

Developing Models on Alveo U200

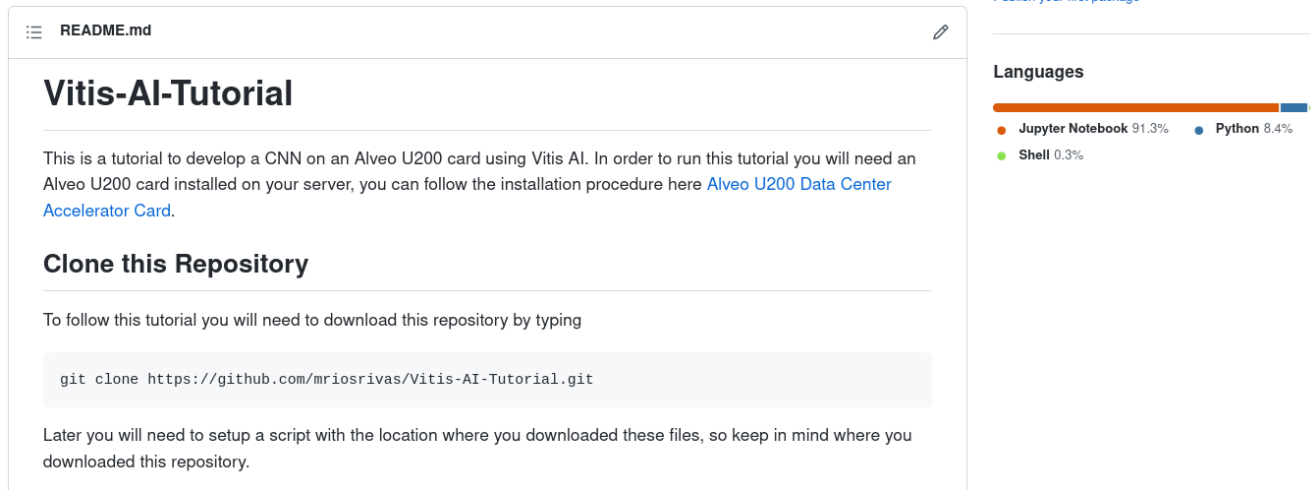
Manuel Rios

GitHub Repository

Clone GitHub Repository

You can download all the source material from here:

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



The screenshot shows the GitHub repository page for 'Vitis-AI-Tutorial'. The main content area displays the README file, which includes the repository title, a description of the tutorial, and the command to clone the repository. The right sidebar shows the 'Languages' section with a horizontal bar chart indicating the distribution of code languages: Jupyter Notebook (91.3%), Python (8.4%), and Shell (0.3%).

Vitis-AI-Tutorial

This is a tutorial to develop a CNN on an Alveo U200 card using Vitis AI. In order to run this tutorial you will need an Alveo U200 card installed on your server, you can follow the installation procedure here [Alveo U200 Data Center Accelerator Card](#).

Clone this Repository

To follow this tutorial you will need to download this repository by typing

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

Later you will need to setup a script with the location where you downloaded these files, so keep in mind where you downloaded this repository.

Languages

- Jupyter Notebook 91.3%
- Python 8.4%
- Shell 0.3%

Alveo U200

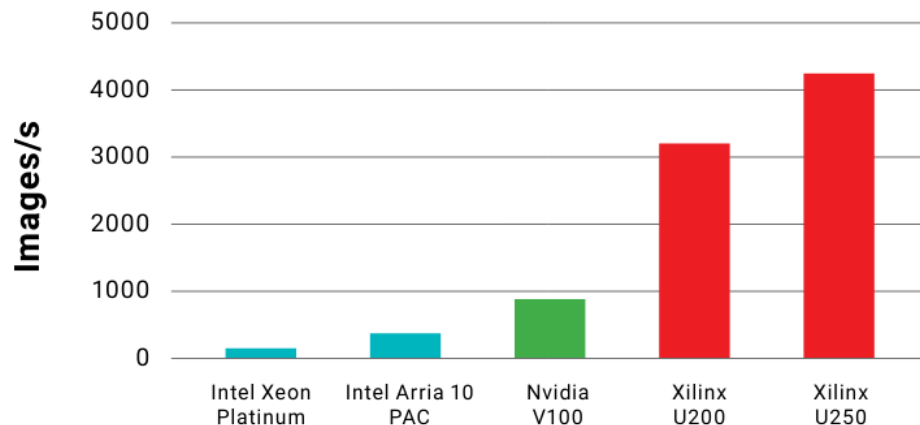
Alveo U200



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

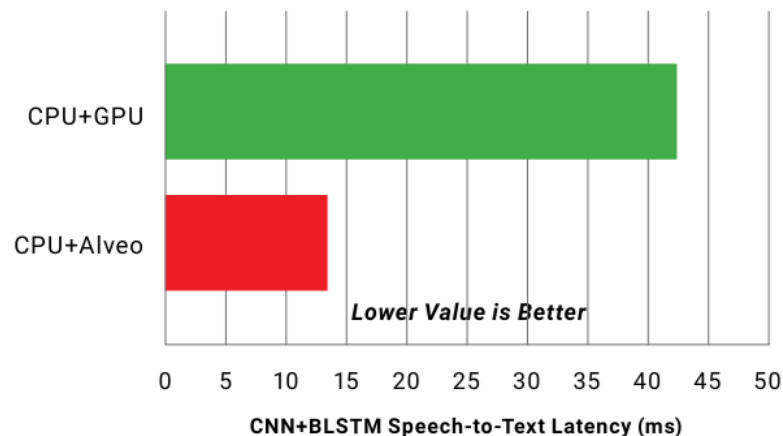
Alveo U200

Increase Real-Time Machine Learning* Throughput by 20X



[*GoogleNet V1: Accelerating DNNs with Xilinx Alveo Accelerator Cards White Paper](#)

Reduce ML Inference Latency by 3X



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

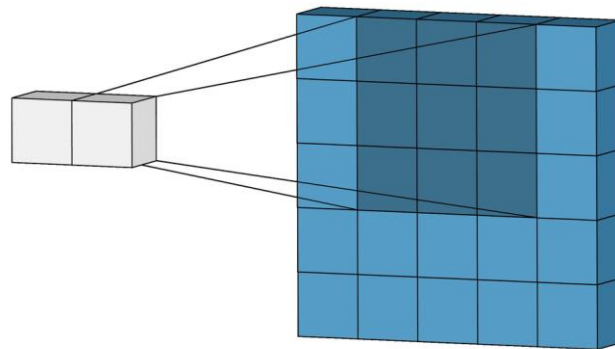
Highlights of DPUCADF8H functionality include:

- 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:

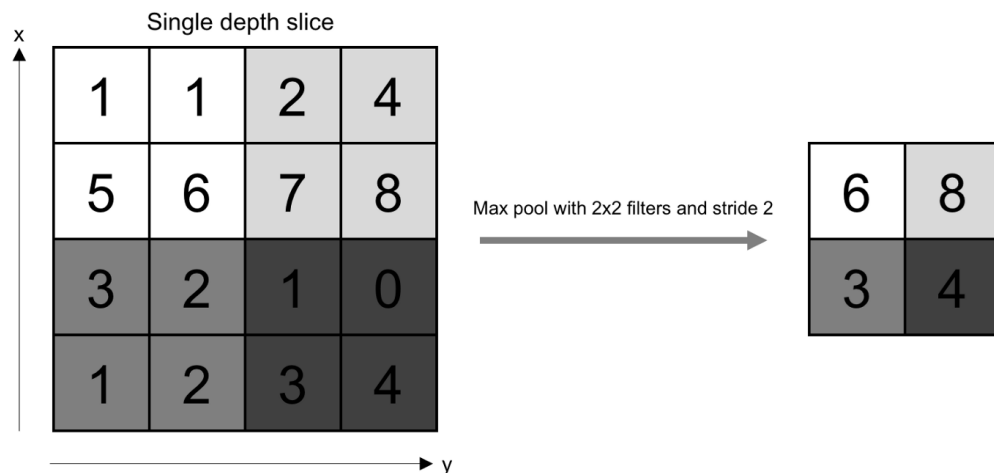
- **Convolution, dilated convolution, and deconvolution**
- Maximum and average pooling
- Element wise sum
- ReLU
- Data split and concat
- Data reorganization
- Fully connected layer
- Batch normalization



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

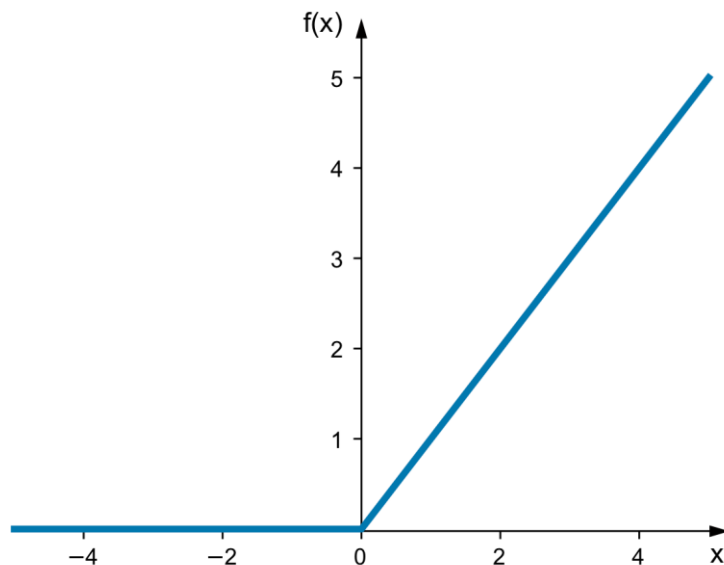
$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$$

Add to each row

Alveo U200 -DPUCADF8H

Highlights of DPUCADF8H functionality include:

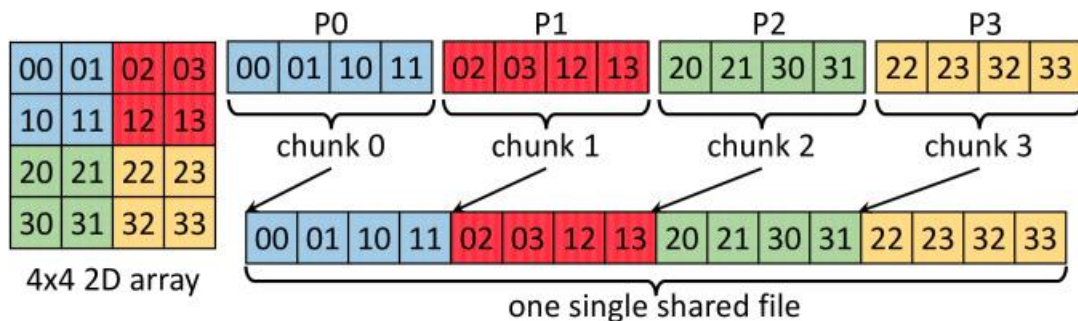
- Convolution, dilated convolution, and deconvolution
- Maximum and average pooling
- Element wise sum
- **ReLU**
- Data split and concat
- Data reorganization
- Fully connected layer
- Batch normalization



Alveo U200 -DPUCADF8H

Highlights of DPUCADF8H functionality include:

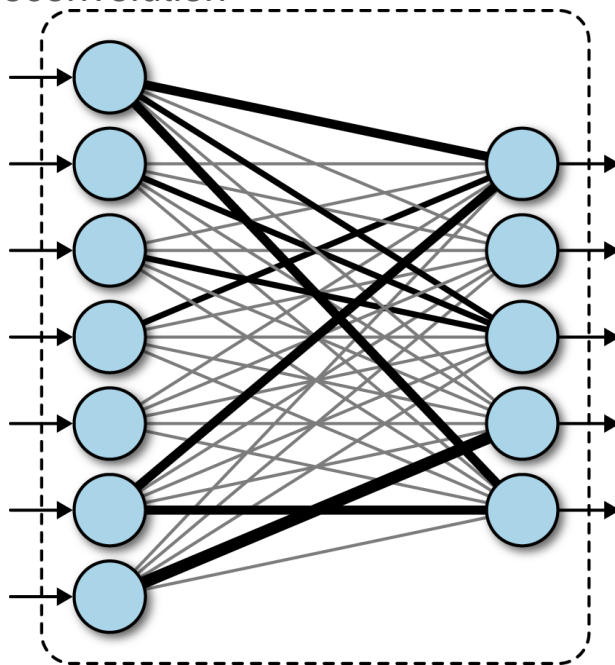
- Convolution, dilated convolution, and deconvolution
- Maximum and average pooling
- Element wise sum
- ReLU
- **Data split and concat**
- **Data reorganization**
- Fully connected layer
- Batch normalization



Alveo U200 -DPUCADF8H

Highlights of DPUCADF8H functionality include:

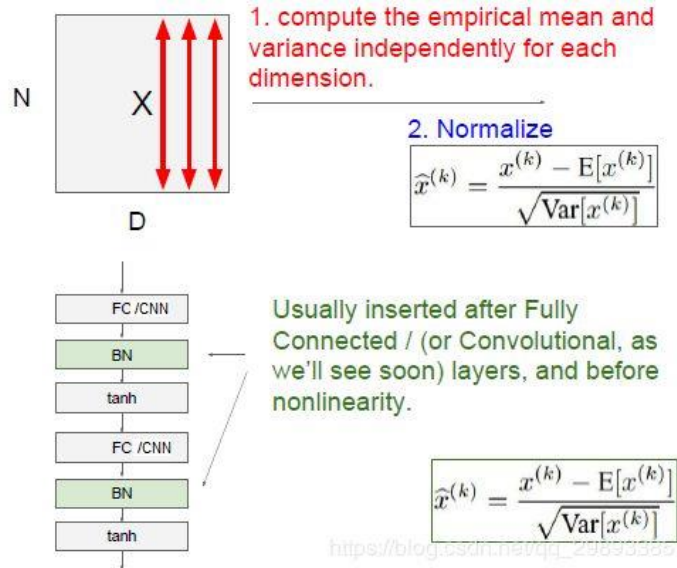
- Convolution, dilated convolution, and deconvolution
- Maximum and average pooling
- Element wise sum
- ReLU
- Data split and concat
- Data reorganization
- **Fully connected layer**
- Batch normalization



Alveo U200 -DPUCADF8H

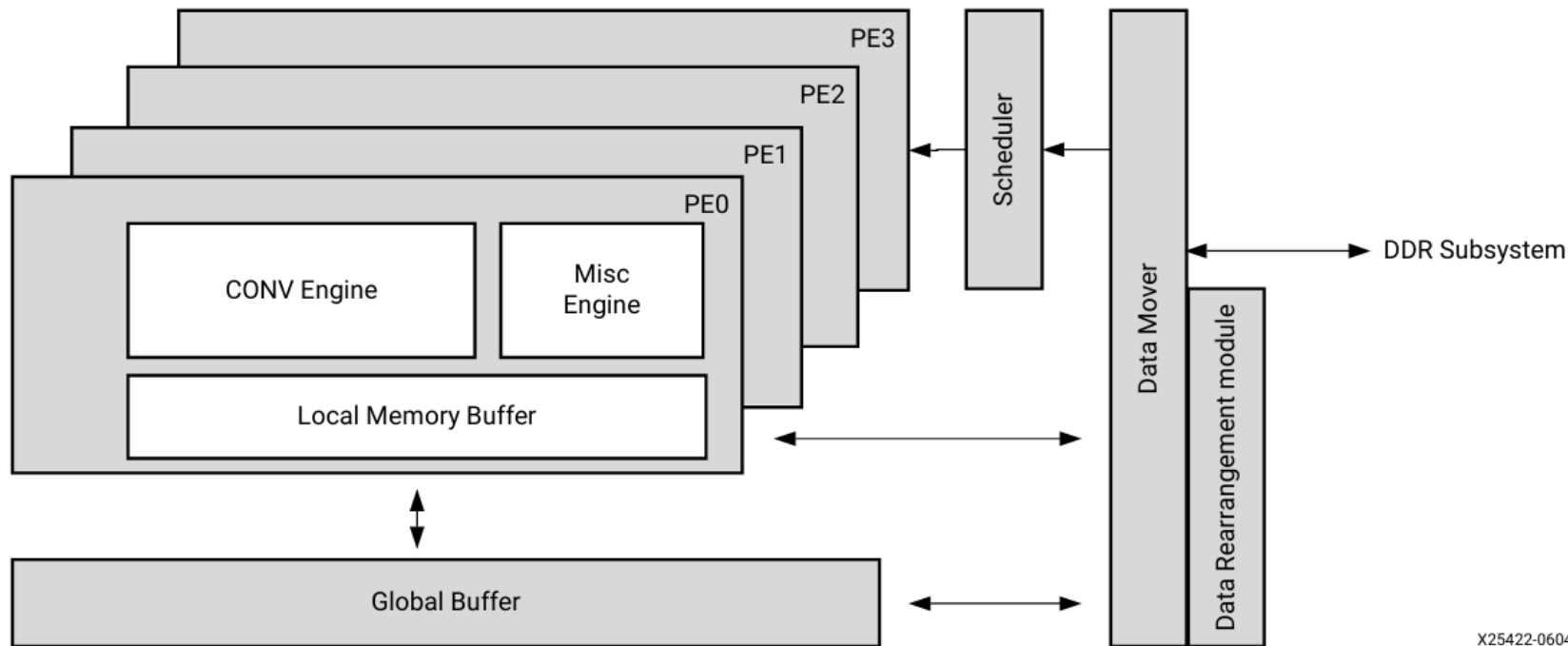
Highlights of DPUCADF8H functionality include:

- 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

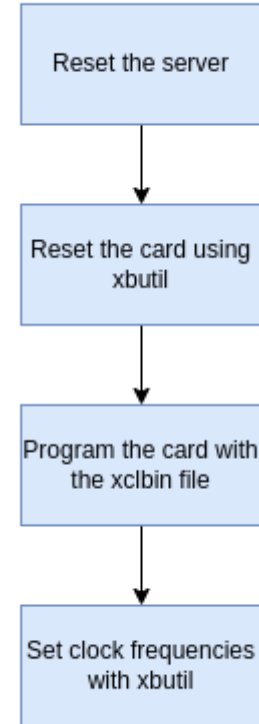
Figure 1: DPUCADF8H Top-Level Block Diagram



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

```
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
```

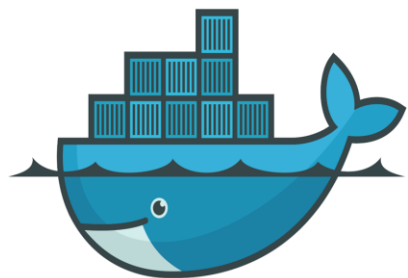
Set the clock frequency

```
xbutil --legacy clock -d 0000:17:00.1 -f 70 -g 70
```

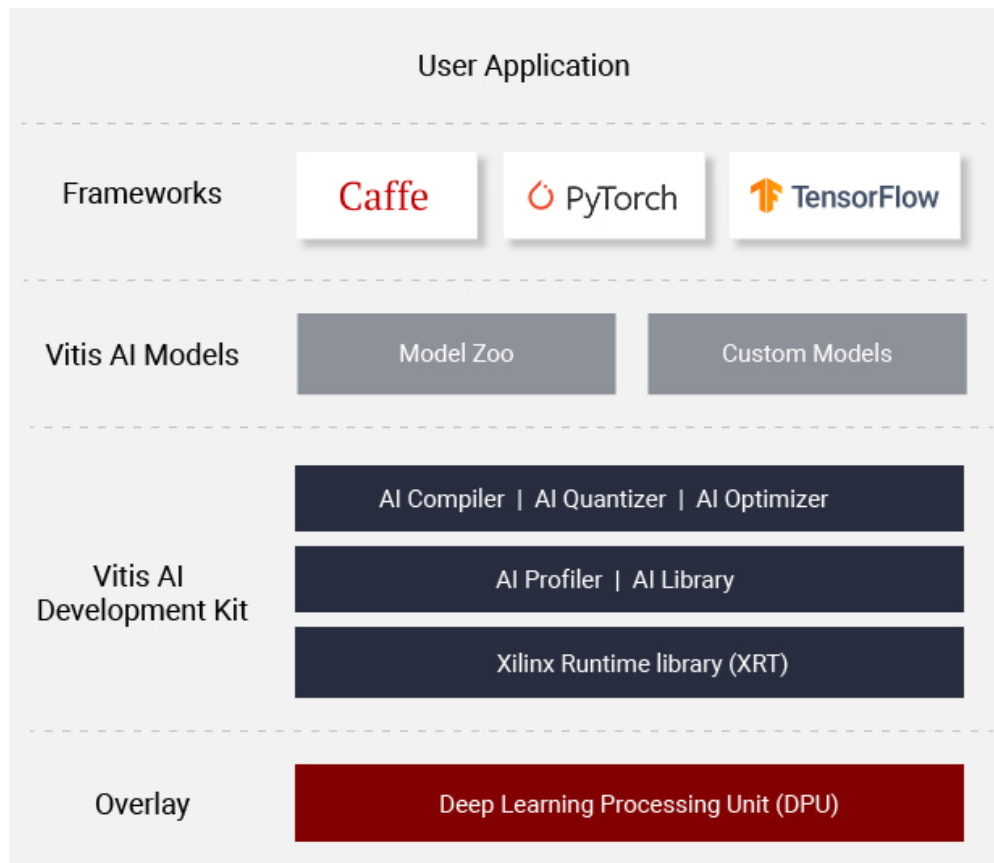
Vitis AI

Vitis AI

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



docker



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**

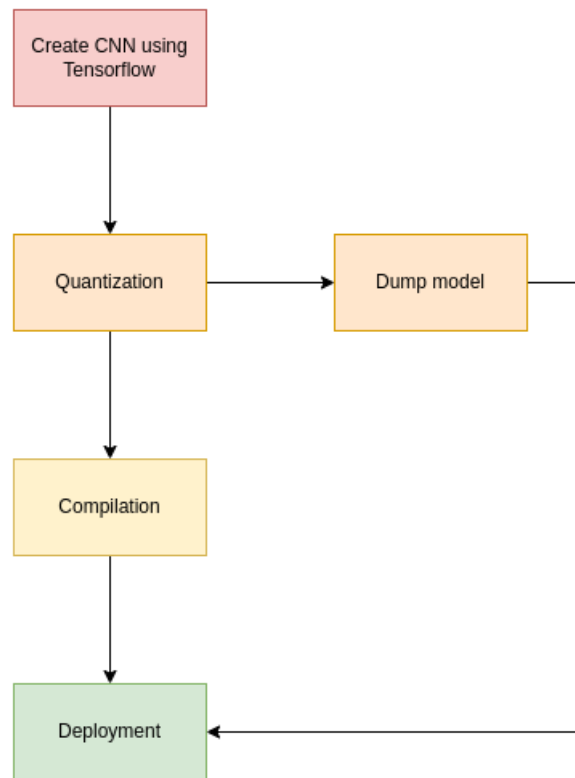
```
conda activate vitis-ai-tensorflow2
```

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

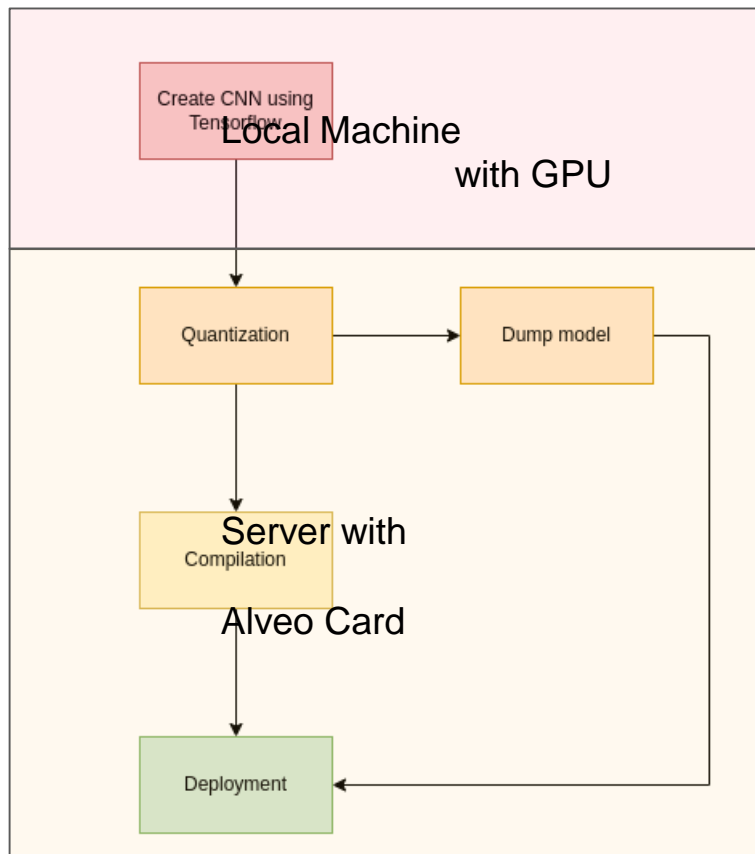
```
source /workspace/setup/alveo/setup.sh DPUCADF8H
```


