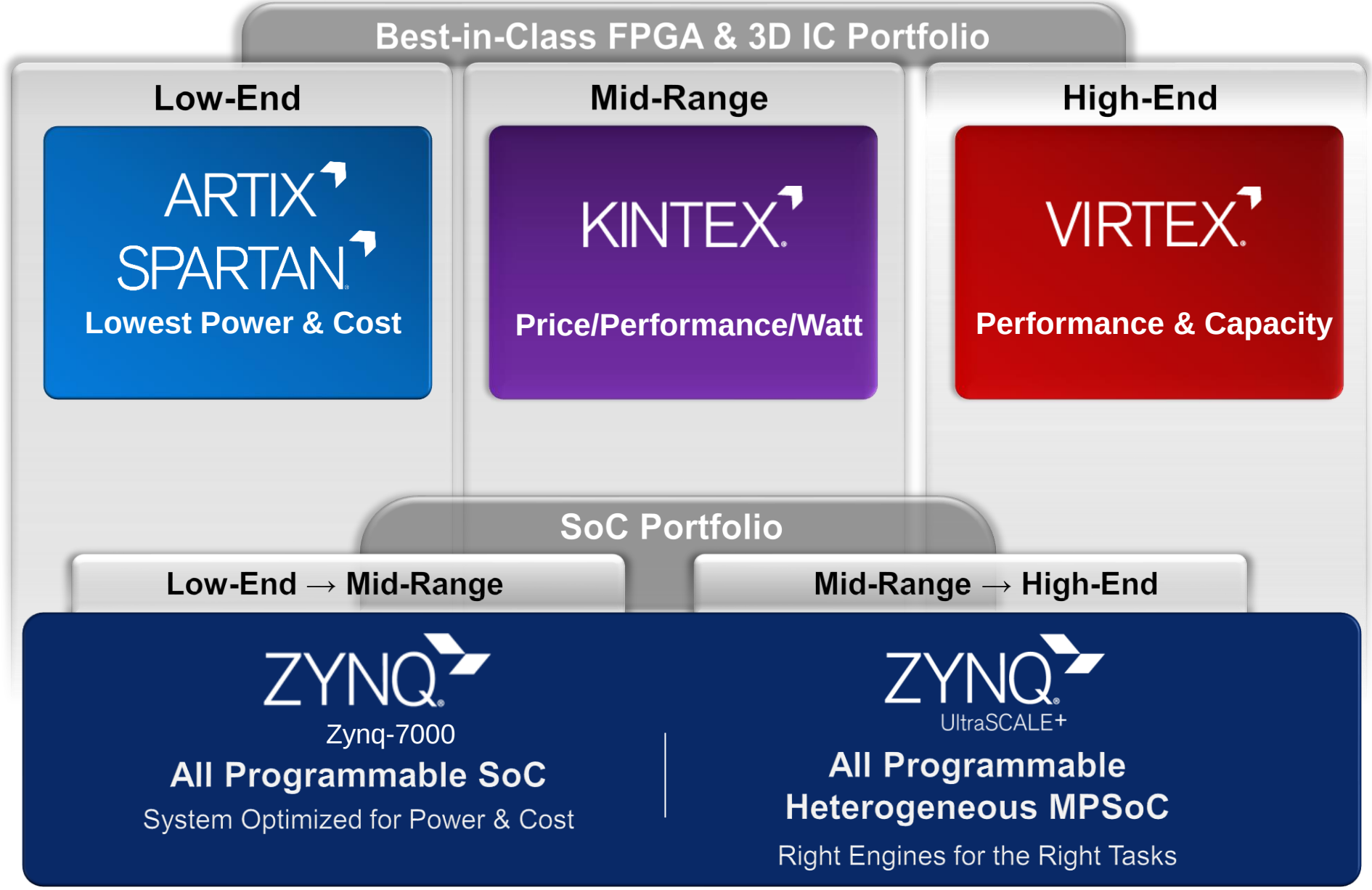
XILINX

Introduction to the Vitis AI Development Environment

Xilinx Product Families: A Broad Portfolio



XILINX

Expanding Our Portfolio with System-on-Modules

Devices	Accelerator Cards	System-On-Modules
		
FPGAs, Adaptive SoCs	x86 Applications	Embedded Applications

A Wide Range of Deployment Methods

XILINX

Design Path for Any Developer

TIME

AI Developer

Customize AI Model

Custom AI application, AI processor configured to your requirements

XILINX VITIS™ | AI

Embedded Developer

Application SW

Customize application SW targeting Arm Cortex-A53 processing subsystem

XILINX VITIS™

Software Developer

Customize Vision Pipeline

Change and accelerate vision pre- and post-processing purely in SW dev environment

