**Pipeline for Deploying CNN Models on FPGA using Vitis AI with PYNQ Framework and PYNQ-DPU Overlay.**

1. Software and Hardware Requirements (With brief details)

**Software Requirements :** Vitis AI, Vitis AI Docker, PYNQ, PYNQ OS, PYNQ DPU

**Hardware Requirement:** Host system with Ubuntu 20.04 or 22.04 LTS (64-bit), FPGA ZCU104 (or ZCU102, Ultra96 v2, Kria KV260 SOM, Pynq-ZU etc.)

* **Vitis AI** is **Xilinx’s unified AI development platform** for deploying deep learning models on Xilinx hardware including FPGAs, SoCs (like Zynq UltraScale+), and Alveo accelerator cards. It gives a **complete toolchain** to go from a trained deep learning model (TensorFlow, PyTorch, Caffe, etc.) → quantized → compiled → deployed on a Xilinx DPU (Deep Learning Processing Unit).
* **Vitis AI Docker** is basically a **prebuilt container** provided by Xilinx that contains the **entire Vitis AI development environment** so it provides all the tools, dependencies, and compilers without manually installing.
* **PYNQ (Python on Zynq)** is an open-source framework developed initially by Xilinx (now AMD) for simplifying the use of FPGAs, especially Zynq-based Systems on Chips (SoCs), for applications like deep learning and convolutional neural network (CNN) deployment. PYNQ enables FPGA programming using Python and Jupyter Notebooks.
* **PYNQ OS** is a specialized Linux-based operating system that runs on FPGA platforms like Xilinx Zynq and similar adaptive computing boards. It is the foundation of the PYNQ framework, combining a pre-built Linux system with installed Python libraries, drivers, and the Jupyter Notebook server for interactive development. Supports running Python code that interacts with custom hardware overlays loaded into the FPGA fabric, directly from the operating system.
* **PYNQ DPU** is a specialized hardware IP core (Intellectual Property block) developed by Xilinx/AMD and tailored for accelerating neural network computations (including convolution, pooling, activation, etc.), making FPGAs more suitable for real-time AI inference tasks.It refers to the Deep Learning Processing Unit (DPU) overlay for PYNQ, which enables hardware acceleration of deep learning models especially convolutional neural networks (CNNs) on Xilinx FPGA devices running the PYNQ environment.
* The **ZCU104** is an evaluation board from AMD/Xilinx featuring the Zynq UltraScale+ MPSoC (multiprocessor System-on-Chip). It is designed for rapid prototyping and development of advanced embedded systems, especially those requiring high-performance processing, machine learning, and real-time video or vision tasks.

**Key Features**

* Zynq UltraScale+ MPSoC ZU7EV:
* Quad-core ARM Cortex-A53 applications processor.
* Dual-core ARM Cortex-R5 real-time processor.
* Mali-400 MP2 graphics processing unit (GPU).
* H.264/H.265 video codec capable of 4K60 resolution.
* 16nm FinFET+ programmable logic for custom hardware acceleration.
* 4GB DDR4 memory (processing system).
* 64MB DDR4 (programmable logic, via SODIMM slot for expansion).

2. Installation of required software and tools.

2.1 Installation of Vitis AI Docker (Host System)

Version : 2.5

System requirement: Install Ubuntu 20.04 or 22.04 LTS (64-bit)

Dockser is a platform for developing, shipping, and running applications in containers. A container is like a lightweight, portable virtual environment that packages everything your application needs code, libraries, dependencies, runtime so it runs the same way on any system.

* Install first the docker using the link : <https://docs.docker.com/engine/install/ubuntu/>
* Vitis AI docker 2.5 available in : <https://hub.docker.com/r/xilinx/vitis-ai/tags>
  + Open Ubuntu Terminal, Check available docker images: >>docker images
  + Pull the docker Vitis AI 2.5.0 >> docker pull xilinx/vitis-ai:2.5.0
  + It will install vitis ai docker 2.5.0, it support both tensorflow1 and 2
  + After installation go to the specified directory of vitis ai via terminal and run the docker >> /home/user/Vitis-AI/docker\_run.sh xilinx/vitis-ai:2.5

2.2 Install TensorFlow on GPU

Install **TensorFlow GPU** and actually use GPU acceleration, your system needs to have a **compatible graphics card**.

(**NVIDIA GPU** (TensorFlow GPU only supports CUDA-capable NVIDIA GPUs AMD GPUs aren’t supported natively). **CUDA Toolkit** and **cuDNN** installed (TensorFlow GPU version must match these versions exactly)

* + - Download and Install Anaconda: <https://www.anaconda.com/download>
    - Install NVIDIA Cuda and TensorFlow GPU (Open anaconda prompt)
    - Install NVIDIA cuda >>> conda install cuda -c nvidia/label/cuda-11.6.0
    - Check version>>> nvcc -V
    - Check Nvidia GPU details>>> nvidia-smi
    - Create a new environment with specific python version>>> conda create --name <env\_name> python==3.9
    - Activate the environment>>> conda activate <env\_name>
    - Install TensorFlow specific version>>> conda install tensorflow-gpu==1.15 or

2.4.0

* + - Install Jupyter notebook>>> pip install jupyter
    - Install required library like OpenCV, Matplotlib, Glob, Tqdm, Scikit-learn
    - Activate the environment>>> conda activate tensorflow-gpu
    - Open the Jupyter Notebook>>> jupyter notebook