Workflow

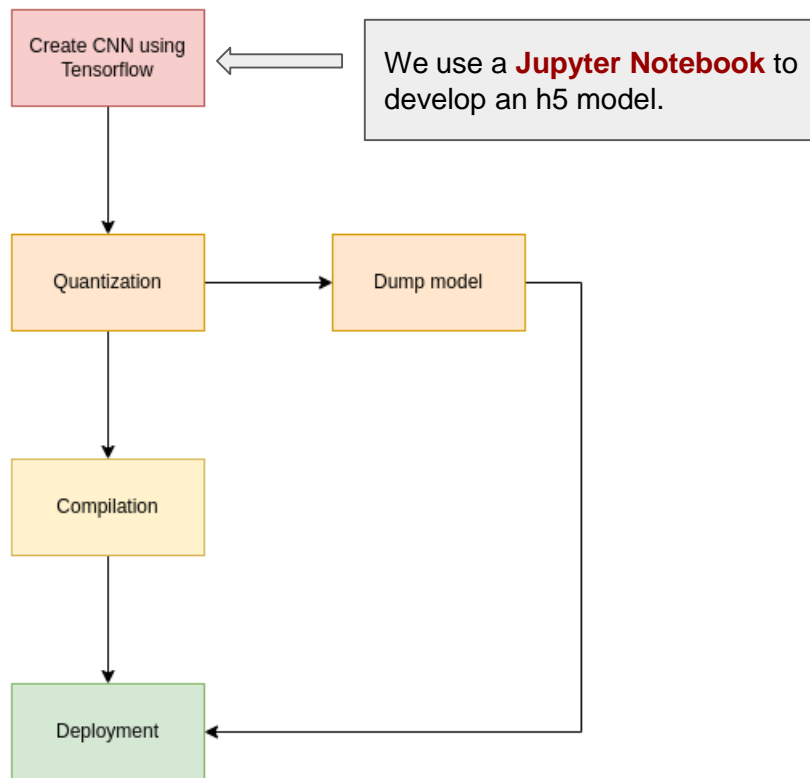
Workflow



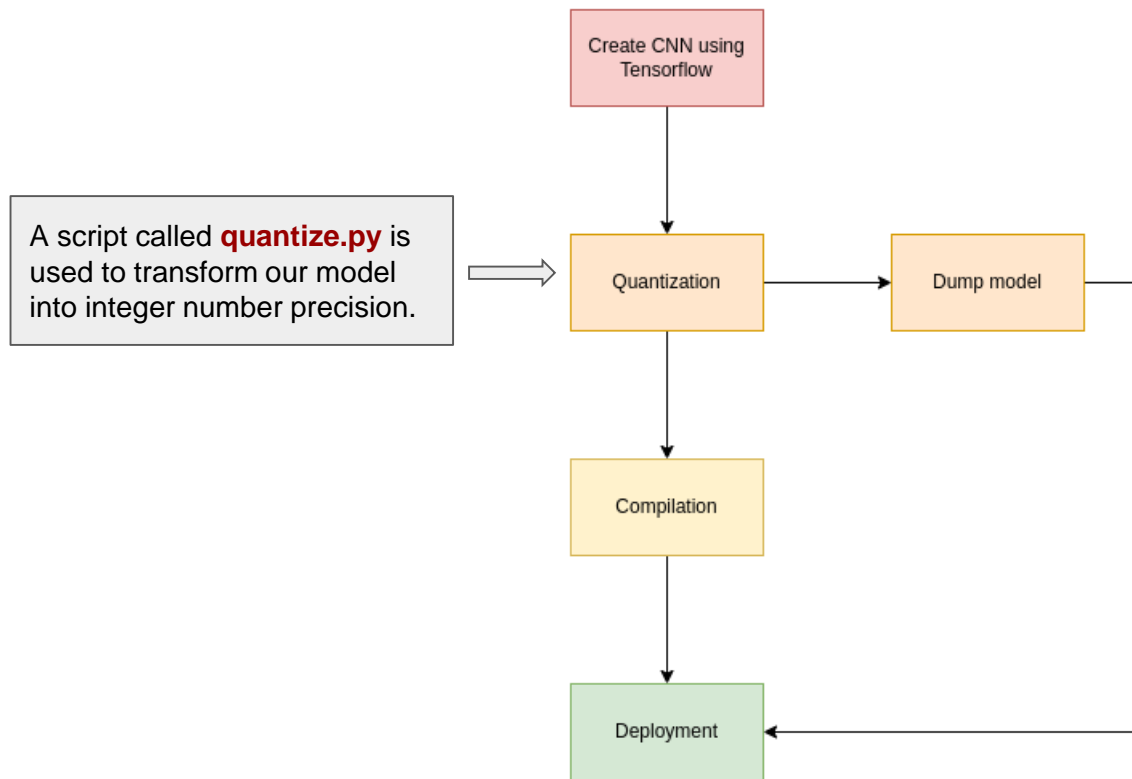
Workflow



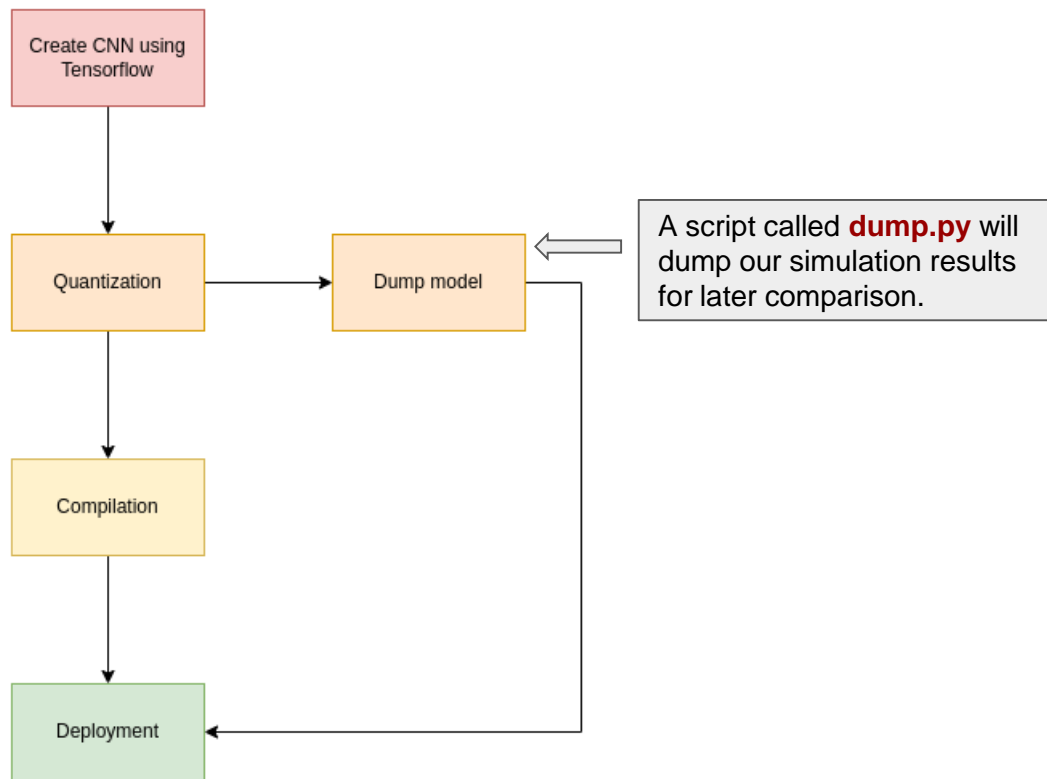
Workflow



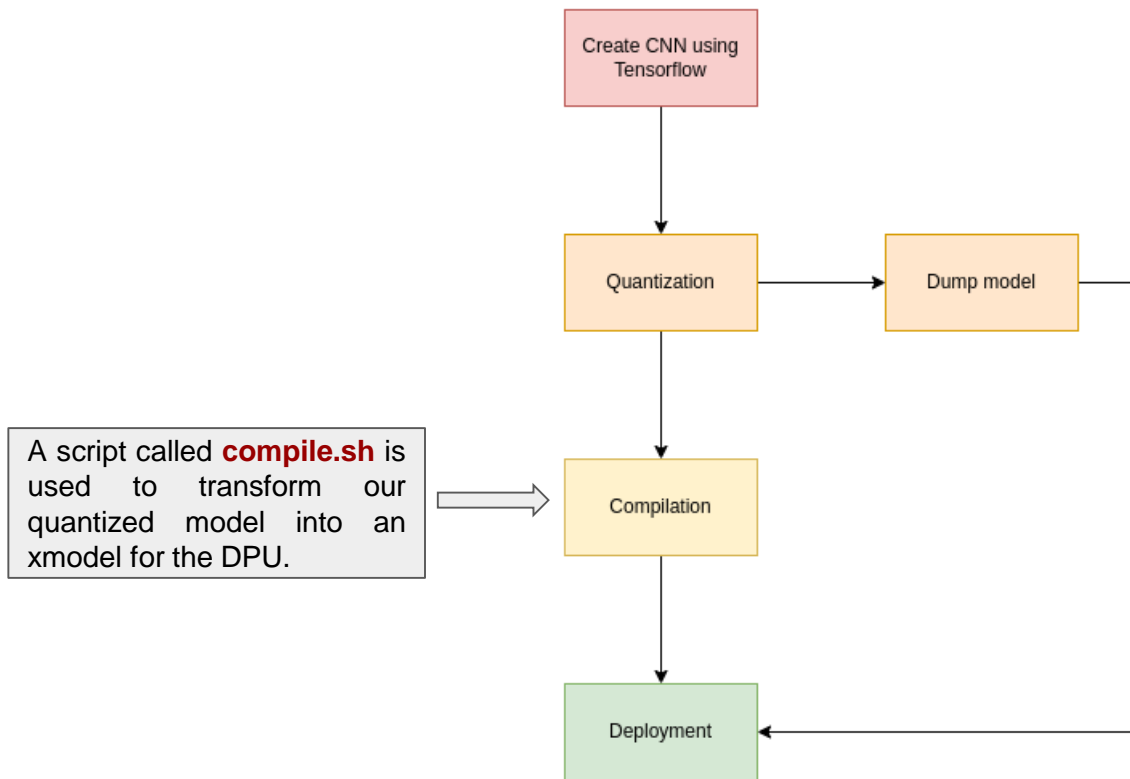
Workflow



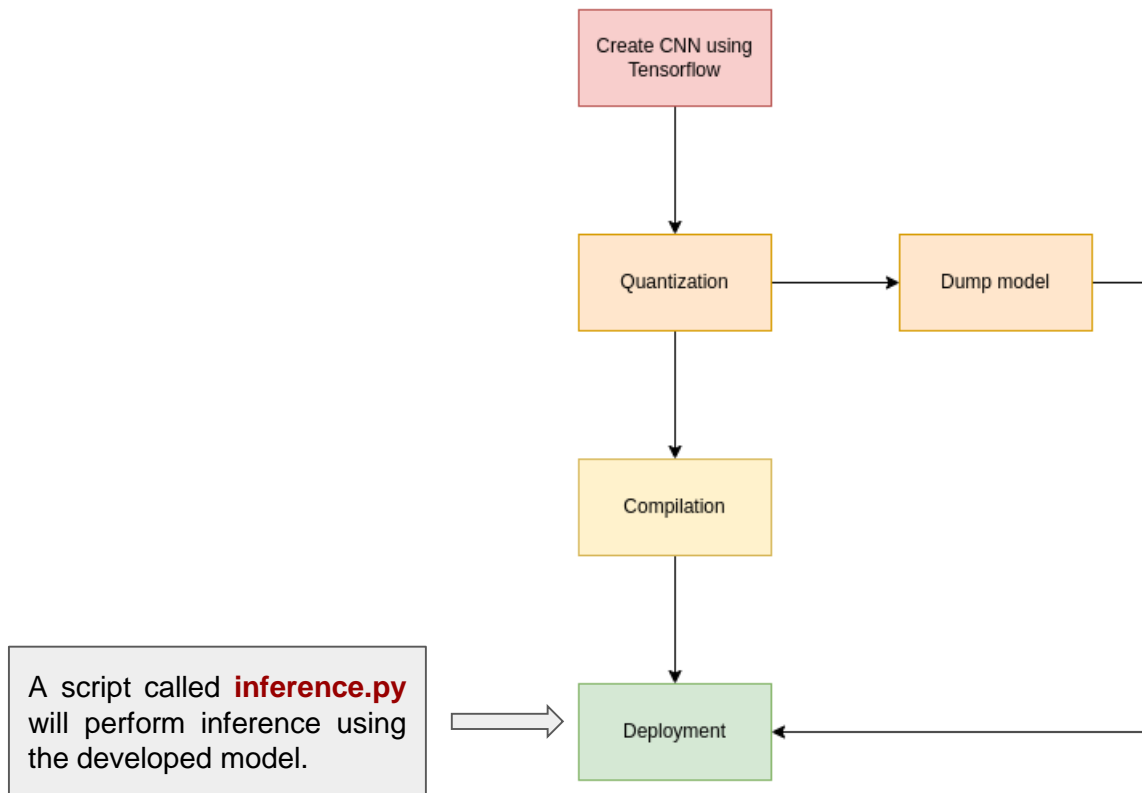
Workflow



Workflow



Workflow



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:

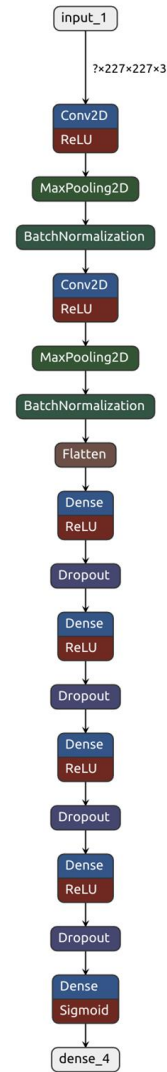
- RMSprop

Loss:

- Binary Cross Entropy

Metric:

- Binary Accuracy



1. Create CNN Using Tensorflow

Basic architecture:

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

Optimizer:

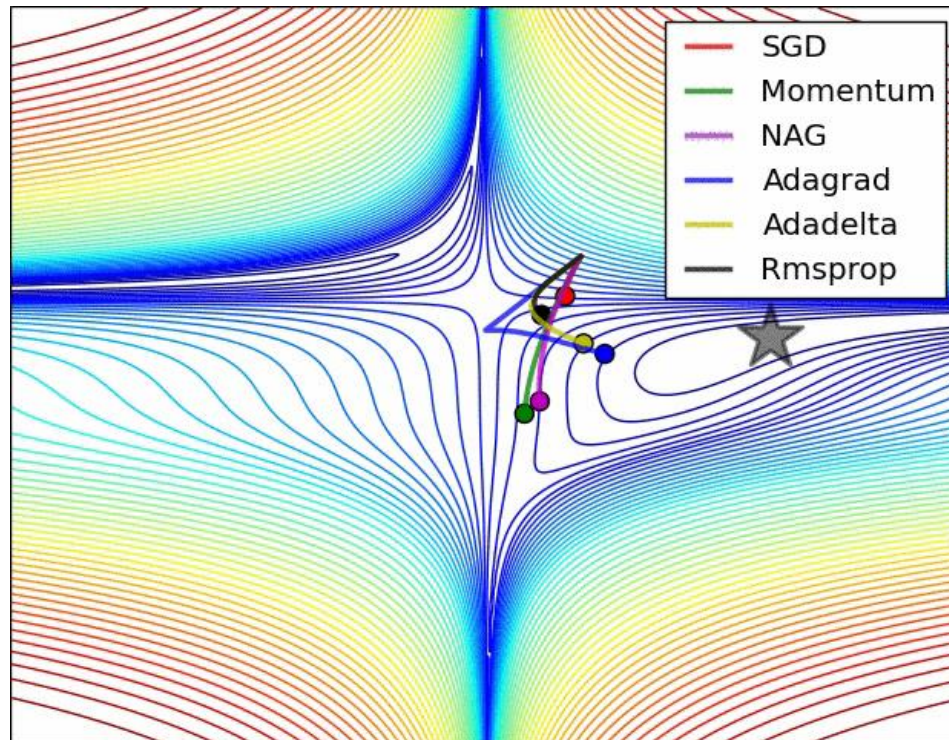
- RMSprop

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

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

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

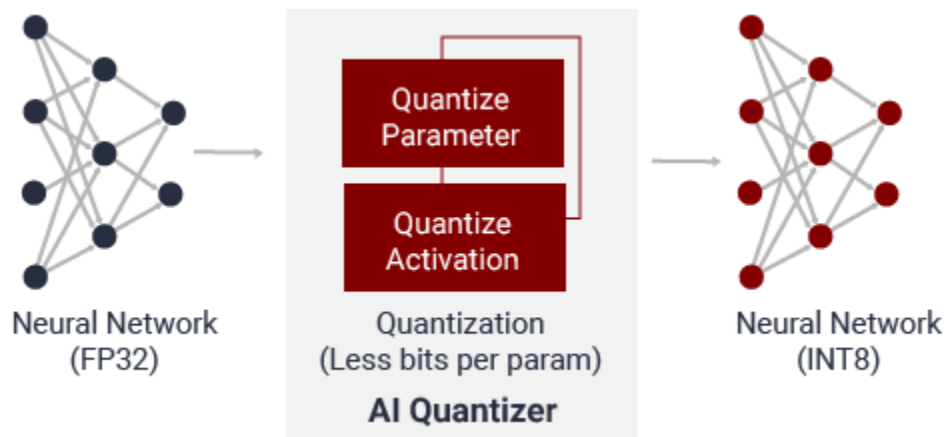
Metric:

