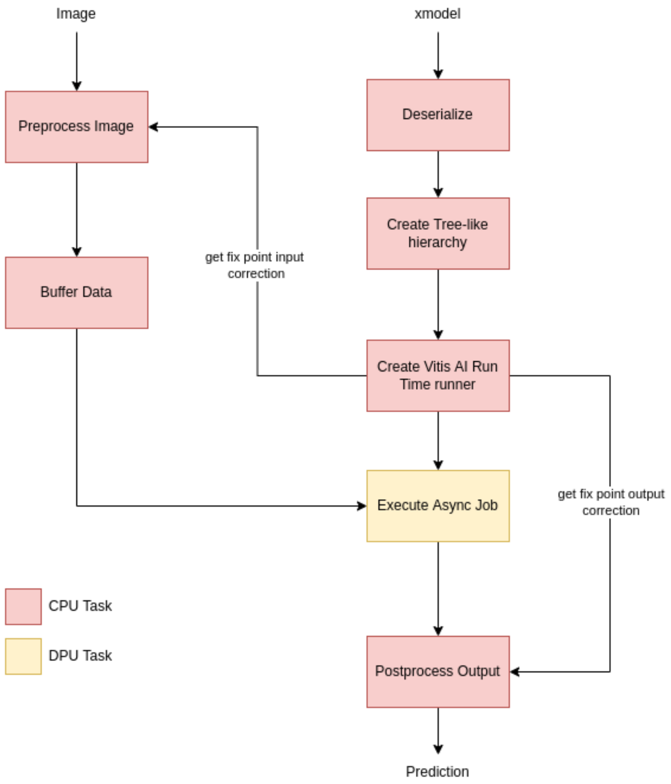
5. Deployment: API Levels



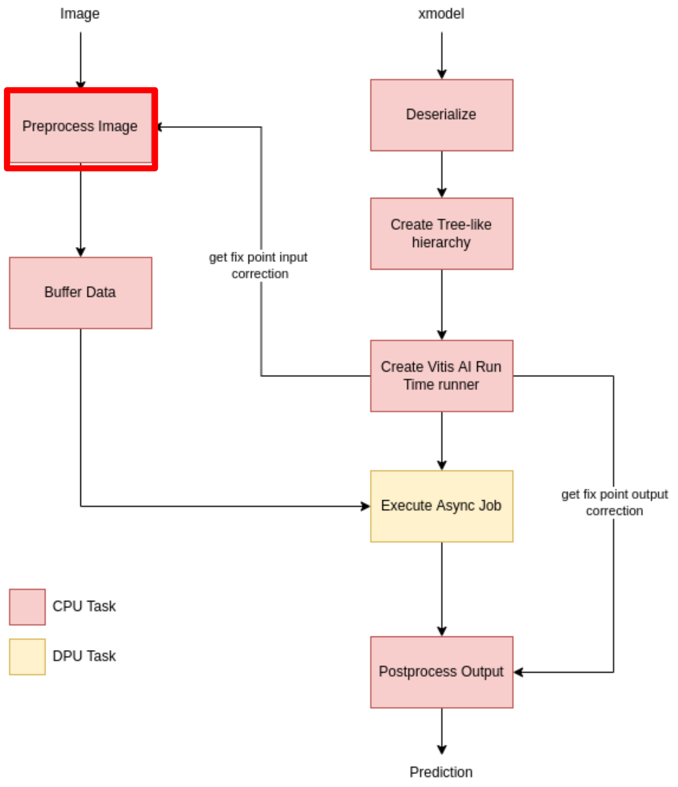
5. Deployment: API Levels



5. Deployment: Application

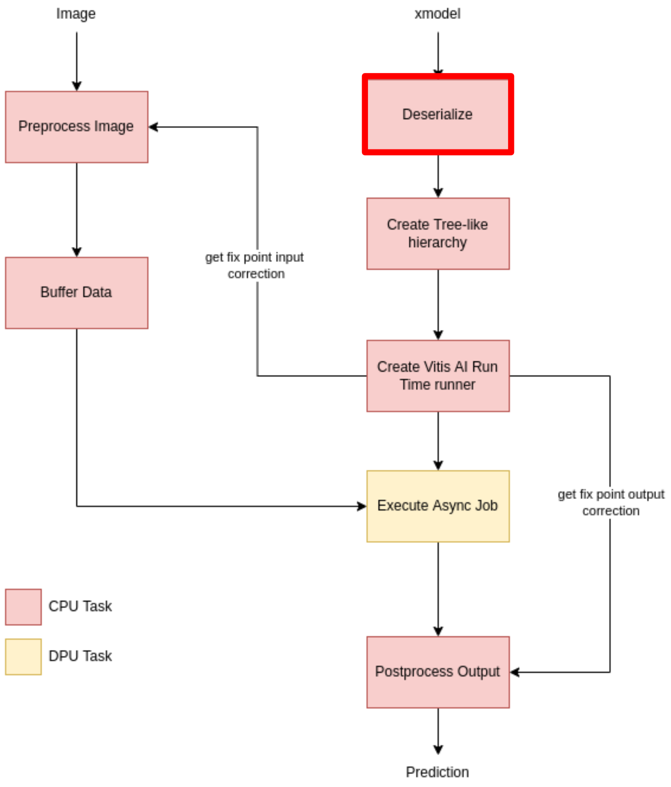


5. Deployment: Application



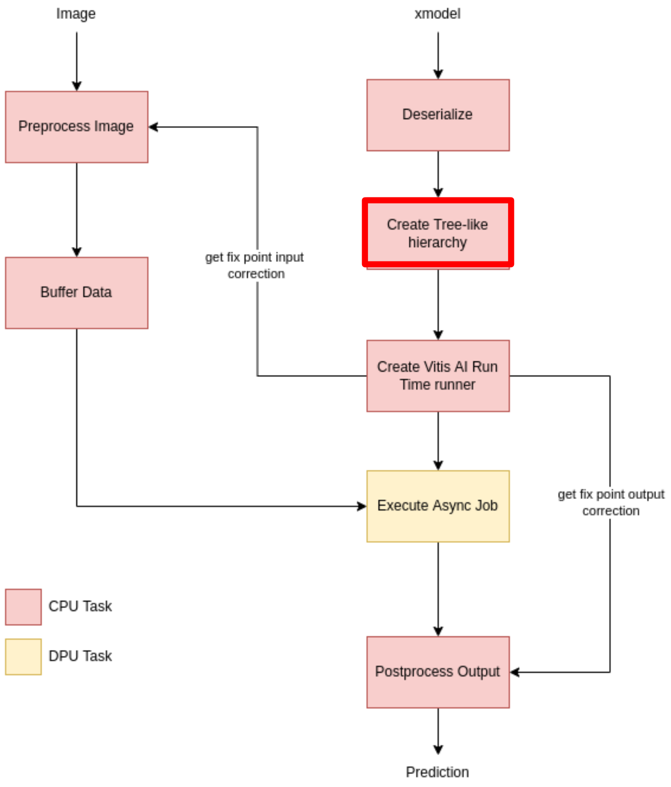

```python
from PIL import Image
import vart
import xir
import numpy as np

def preprocess_fn(image_path):
    with Image.open(image_path) as img:
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img = img.resize((227, 227), Image.NEAREST)

    data = np.asarray(img, dtype="float32" )
    data = data/255.0
    return data

x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')

model = '../compilation/compiled_model/deploy.xmodel'

g = xir.Graph.deserialize(model)

root_subgraph = g.get_root_subgraph()

child_subgraph = root_subgraph.toposort_child_subgraph()

dpu = vart.Runner.create_runner(child_subgraph[1], 'run')

inputTensors = dpu.get_input_tensors()
outputTensors = dpu.get_output_tensors()
input_ndim = tuple(inputTensors[0].dims)
output_ndim = tuple(outputTensors[0].dims)


input_fixpos = inputTensors[0].get_attr("fix_point")
output_fixpos = outputTensors[0].get_attr("fix_point")

outputData = []
inputData = []

inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]

imageRun = inputData[0]

imageRun[0] = x0*(2**input_fixpos-1)
imageRun[1] = x1*(2**input_fixpos-1)
imageRun[2] = x2*(2**input_fixpos-1)
imageRun[3] = x3*(2**input_fixpos-1)

job_id = dpu.execute_async(inputData, outputData)
dpu.wait(job_id)

result = outputData[0]*(2**(output_fixpos)-1)
print(result.astype('uint8'))
```