2.3 Load PYNQ Operating System and Install PYNQ DPU (on ZCU104)

Requirement : 16 GB(minimum) micro SD card

PYNQ DPU:2.5.0 , PYNQ OS:3.0

* Download the PYNQ Linux image tailored for ZCU104 from the official PYNQ site: <https://www.pynq.io/boards.html>
* Flash the image onto a microSD card using Balena Etcher(<https://etcher.balena.io/> ) or similar tool
* Insert the SD card and boot up the ZCU104 board (follow these instructions: <https://pynq.readthedocs.io/en/v2.3/getting_started/zcu104_setup.html> )
* Connect to the board with USB cable via SSH or serial console or terminal emulator like Tera Term (<https://teratermproject.github.io/index-en.html> )
* Open Tera Term : Select the serial port>setup>serial port>set speed:115200
* Activate the internet with below code: (Connect the board to network via ethernet cable)
  + Disable “systemd-resolved” service:

>>>sudo systemctl disable systemd-resolved.service

* + Stop the service:

>>>sudo systemctl stop systemd-resolved.service

* + Remove the link to “/run/systemd/resolve/stub-resolv.conf” in “/etc/resolv.conf”

>>>sudo rm /etc/resolv.conf

* + Add a manually created “resolv.conf” in “/etc/”

>>>sudo vim /etc/resolv.conf

* + Add your prefered DNS server there

>>nameserver 8.8.8.8

* Install PYNQ DPU(2.5.0) >>> pip3 install pynq-dpu --no-build-isolation

3. Training and Deployment with the Dataset for inference:

3.1 Training of model of save the model file

* Read the dataset and split them to train, test and valid with a ratio of 70:20:10.
* Perform the data augmentation (Can use offline augmentation tool like Albumentations python library or can use ImageDatagenerator for fly on the go online augmentation. Augmentation will be performed on Train set only.
* Load the pretrained model/custom model with custom classification head (can execute the file train.ipynb) : for ease of demonstration used ImageDatagenerator for augumentation
* The trained model(save best weights) has been saved as model.hdf5 format (floating point 32bit to integer 8bit)

3.2 Conversion of floating point .hdf5 file to quantized and compiled model(.hdf5 to .xmodel) (Summarized from <https://github.com/Xilinx/Vitis-AI-Tutorials/tree/1.4/Introduction/03-Basic/Module_4> )

3.2.1 Converting the Keras HDF5 Checkpoint to a TensorFlow Frozen Graph

The TensorFlow checkpoint is converted into a frozen graph (.pb) where variables become constants and training nodes (optimizer, loss) are removed. The resulting frozen\_graph.pb is stored in ./files/build/freeze/.

* In the host system run the docker >>> /home/user/Vitis-AI/docker\_run.sh xilinx/vitis-ai:2.5
* Activate the TensorFlow environment>>> conda activate tensorflow
* Run the script: keras2tf.py>>>python keras2tf.sh
* Output: it will generate the frozen graph

### 3.2.1 Quantization and Compilation of the model:

The DPU runs only in INT8, so the floating-point frozen graph must be quantized to 8-bit before deployment. Calibration uses a small dataset processed exactly like training data. Images are generated with tf\_gen\_images.py and stored in ./files/build/quantize/images with a list file (later deleted). The function calib\_input in image\_input\_fn.py loads images with OpenCV, converts BGR→RGB, and normalizes pixels to [0,1]. After quantization, two models are produced in ./files/build/quantize: deploy\_model.pb (for deployment) and quantize\_eval\_model.pb (for evaluation).

The Vitis AI compiler converts the quantized model into optimized micro-instructions and outputs an .xmodel in ./build/compile/.

* Run the script: qt\_comp.sh >>>source ./qt\_comp.sh
* Output: model.xmodel

4. Perform the inference on ZCU104 with the .xmodel file

* Boot the zcu104 from the SD card image and perform the connection (using: <https://pynq.readthedocs.io/en/v2.3/getting_started/zcu104_setup.html> )
* Connect the board to a PC using ethernet cable and the PC IP can set to 192.168.2.0 via Tera Term to access the board
* Login to the board by browsing to <http://192.168.2.99> on PC, required password: xilinx
* It open jupyter notebook, upload the .xmodel file and dataset to a directory.
* Upload three more files dpu.bit, dpu.xclbin, and dpu.hwh (specific to the board these can be downloaded to PC from:

<https://github.com/Xilinx/DPU-PYNQ/blob/v2.5.0/pynq_dpu/dpu.bit.link> , <https://github.com/Xilinx/DPU-PYNQ/blob/v2.5.0/pynq_dpu/dpu.hwh.link> , <https://github.com/Xilinx/DPU-PYNQ/blob/v2.5.0/pynq_dpu/dpu.xclbin.link>

dpu.bit **→ Programs the FPGA hardware (the DPU logic).**

dpu.hwh **→ Describes the hardware design to the PYNQ framework.**

dpu.xclbin **→ Used by Vitis AI runtime to run AI models on the DPU**

* + Run the inference.ipynb file that initially uploaded to the board.

The .ipynb file loads the DPU bitstream (dpu.bit) onto the FPGA, which configures the FPGA fabric to instantiate the Deep Learning Processing Unit (DPU). It then loads the compiled deep learning model (.xmodel) onto the DPU. Finally, it runs inference to evaluate accuracy, power, and throughput.