- Binary Accuracy

		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)

```
## Quantize the model
quantizer = vitis_quantize.VitisQuantizer('float_model.h5')

quantized_model = quantizer.quantize_model(calib_dataset=train_ds,
                                           calib_batch_size=32,
                                           replace_sigmoid=False,
                                           input_shape="?,227,227,3",
                                           weight_bit=8,
                                           bias_bit=8)

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.

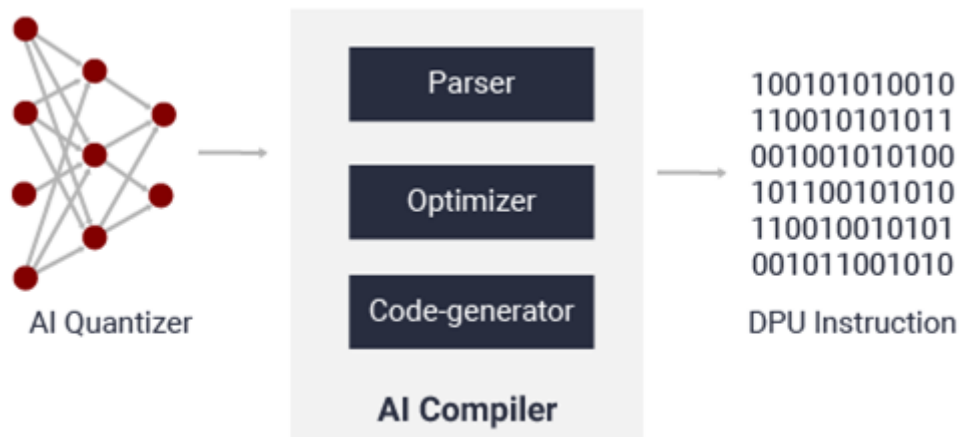
3. Dump Model

[illegible]

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

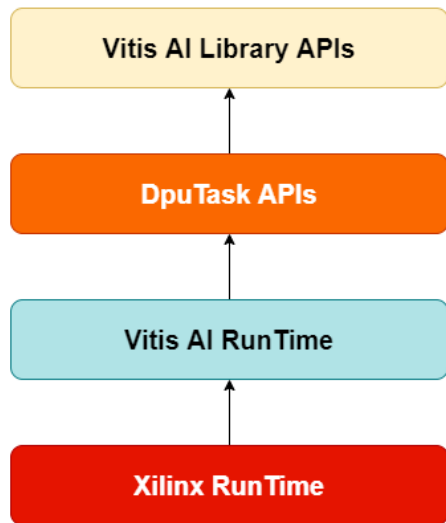
```
vai_c_tensorflow2 \  
  --model quantized_model.h5\  
  --arch /opt/vitis_ai/compiler/arch/DPUCADF8H/U200/arch.json \  
  --output_dir compile_model \  
  --net_name deploy \  
  --options '{"input_shape": "4,227,227,3"}'
```

4. Compile model