5. Deployment: Application



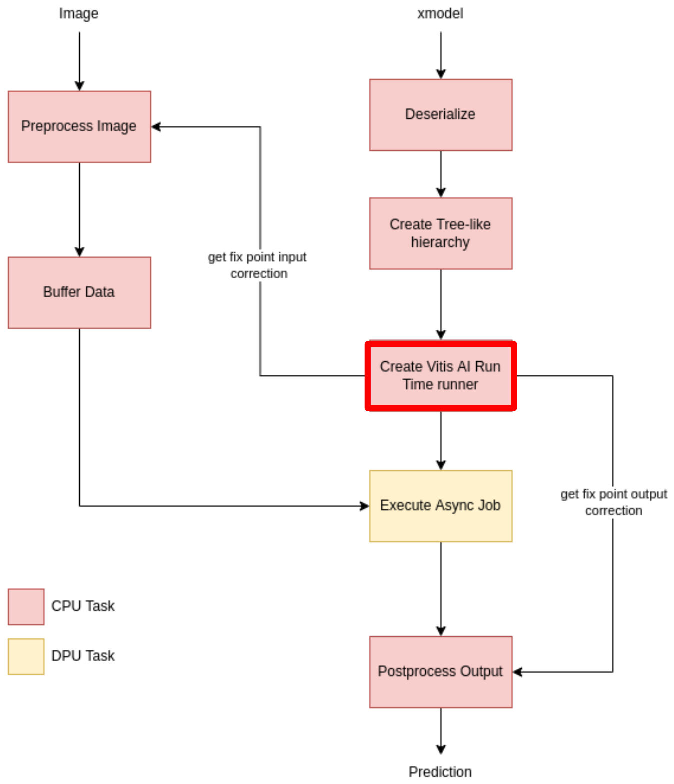

```python
from PIL import Image
import vart
import xir
import numpy as np

def preprocess_fn(image_path):
    with Image.open(image_path) as img:
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img = img.resize((227, 227), Image.NEAREST)

    data = np.asarray(img, dtype="float32" )
    data = data/255.0
    return data

x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')

model = '../compilation/compiled_model/deploy.xmodel'

g = xir.Graph.deserialize(model)

root_subgraph = g.get_root_subgraph()

child_subgraph = root_subgraph.toposort_child_subgraph()

dpu = vart.Runner.create_runner(child_subgraph[1], 'run')

inputTensors = dpu.get_input_tensors()
outputTensors = dpu.get_output_tensors()
input_ndim = tuple(inputTensors[0].dims)
output_ndim = tuple(outputTensors[0].dims)


input_fixpos = inputTensors[0].get_attr("fix_point")
output_fixpos = outputTensors[0].get_attr("fix_point")

outputData = []
inputData = []

inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]

imageRun = inputData[0]

imageRun[0] = x0*(2**input_fixpos-1)
imageRun[1] = x1*(2**input_fixpos-1)
imageRun[2] = x2*(2**input_fixpos-1)
imageRun[3] = x3*(2**input_fixpos-1)

job_id = dpu.execute_async(inputData, outputData)
dpu.wait(job_id)

result = outputData[0]*(2**(output_fixpos)-1)
print(result.astype('uint8'))
```