XILINX VITIS™

HW Developer

Optional, Not Required

Full Custom RTL Development

Ultimate flexibility for HW acceleration & differentiation

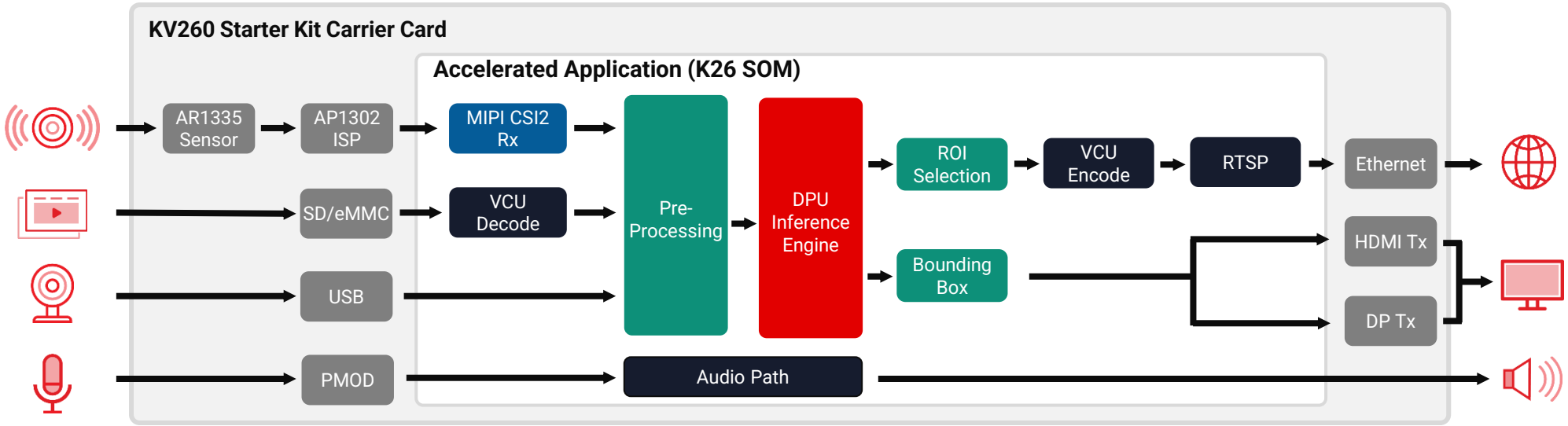
VIVADO®

XILINX®

Let's Use an Example: Smart Camera Accelerated Application

Face Detection + Network & Display

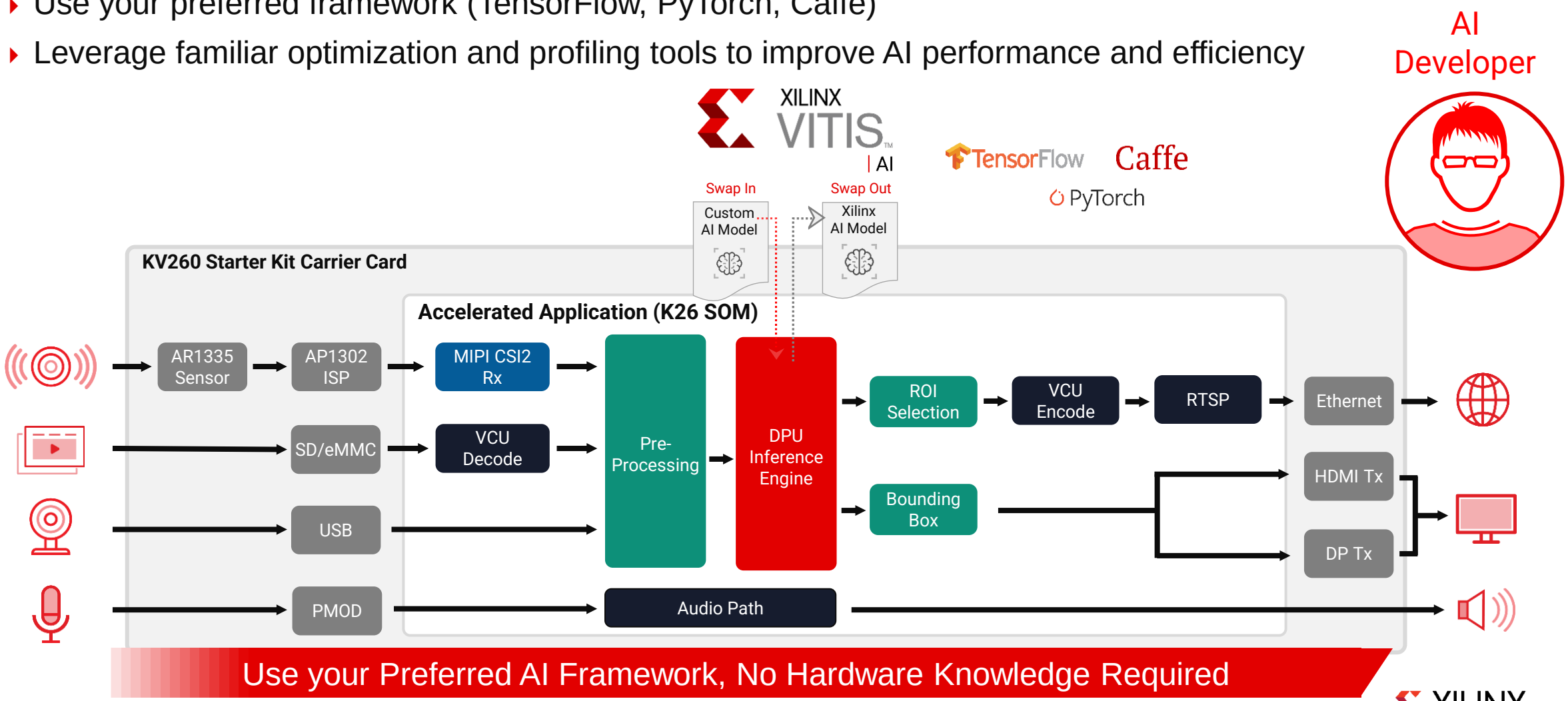
- First, evaluate application by quickly running face detection demo with a local display
- Supports Region of Interest encoding over RTSP for video surveillance & analytics applications
- Capable of ML benchmarking