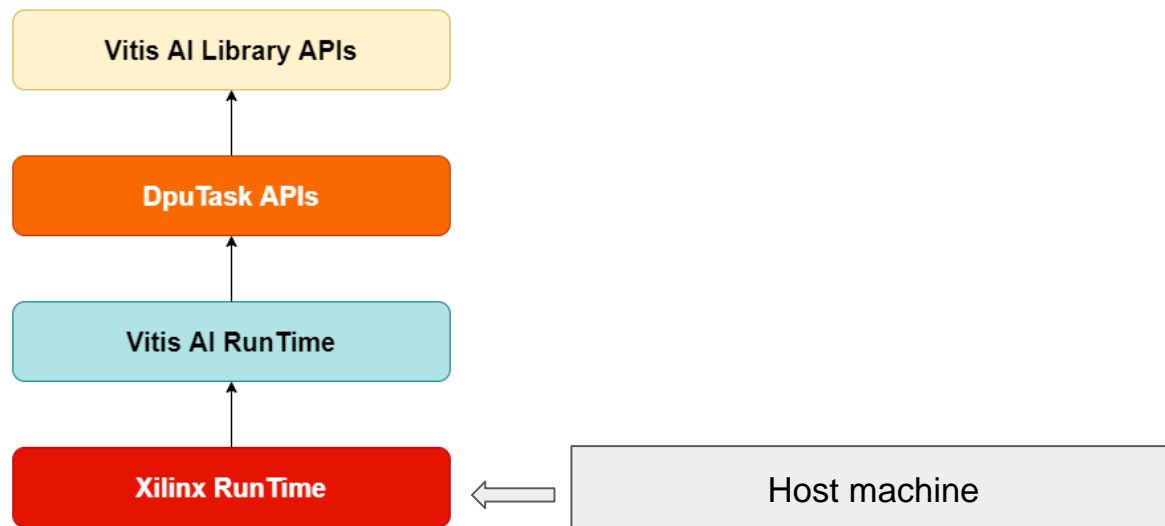
We can plot the model graph using

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

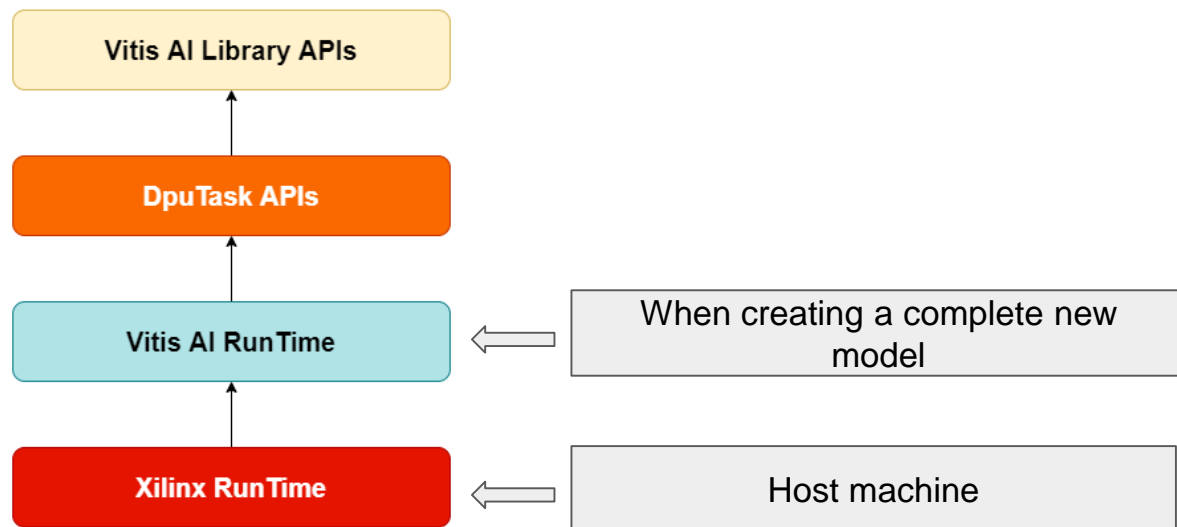
5. Deployment: API Levels



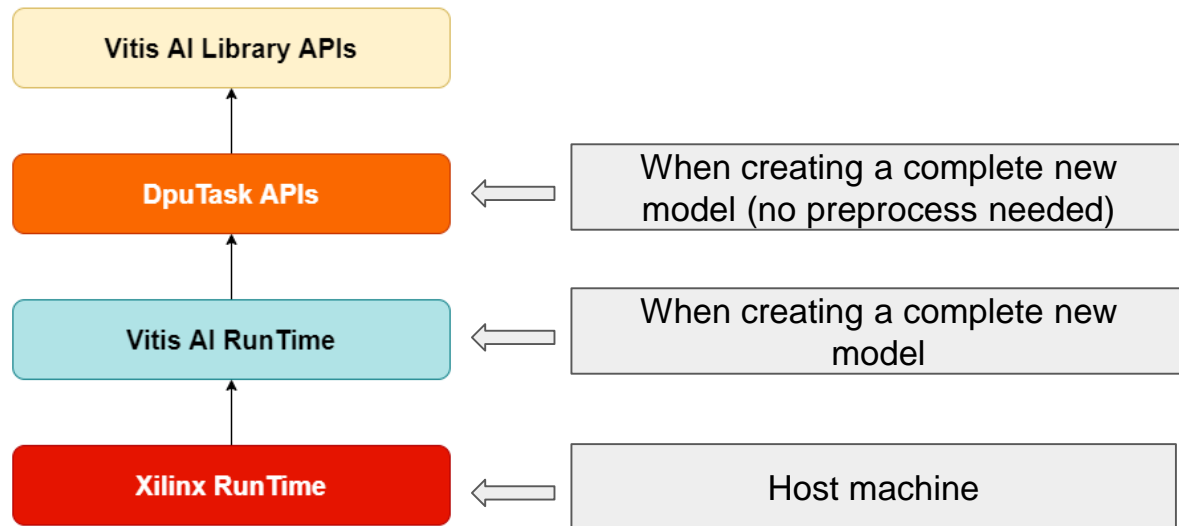
5. Deployment: API Levels



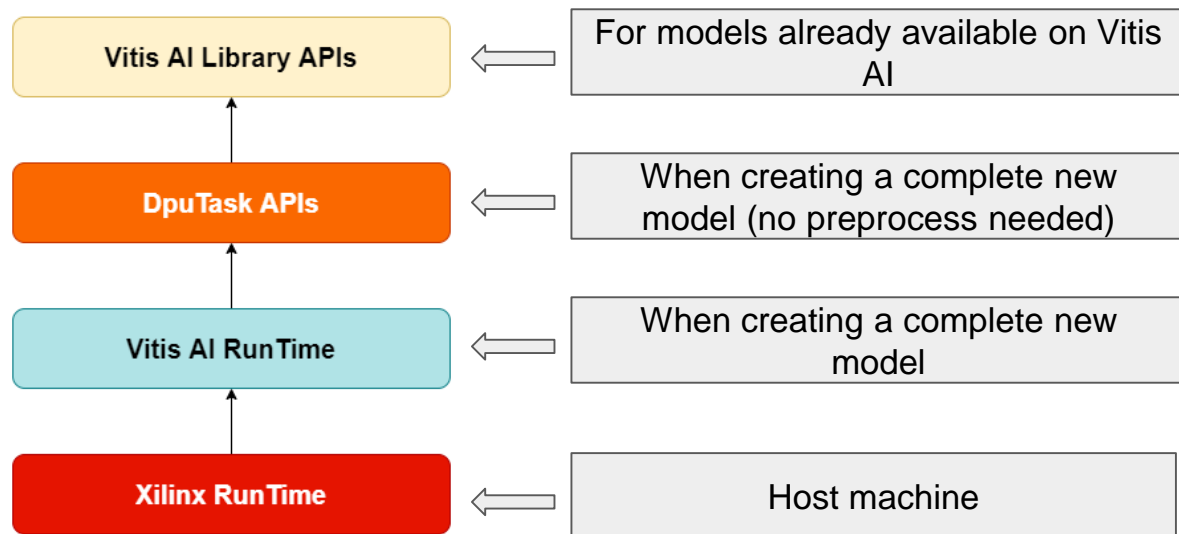
5. Deployment: API Levels



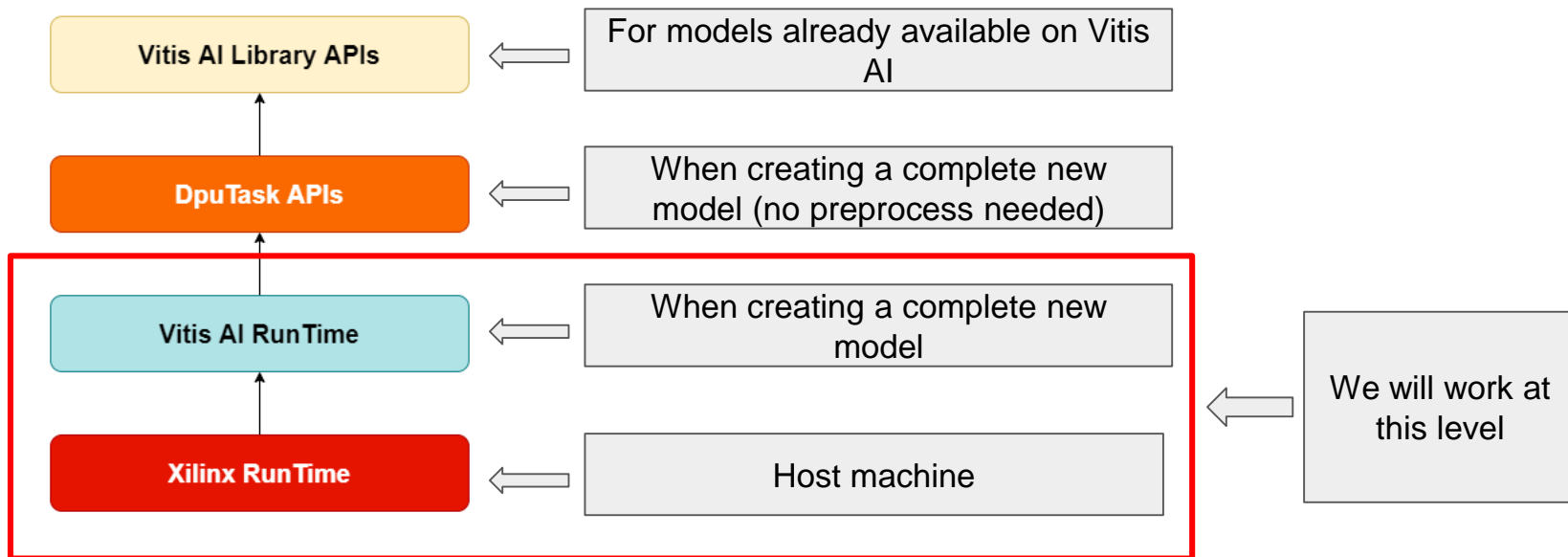
5. Deployment: API Levels



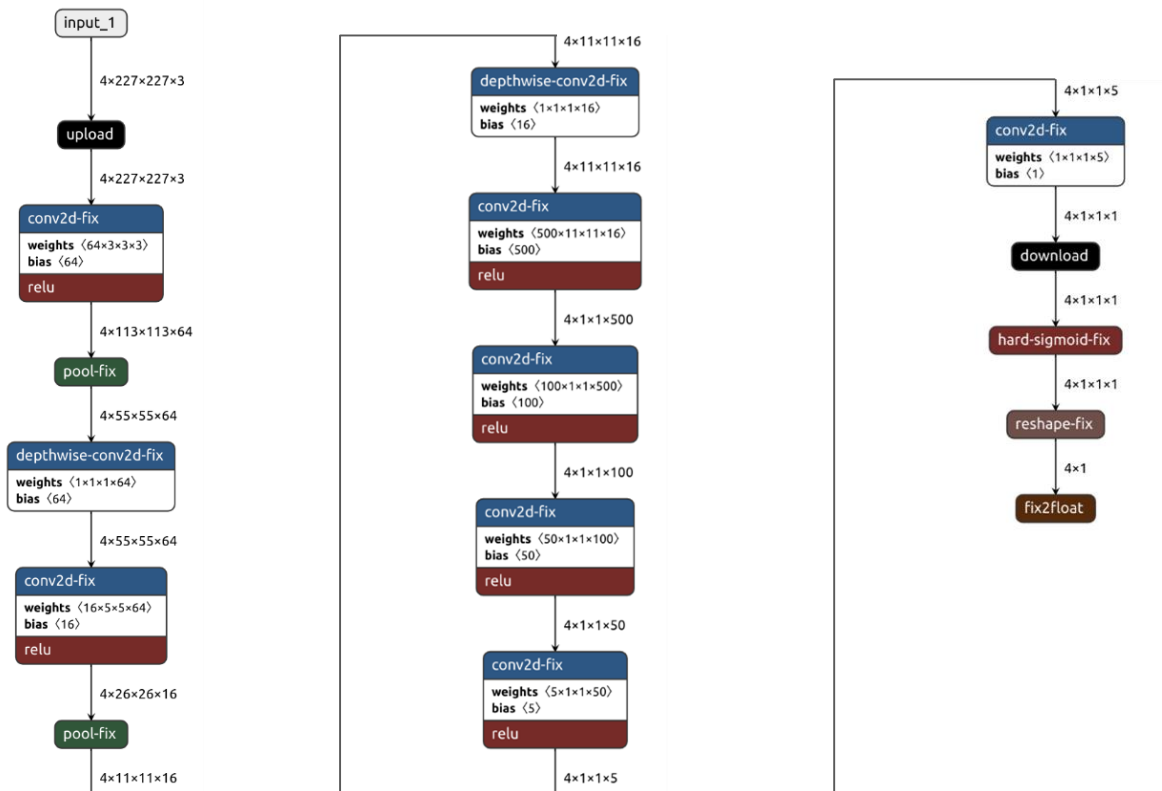
5. Deployment: API Levels



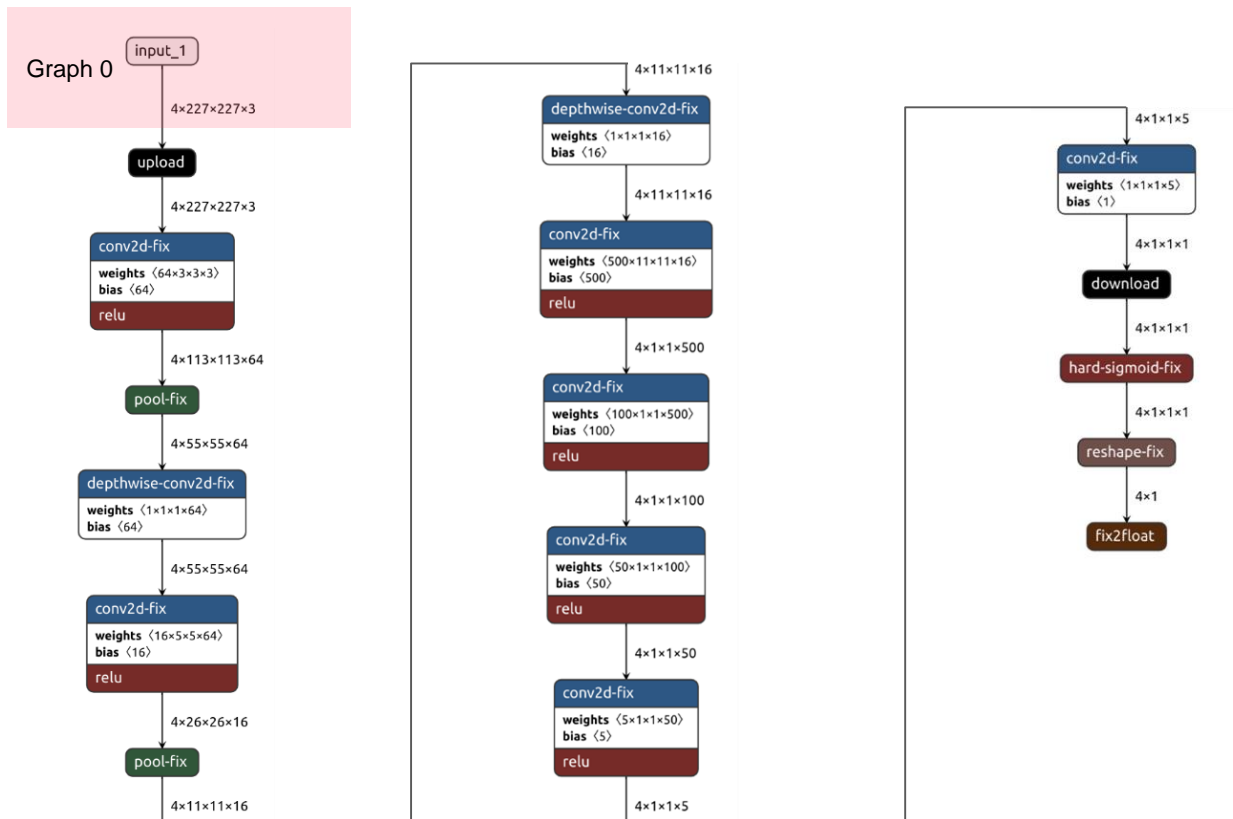
5. Deployment: API Levels



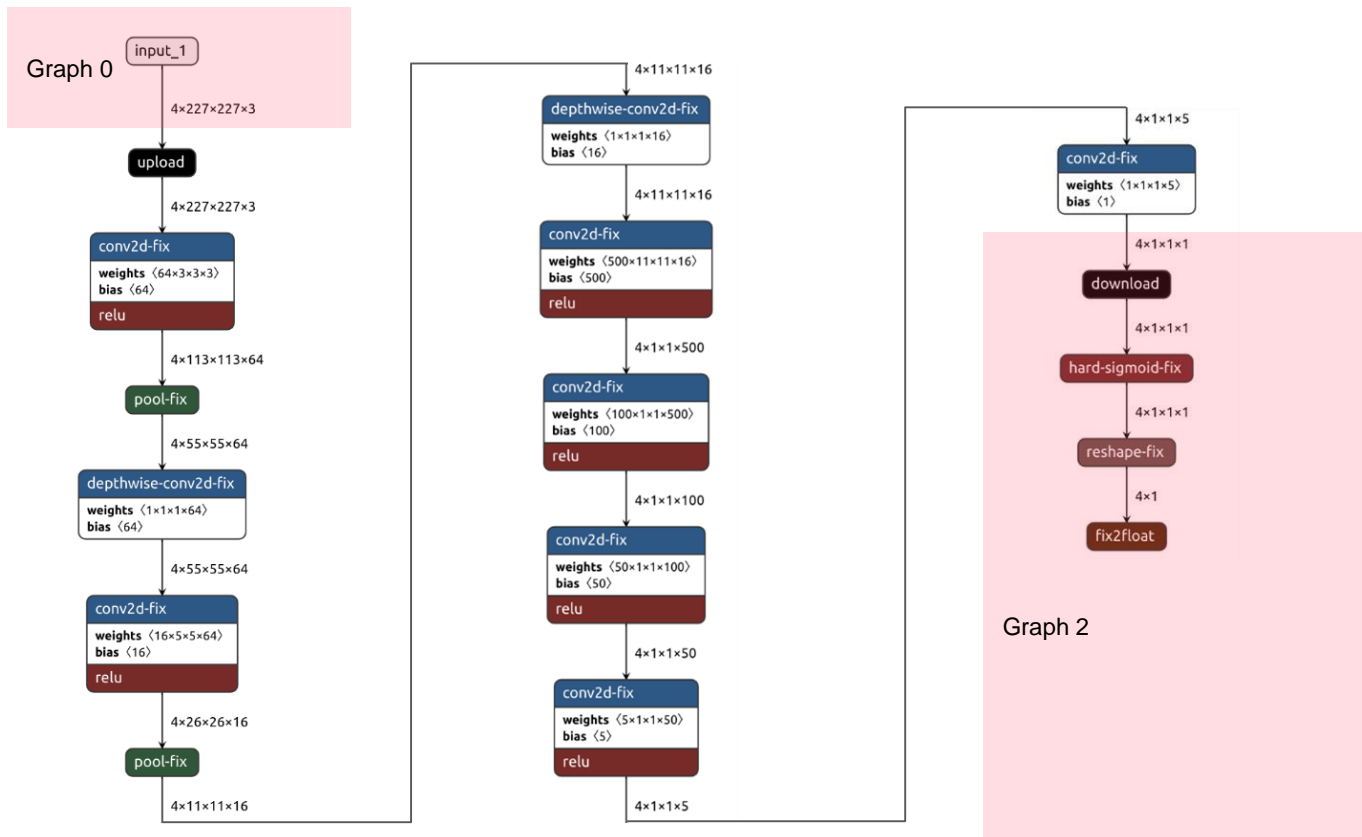
5. Deployment: Graph



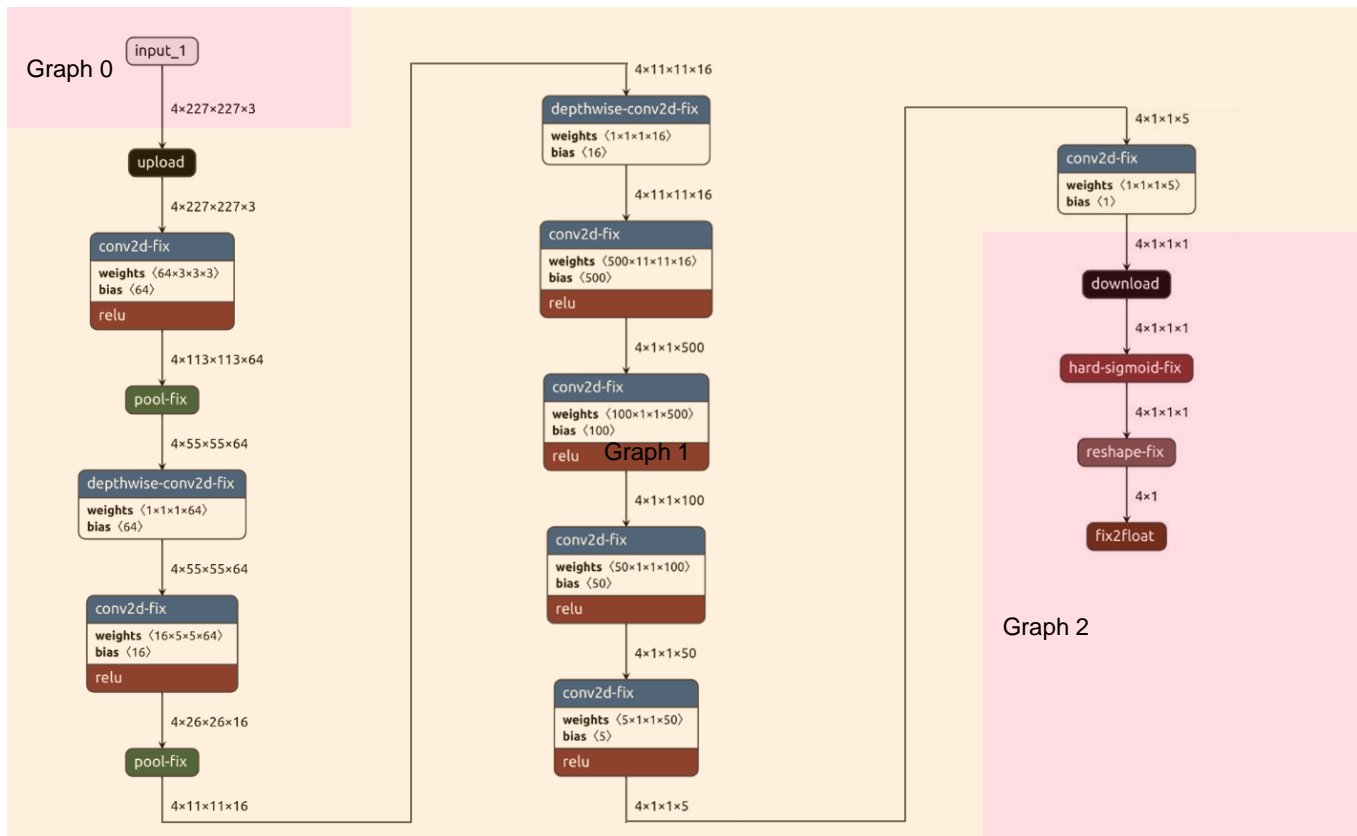
5. Deployment: Sub-Graphs



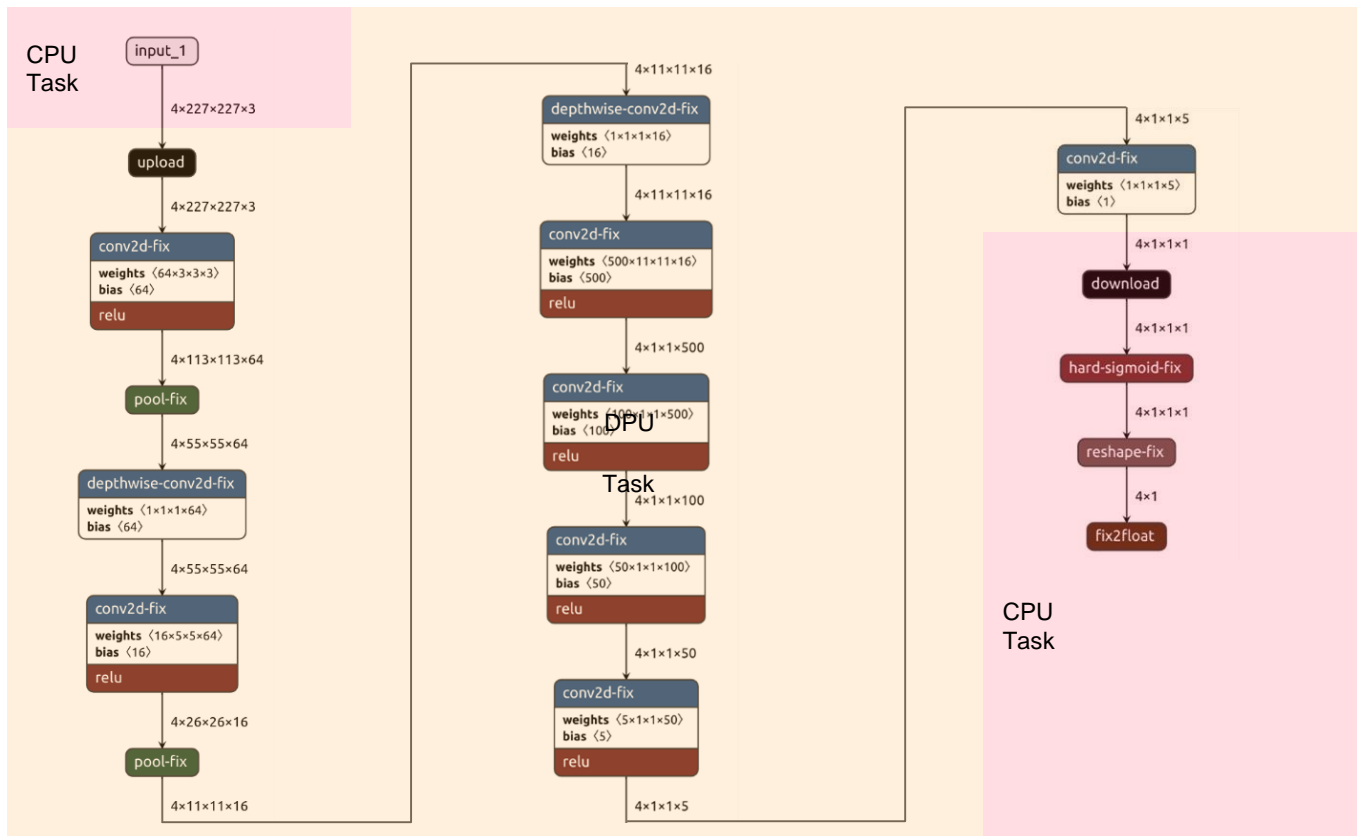
5. Deployment: Sub-Graphs



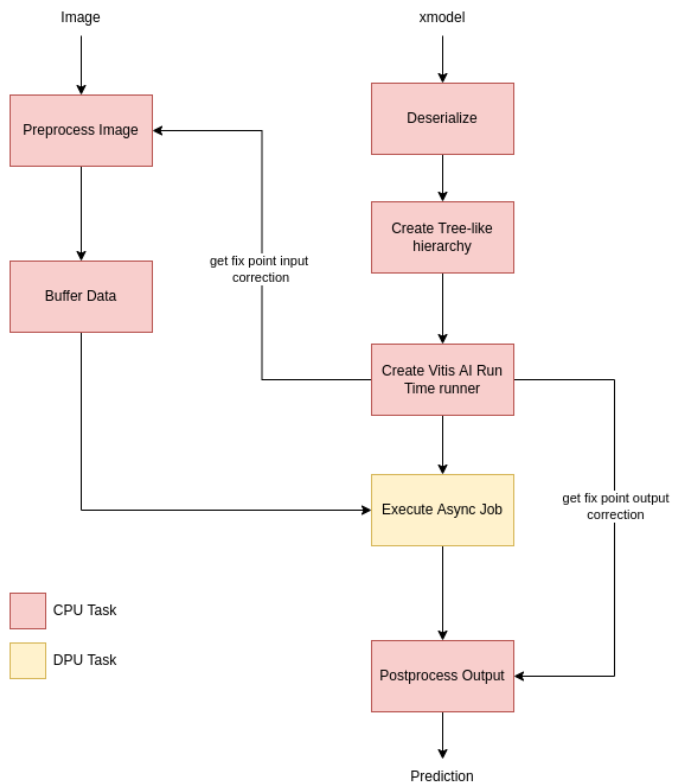
5. Deployment: Sub-Graphs



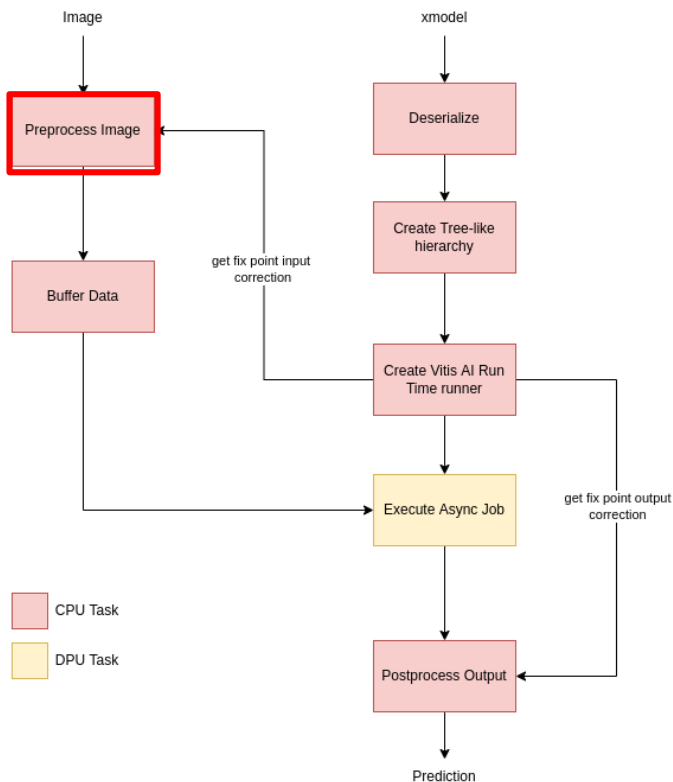
5. Deployment: Sub-Graphs



5. Deployment: Application



5. Deployment: Application



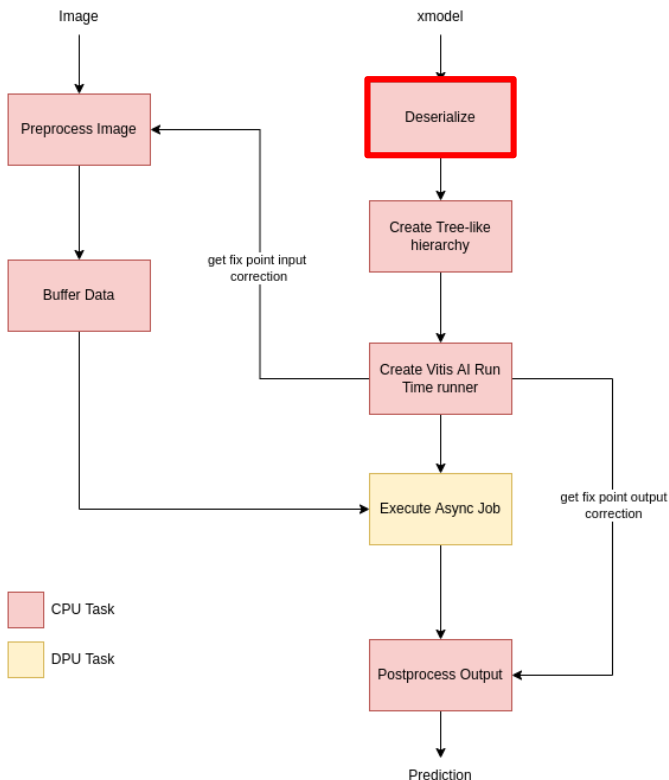
```