5. Deployment: Application



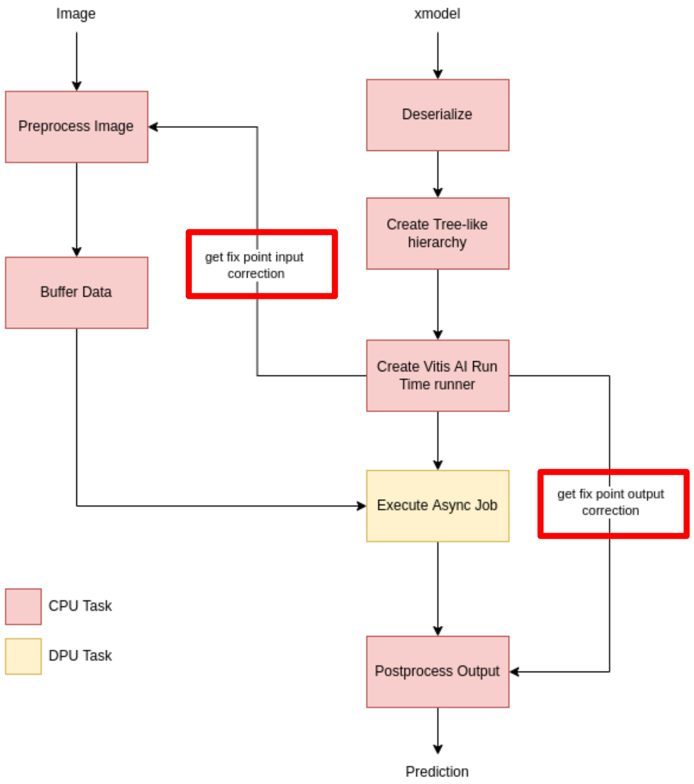

```python
from PIL import Image
import vart
import xir
import numpy as np

def preprocess_fn(image_path):
    with Image.open(image_path) as img:
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img = img.resize((227, 227), Image.NEAREST)

    data = np.asarray(img, dtype="float32" )
    data = data/255.0
    return data

x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')

model = '../compilation/compiled_model/deploy.xmodel'

g = xir.Graph.deserialize(model)

root_subgraph = g.get_root_subgraph()

child_subgraph = root_subgraph.toposort_child_subgraph()

dpu = vart.Runner.create_runner(child_subgraph[1], 'run')

inputTensors = dpu.get_input_tensors()
outputTensors = dpu.get_output_tensors()
input_ndim = tuple(inputTensors[0].dims)
output_ndim = tuple(outputTensors[0].dims)


input_fixpos = inputTensors[0].get_attr("fix_point")
output_fixpos = outputTensors[0].get_attr("fix_point")

outputData = []
inputData = []

inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]

imageRun = inputData[0]

imageRun[0] = x0*(2**input_fixpos-1)
imageRun[1] = x1*(2**input_fixpos-1)
imageRun[2] = x2*(2**input_fixpos-1)
imageRun[3] = x3*(2**input_fixpos-1)

job_id = dpu.execute_async(inputData, outputData)
dpu.wait(job_id)

result = outputData[0]*(2**(output_fixpos)-1)
print(result.astype('uint8'))
```

5. Deployment: Application



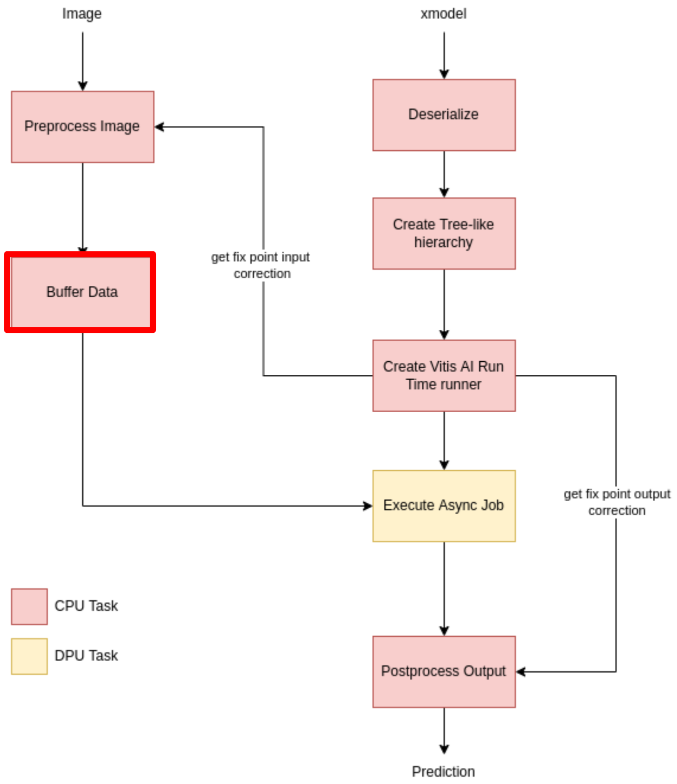

```python
from PIL import Image
import vart
import xir
import numpy as np

def preprocess_fn(image_path):
    with Image.open(image_path) as img:
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img = img.resize((227, 227), Image.NEAREST)

    data = np.asarray(img, dtype="float32" )
    data = data/255.0
    return data

x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')

model = '../compilation/compiled_model/deploy.xmodel'

g = xir.Graph.deserialize(model)

root_subgraph = g.get_root_subgraph()

child_subgraph = root_subgraph.toposort_child_subgraph()

dpu = vart.Runner.create_runner(child_subgraph[1], 'run')

inputTensors = dpu.get_input_tensors()
outputTensors = dpu.get_output_tensors()
input_ndim = tuple(inputTensors[0].dims)
output_ndim = tuple(outputTensors[0].dims)


input_fixpos = inputTensors[0].get_attr("fix_point")
output_fixpos = outputTensors[0].get_attr("fix_point")

outputData = []
inputData = []

inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]

imageRun = inputData[0]

imageRun[0] = x0*(2**input_fixpos-1)
imageRun[1] = x1*(2**input_fixpos-1)
imageRun[2] = x2*(2**input_fixpos-1)
imageRun[3] = x3*(2**input_fixpos-1)

job_id = dpu.execute_async(inputData, outputData)
dpu.wait(job_id)

result = outputData[0]*(2**(output_fixpos)-1)
print(result.astype('uint8'))
```