1 from PIL import Image
2 import var
3 import xir
4 import numpy as np

5
6 def preprocess_fn(image_path):
7     with Image.open(image_path) as img:
8         if img.mode != 'RGB':
9             img = img.convert('RGB')
10            img = img.resize((227, 227), Image.NEAREST)
11
12            data = np.asarray(img, dtype="float32" )
13            data = data/255.0
14            return data
15
16 x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
17 x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
18 x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
19 x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')
20
21 model = '../compilation/compiled_model/deploy.xmodel'
22
23 g = xir.Graph.deserialize(model)
24
25 root_subgraph = g.get_root_subgraph()
26
27 child_subgraph = root_subgraph.toposort_child_subgraph()
28
29 dpu = var.Runner.create_runner(child_subgraph[1], 'run')
30
31 inputTensors = dpu.get_input_tensors()
32 outputTensors = dpu.get_output_tensors()
33 input_ndim = tuple(inputTensors[0].dims)
34 output_ndim = tuple(outputTensors[0].dims)
35
36
37 input_fixpos = inputTensors[0].get_attr("fix_point")
38 output_fixpos = outputTensors[0].get_attr("fix_point")
39
40 outputData = []
41 inputData = []
42
43 inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
44 outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]
45
46 imageRun = inputData[0]
47
48 imageRun[0] = x0*(2**input_fixpos-1)
49 imageRun[1] = x1*(2**input_fixpos-1)
50 imageRun[2] = x2*(2**input_fixpos-1)
51 imageRun[3] = x3*(2**input_fixpos-1)
52
53 job_id = dpu.execute_async(inputData, outputData)
54 dpu.wait(job_id)
55
56 result = outputData[0]*(2**(output_fixpos)-1)
57 print(result.astype('uint8'))
58

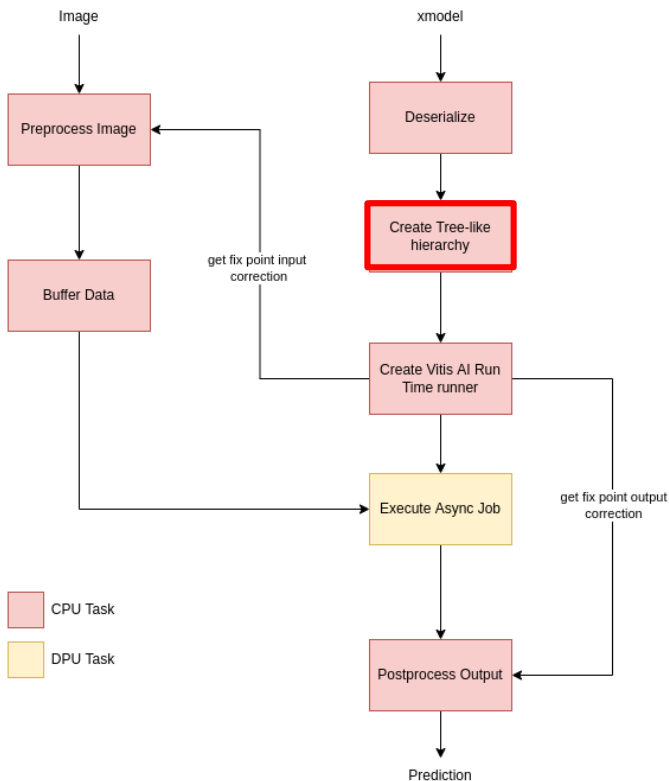
```

5. Deployment: Application



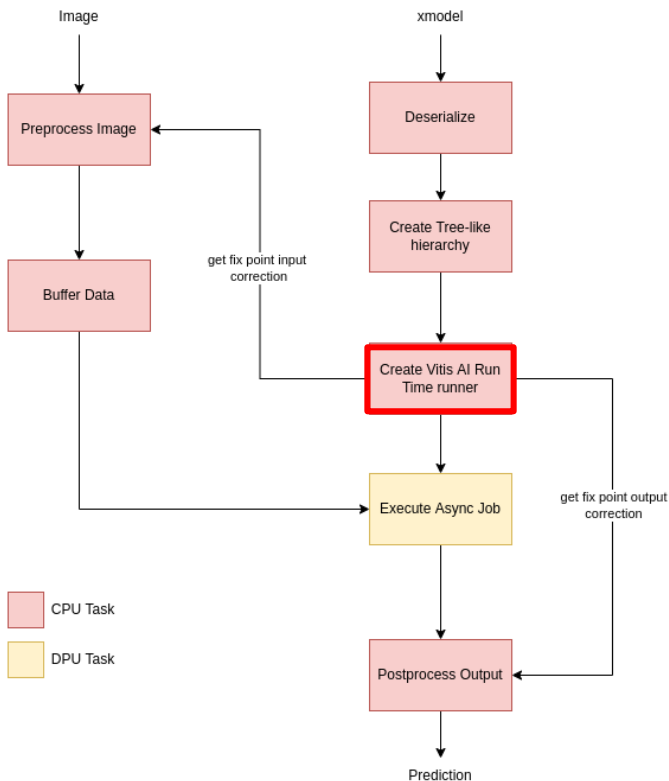
```
1 from PIL import Image
2 import vart
3 import xir
4 import numpy as np
5
6 def preprocess_fn(image_path):
7     with Image.open(image_path) as img:
8         if img.mode != 'RGB':
9             img = img.convert('RGB')
10            img = img.resize((227, 227), Image.NEAREST)
11
12            data = np.asarray(img, dtype="float32" )
13            data = data/255.0
14            return data
15
16 x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
17 x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
18 x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
19 x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')
20
21 model = '../compilation/compiled_model/deploy.xmodel'
22 g = xir.Graph.deserialize(model)
23
24 root_subgraph = g.get_root_subgraph()
25 child_subgraph = root_subgraph.toposort_child_subgraph()
26
27 dpu = vart.Runner.create_runner(child_subgraph[1], 'run')
28
29 inputTensors = dpu.get_input_tensors()
30 outputTensors = dpu.get_output_tensors()
31 input_ndim = tuple(inputTensors[0].dims)
32 output_ndim = tuple(outputTensors[0].dims)
33
34 input_fixpos = inputTensors[0].get_attr("fix_point")
35 output_fixpos = outputTensors[0].get_attr("fix_point")
36
37 outputData = []
38 inputData = []
39
40 inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
41 outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]
42
43 imageRun = inputData[0]
44
45 imageRun[0] = x0*(2**input_fixpos-1)
46 imageRun[1] = x1*(2**input_fixpos-1)
47 imageRun[2] = x2*(2**input_fixpos-1)
48 imageRun[3] = x3*(2**input_fixpos-1)
49
50 job_id = dpu.execute_async(inputData, outputData)
51 dpu.wait(job_id)
52
53 result = outputData[0]*(2**(output_fixpos)-1)
54 print(result.astype('uint8'))
55
```

5. Deployment: Application



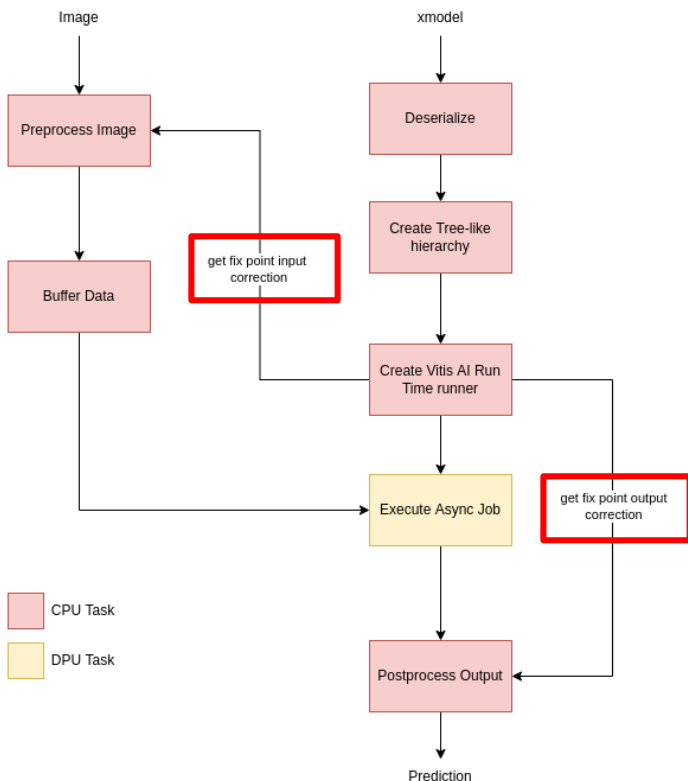
```
1 from PIL import Image
2 import var
3 import xir
4 import numpy as np
5
6 def preprocess_fn(image_path):
7     with Image.open(image_path) as img:
8         if img.mode != 'RGB':
9             img = img.convert('RGB')
10            img = img.resize((227, 227), Image.NEAREST)
11
12            data = np.asarray(img, dtype="float32" )
13            data = data/255.0
14            return data
15
16 x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
17 x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
18 x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
19 x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')
20
21 model = '../compilation/compiled_model/deploy.xmodel'
22
23 g = xir.Graph.deserialize(model)
24
25 root_subgraph = g.get_root_subgraph()
26
27 child_subgraph = root_subgraph.toposort_child_subgraph()
28
29 dpu = var.Runner.create_runner(child_subgraph[1], 'run')
30
31 inputTensors = dpu.get_input_tensors()
32 outputTensors = dpu.get_output_tensors()
33 input_ndim = tuple(inputTensors[0].dims)
34 output_ndim = tuple(outputTensors[0].dims)
35
36
37 input_fixpos = inputTensors[0].get_attr("fix_point")
38 output_fixpos = outputTensors[0].get_attr("fix_point")
39
40 outputData = []
41 inputData = []
42
43 inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
44 outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]
45
46 imageRun = inputData[0]
47
48 imageRun[0] = x0*(2**(input_fixpos-1))
49 imageRun[1] = x1*(2**(input_fixpos-1))
50 imageRun[2] = x2*(2**(input_fixpos-1))
51 imageRun[3] = x3*(2**(input_fixpos-1))
52
53 job_id = dpu.execute_async(inputData, outputData)
54 dpu.wait(job_id)
55
56 result = outputData[0]*(2**(output_fixpos-1))
57 print(result.astype('uint8'))
58
```

5. Deployment: Application



```
1 from PIL import Image
2 import vart
3 import xir
4 import numpy as np
5
6 def preprocess_fn(image_path):
7     with Image.open(image_path) as img:
8         if img.mode != 'RGB':
9             img = img.convert('RGB')
10            img = img.resize((227, 227), Image.NEAREST)
11
12            data = np.asarray(img, dtype="float32" )
13            data = data/255.0
14            return data
15
16 x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
17 x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
18 x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
19 x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')
20
21 model = '../compilation/compiled_model/deploy.xmodel'
22
23 g = xir.Graph.deserialize(model)
24
25 root_subgraph = g.get_root_subgraph()
26
27 child_subgraph = root_subgraph.toposort_child_subgraph()
28
29 dpu = vart.Runner.create_runner(child_subgraph[1], 'run')
30
31 inputTensors = dpu.get_input_tensors()
32 outputTensors = dpu.get_output_tensors()
33 input_ndim = tuple(inputTensors[0].dims)
34 output_ndim = tuple(outputTensors[0].dims)
35
36
37 input_fixpos = inputTensors[0].get_attr("fix_point")
38 output_fixpos = outputTensors[0].get_attr("fix_point")
39
40 outputData = []
41 inputData = []
42
43 inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
44 outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]
45
46 imageRun = inputData[0]
47
48 imageRun[0] = x0*(2**input_fixpos-1)
49 imageRun[1] = x1*(2**input_fixpos-1)
50 imageRun[2] = x2*(2**input_fixpos-1)
51 imageRun[3] = x3*(2**input_fixpos-1)
52
53 job_id = dpu.execute_async(inputData, outputData)
54 dpu.wait(job_id)
55
56 result = outputData[0]*(2**(output_fixpos)-1)
57 print(result.astype('uint8'))
58
```

5. Deployment: Application

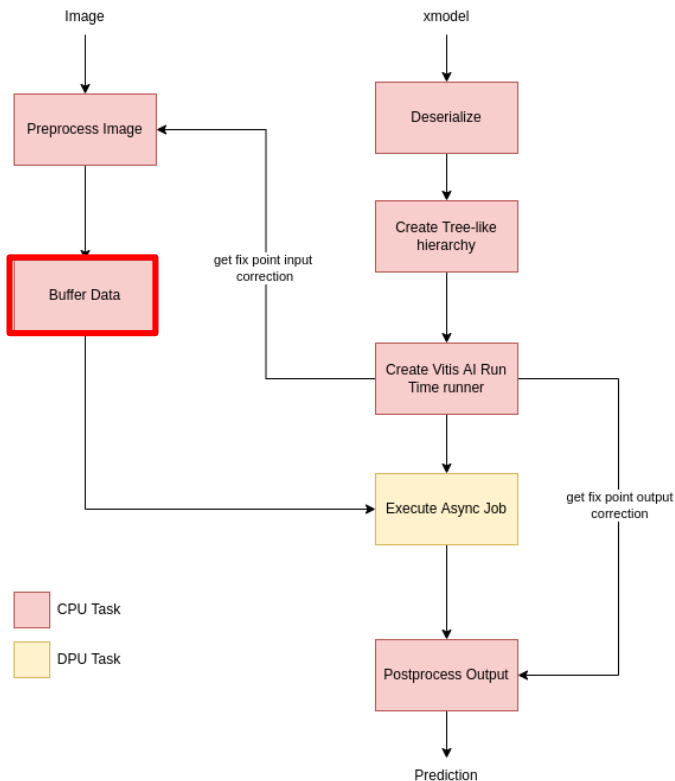


```

1 from PIL import Image
2 import vart
3 import xir
4 import numpy as np
5
6 def preprocess_fn(image_path):
7     with Image.open(image_path) as img:
8         if img.mode != 'RGB':
9             img = img.convert('RGB')
10            img = img.resize((227, 227), Image.NEAREST)
11
12            data = np.asarray(img, dtype="float32" )
13            data = data/255.0
14            return data
15
16 x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
17 x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
18 x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
19 x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')
20
21 model = '../compilation/compiled_model/deploy.xmodel'
22
23 g = xir.Graph.deserialize(model)
24
25 root_subgraph = g.get_root_subgraph()
26
27 child_subgraph = root_subgraph.toposort_child_subgraph()
28
29 dpu = vart.Runner.create_runner(child_subgraph[1], 'run')
30
31 inputTensors = dpu.get_input_tensors()
32 outputTensors = dpu.get_output_tensors()
33 input_ndim = tuple(inputTensors[0].dims)
34 output_ndim = tuple(outputTensors[0].dims)
35
36
37 input_fixpos = inputTensors[0].get_attr("fix_point")
38 output_fixpos = outputTensors[0].get_attr("fix_point")
39
40
41 outputData = []
42 inputData = []
43
44 inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
45 outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]
46
47 imageRun = inputData[0]
48
49 imageRun[0] = x0*(2**input_fixpos-1)
50 imageRun[1] = x1*(2**input_fixpos-1)
51 imageRun[2] = x2*(2**input_fixpos-1)
52 imageRun[3] = x3*(2**input_fixpos-1)
53
54 job_id = dpu.execute_async(inputData, outputData)
55 dpu.wait(job_id)
56
57 result = outputData[0]*(2**(output_fixpos)-1)
58 print(result.astype('uint8'))
59

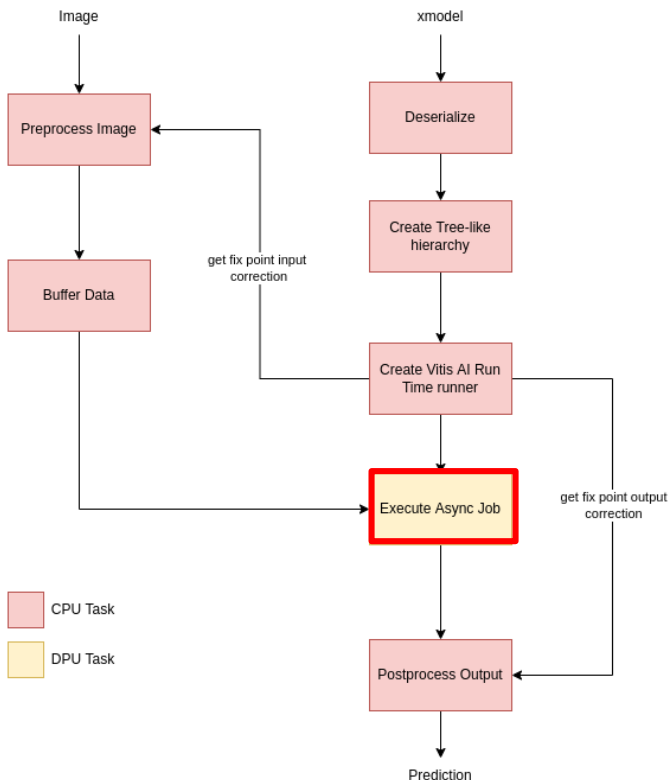
```

5. Deployment: Application



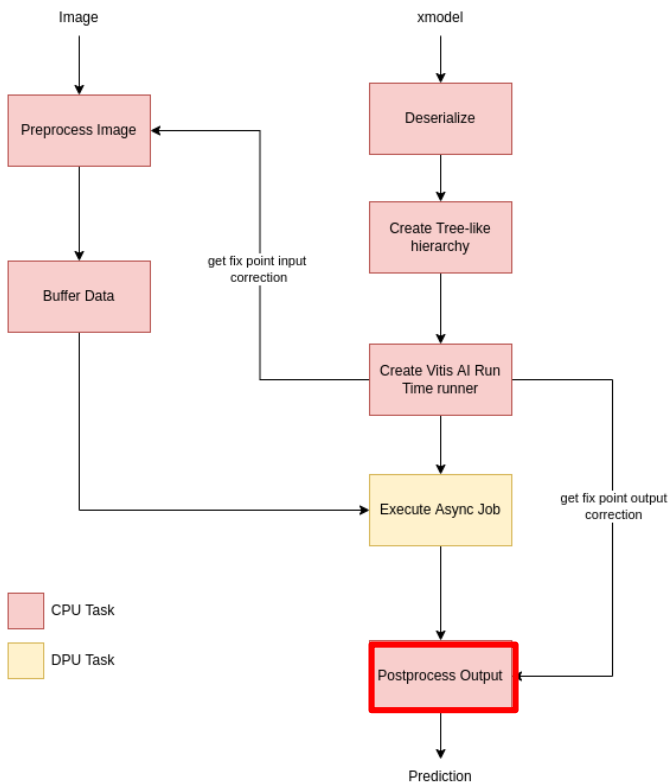
```
1 from PIL import Image
2 import vart
3 import xir
4 import numpy as np
5
6 def preprocess_fn(image_path):
7     with Image.open(image_path) as img:
8         if img.mode != 'RGB':
9             img = img.convert('RGB')
10            img = img.resize((227, 227), Image.NEAREST)
11
12            data = np.asarray(img, dtype="float32" )
13            data = data/255.0
14            return data
15
16 x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
17 x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
18 x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
19 x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')
20
21 model = '../compilation/compiled_model/deploy.xmodel'
22
23 g = xir.Graph.deserialize(model)
24
25 root_subgraph = g.get_root_subgraph()
26
27 child_subgraph = root_subgraph.toposort_child_subgraph()
28
29 dpu = vart.Runner.create_runner(child_subgraph[1], 'run')
30
31 inputTensors = dpu.get_input_tensors()
32 outputTensors = dpu.get_output_tensors()
33 input_ndim = tuple(inputTensors[0].dims)
34 output_ndim = tuple(outputTensors[0].dims)
35
36
37 input_fixpos = inputTensors[0].get_attr("fix_point")
38 output_fixpos = outputTensors[0].get_attr("fix_point")
39
40 outputData = []
41 inputData = []
42
43 inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
44 outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]
45
46 imageRun = inputData[0]
47
48 imageRun[0] = x0*(2**input_fixpos-1)
49 imageRun[1] = x1*(2**input_fixpos-1)
50 imageRun[2] = x2*(2**input_fixpos-1)
51 imageRun[3] = x3*(2**input_fixpos-1)
52
53 job_id = dpu.execute_async(inputData, outputData)
54 dpu.wait(job_id)
55
56 result = outputData[0]*(2**(output_fixpos)-1)
57 print(result.astype('uint8'))
58
```


5. Deployment: Application



```
1 from PIL import Image
2 import vart
3 import xir
4 import numpy as np
5
6 def preprocess_fn(image_path):
7     with Image.open(image_path) as img:
8         if img.mode != 'RGB':
9             img = img.convert('RGB')
10            img = img.resize((227, 227), Image.NEAREST)
11
12            data = np.asarray(img, dtype="float32" )
13            data = data/255.0
14            return data
15
16 x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
17 x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
18 x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
19 x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')
20
21 model = '../compilation/compiled_model/deploy.xmodel'
22
23 g = xir.Graph.deserialize(model)
24
25 root_subgraph = g.get_root_subgraph()
26
27 child_subgraph = root_subgraph.toposort_child_subgraph()
28
29 dpu = vart.Runner.create_runner(child_subgraph[1], 'run')
30
31 inputTensors = dpu.get_input_tensors()
32 outputTensors = dpu.get_output_tensors()
33 input_ndim = tuple(inputTensors[0].dims)
34 output_ndim = tuple(outputTensors[0].dims)
35
36
37 input_fixpos = inputTensors[0].get_attr("fix_point")
38 output_fixpos = outputTensors[0].get_attr("fix_point")
39
40 outputData = []
41 inputData = []
42
43 inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
44 outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]
45
46 imageRun = inputData[0]
47
48 imageRun[0] = x0*(2**input_fixpos-1)
49 imageRun[1] = x1*(2**input_fixpos-1)
50 imageRun[2] = x2*(2**input_fixpos-1)
51 imageRun[3] = x3*(2**input_fixpos-1)
52
53 job_id = dpu.execute_async(inputData, outputData)
54 dpu.wait(job_id)
55
56 result = outputData[0]*(2**(output_fixpos)-1)
57 print(result.astype('uint8'))
58
```

5. Deployment: Application



```
1 from PIL import Image
2 import vart
3 import xir
4 import numpy as np
5
6 def preprocess_fn(image_path):
7     with Image.open(image_path) as img:
8         if img.mode != 'RGB':
9             img = img.convert('RGB')
10            img = img.resize((227, 227), Image.NEAREST)
11
12            data = np.asarray(img, dtype="float32" )
13            data = data/255.0
14            return data
15
16 x0 = preprocess_fn('../dataset/dump/cat/cat.10046.jpg')
17 x1 = preprocess_fn('../dataset/dump/cat/cat.11175.jpg')
18 x2 = preprocess_fn('../dataset/dump/dog/dog.10048.jpg')
19 x3 = preprocess_fn('../dataset/dump/dog/dog.11175.jpg')
20
21 model = '../compilation/compiled_model/deploy.xmodel'
22
23 g = xir.Graph.deserialize(model)
24
25 root_subgraph = g.get_root_subgraph()
26
27 child_subgraph = root_subgraph.toposort_child_subgraph()
28
29 dpu = vart.Runner.create_runner(child_subgraph[1], 'run')
30
31 inputTensors = dpu.get_input_tensors()
32 outputTensors = dpu.get_output_tensors()
33 input_ndim = tuple(inputTensors[0].dims)
34 output_ndim = tuple(outputTensors[0].dims)
35
36
37 input_fixpos = inputTensors[0].get_attr("fix_point")
38 output_fixpos = outputTensors[0].get_attr("fix_point")
39
40 outputData = []
41 inputData = []
42
43 inputData = [np.empty(input_ndim, dtype=np.float32, order="C")]
44 outputData = [np.empty(output_ndim, dtype=np.float32, order="C")]
45
46 imageRun = inputData[0]
47
48 imageRun[0] = x0*(2**input_fixpos-1)
49 imageRun[1] = x1*(2**input_fixpos-1)
50 imageRun[2] = x2*(2**input_fixpos-1)
51 imageRun[3] = x3*(2**input_fixpos-1)
52
53 job_id = dpu.execute_async(inputData, outputData)
54 dpu.wait(job_id)
55
56 result = outputData[0]*(2**(output_fixpos)-1)
57 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!