5. Deployment: Application




```python
from PIL import Image
import vart
import xir
import numpy as np

def preprocess_fn(image_path):
    with Image.open(image_path) as img:
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img = img.resize((227, 227), Image.NEAREST)

    data = np.asarray(img, dtype="float32" )
    data = data/255.0
    return data

x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')

model = '../compilation/compiled_model/deploy.xmodel'

g = xir.Graph.deserialize(model)

root_subgraph = g.get_root_subgraph()

child_subgraph = root_subgraph.toposort_child_subgraph()

dpu = vart.Runner.create_runner(child_subgraph[1], 'run')

inputTensors = dpu.get_input_tensors()
outputTensors = dpu.get_output_tensors()
input_ndim = tuple(inputTensors[0].dims)
output_ndim = tuple(outputTensors[0].dims)


input_fixpos = inputTensors[0].get_attr("fix_point")
output_fixpos = outputTensors[0].get_attr("fix_point")

outputData = []
inputData = []

inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]

imageRun = inputData[0]

imageRun[0] = x0*(2**input_fixpos-1)
imageRun[1] = x1*(2**input_fixpos-1)
imageRun[2] = x2*(2**input_fixpos-1)
imageRun[3] = x3*(2**input_fixpos-1)

job_id = dpu.execute_async(inputData, outputData)
dpu.wait(job_id)

result = outputData[0]*(2**(output_fixpos)-1)
print(result.astype('uint8'))
```

5. Deployment: Application

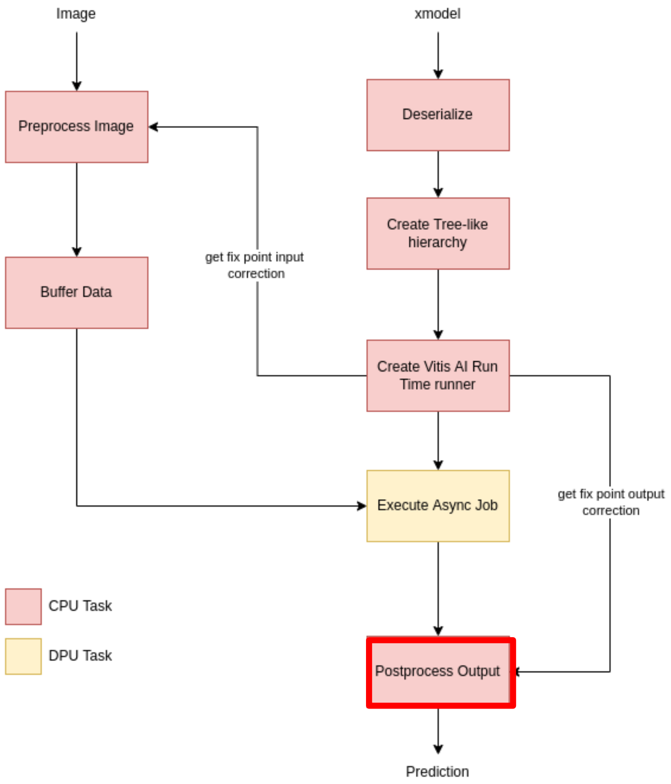



```python
from PIL import Image
import vart
import xir
import numpy as np

def preprocess_fn(image_path):
    with Image.open(image_path) as img:
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img = img.resize((227, 227), Image.NEAREST)

    data = np.asarray(img, dtype="float32" )
    data = data/255.0
    return data

x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')

model = '../compilation/compiled_model/deploy.xmodel'

g = xir.Graph.deserialize(model)

root_subgraph = g.get_root_subgraph()

child_subgraph = root_subgraph.toposort_child_subgraph()

dpu = vart.Runner.create_runner(child_subgraph[1], 'run')

inputTensors = dpu.get_input_tensors()
outputTensors = dpu.get_output_tensors()
input_ndim = tuple(inputTensors[0].dims)
output_ndim = tuple(outputTensors[0].dims)


input_fixpos = inputTensors[0].get_attr("fix_point")
output_fixpos = outputTensors[0].get_attr("fix_point")

outputData = []
inputData = []

inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]

imageRun = inputData[0]

imageRun[0] = x0*(2**input_fixpos-1)
imageRun[1] = x1*(2**input_fixpos-1)
imageRun[2] = x2*(2**input_fixpos-1)
imageRun[3] = x3*(2**input_fixpos-1)

job_id = dpu.execute_async(inputData, outputData)
dpu.wait(job_id)

result = outputData[0]*(2**(output_fixpos)-1)
print(result.astype('uint8'))
```

5. Deployment: Application


```python
from PIL import Image
import vart
import xir
import numpy as np

def preprocess_fn(image_path):
    with Image.open(image_path) as img:
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img = img.resize((227, 227), Image.NEAREST)

    data = np.asarray(img, dtype="float32" )
    data = data/255.0
    return data

x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')

model = '../compilation/compiled_model/deploy.xmodel'

g = xir.Graph.deserialize(model)

root_subgraph = g.get_root_subgraph()

child_subgraph = root_subgraph.toposort_child_subgraph()

dpu = vart.Runner.create_runner(child_subgraph[1], 'run')

inputTensors = dpu.get_input_tensors()
outputTensors = dpu.get_output_tensors()
input_ndim = tuple(inputTensors[0].dims)
output_ndim = tuple(outputTensors[0].dims)


input_fixpos = inputTensors[0].get_attr("fix_point")
output_fixpos = outputTensors[0].get_attr("fix_point")

outputData = []
inputData = []

inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]

imageRun = inputData[0]

imageRun[0] = x0*(2**input_fixpos-1)
imageRun[1] = x1*(2**input_fixpos-1)
imageRun[2] = x2*(2**input_fixpos-1)
imageRun[3] = x3*(2**input_fixpos-1)

job_id = dpu.execute_async(inputData, outputData)
dpu.wait(job_id)

result = outputData[0]*(2**(output_fixpos)-1)
print(result.astype('uint8'))
```

5. Deployment: Application




```python
from PIL import Image
import vart
import xir
import numpy as np

def preprocess_fn(image_path):
    with Image.open(image_path) as img:
        if img.mode != 'RGB':
            img = img.convert('RGB')
        img = img.resize((227, 227), Image.NEAREST)

    data = np.asarray(img, dtype="float32" )
    data = data/255.0
    return data

x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')

model = '../compilation/compiled_model/deploy.xmodel'

g = xir.Graph.deserialize(model)

root_subgraph = g.get_root_subgraph()

child_subgraph = root_subgraph.toposort_child_subgraph()

dpu = vart.Runner.create_runner(child_subgraph[1], 'run')

inputTensors = dpu.get_input_tensors()
outputTensors = dpu.get_output_tensors()
input_ndim = tuple(inputTensors[0].dims)
output_ndim = tuple(outputTensors[0].dims)


input_fixpos = inputTensors[0].get_attr("fix_point")
output_fixpos = outputTensors[0].get_attr("fix_point")

outputData = []
inputData = []

inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]

imageRun = inputData[0]

imageRun[0] = x0*(2**input_fixpos-1)
imageRun[1] = x1*(2**input_fixpos-1)
imageRun[2] = x2*(2**input_fixpos-1)
imageRun[3] = x3*(2**input_fixpos-1)

job_id = dpu.execute_async(inputData, outputData)
dpu.wait(job_id)

result = outputData[0]*(2**(output_fixpos)-1)
print(result.astype('uint8'))
```

5. Deployment: Complete Application

To run a complete implementation

```
cd /tutorial/inference

python inference.py -d ../dataset/test -t 1 -m ../compilation/compiled_model/deploy.xmodel
```

6. Accuracy Check

To check your model implementation accuracy run

```
cd /tutorial/inference

python accuracy_calc.py
```

Thank you!