XILINX

Design with Vitis AI to Customize Your AI Models

- Use Vitis™ AI to swap in your own custom AI model (running on DPU)
- Use your preferred framework (TensorFlow, PyTorch, Caffe)
- Leverage familiar optimization and profiling tools to improve AI performance and efficiency



Use your Preferred AI Framework, No Hardware Knowledge Required

XILINX

Vitis AI

Frameworks

PyTorch | TensorFlow 2.0 | TensorFlow | Caffe

Vitis AI Models

Vitis AI Development Kit

- AI Optimizer
- AI Quantizer
- AI Compiler
- AI Runtime
- AI Library
- AI Profiler
- XRT

DSA

- CNN DPU
- LSTM DPU
- MLP DPU

90+ pretrained, optimized reference models

Open source, performance improvement up to 10-20x

Open-source libraries to achieve optimization easily

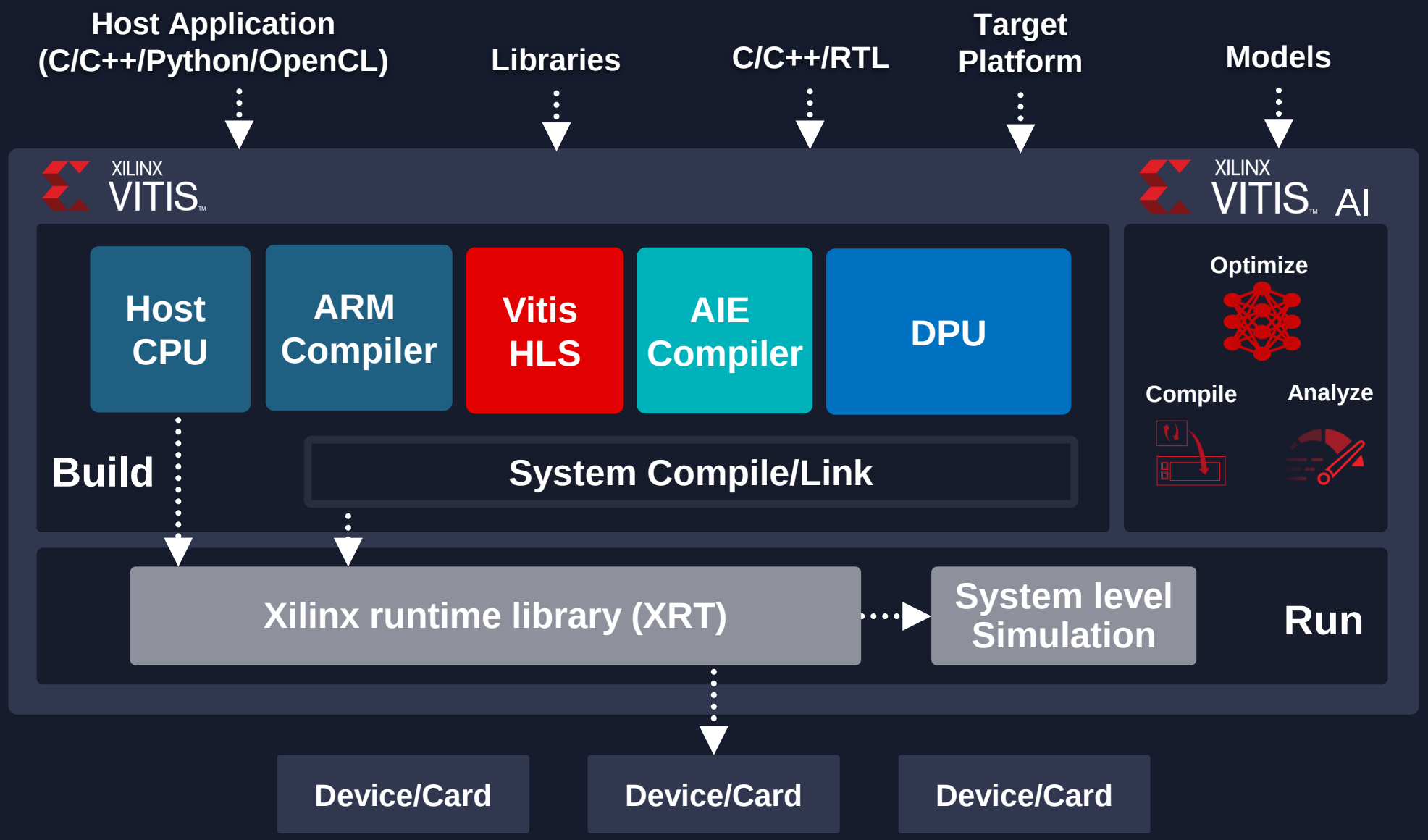
Unified runtime enables edge to cloud

Tensor based ISA for true software programmability

* Coming Soon

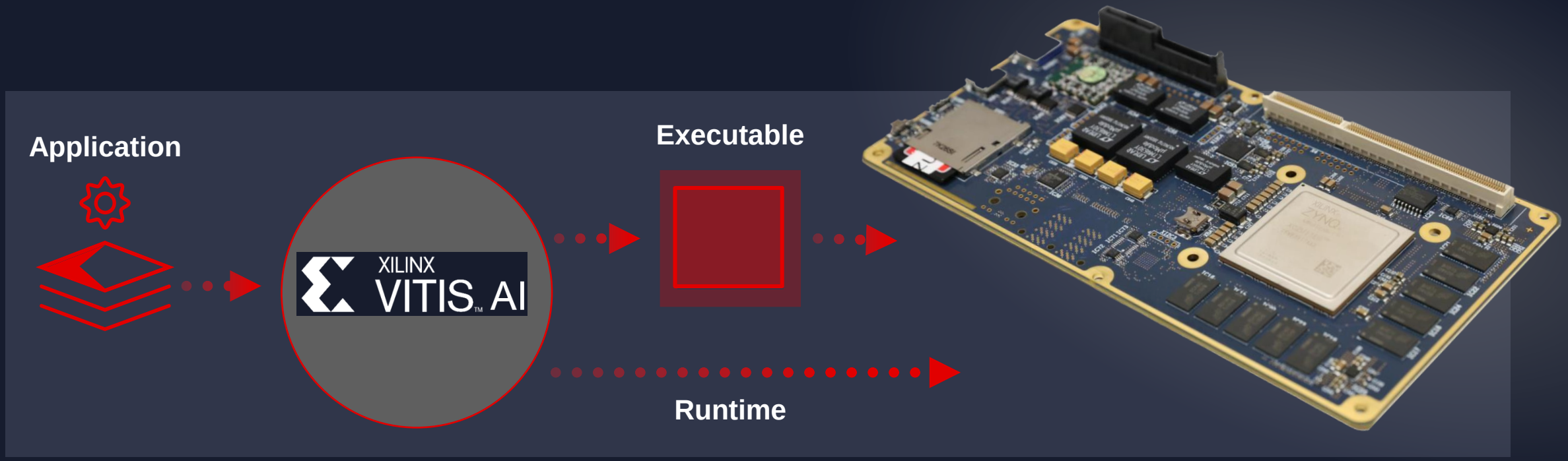
AMD XILINX

Build: Comprehensive Development Tool Suite

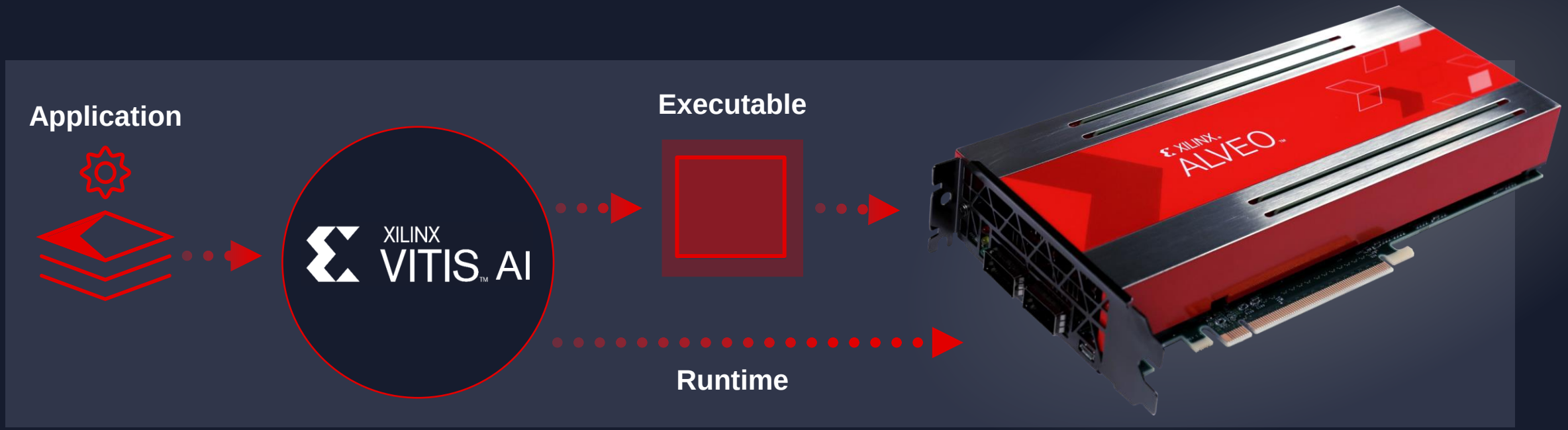


AMD
XILINX

Deploy: Embedded Deployment



Deploy: Single Server Deployment



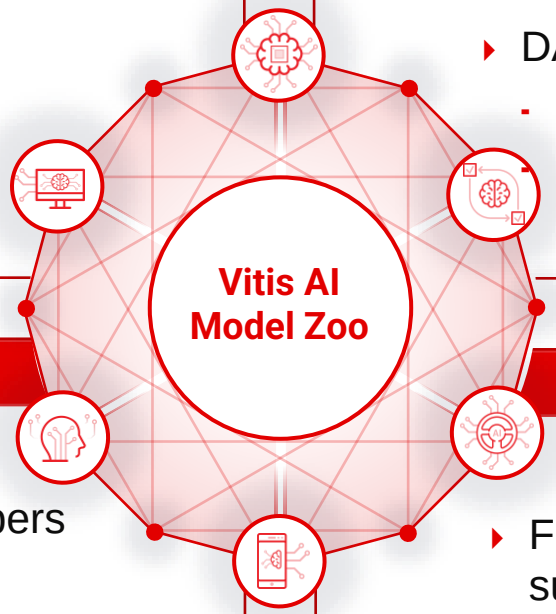
Introducing Vitis AI Model Zoo

Comprehensive model repository

- Support 4 main-stream frameworks
  - Pytorch, TF, TF2 and Caffe
- 134+ ready-to-use CNN models
- More popular applications supported
  - ADAS/Autonomous driving, smart medical, smart city, data center, etc.

Advanced optimization applied

- Pruned model with competitive performance on different Xilinx platforms
- DAS-facing design methods adopted
  - Innovated model structure
  - Multi-task



Easy to use algorithm solutions

- Open and free on Github for all developers
- Provided accuracy evaluation and quantization scripts
- Comprehensive model information included

Deployable from edge to cloud

- Fully tested and deployed on all Vitis AI supported platforms
  - Zyny7000/ZU+/Alveo and Versal
- Retrainable with custom dataset

AMD XILINX

Vitis AI Model Zoo

Vitis AI Model Zoo Repository:
https://github.com/Xilinx/Vitis-AI/tree/2.5/model_zoo

Open and free for all developers

Rich models from TensorFlow and Caffe and PyTorch

Advanced optimization, including **PRUNING** applied

Retrainable with **CUSTOM *dataset***

S.No.	Application	Model	Name	Framework	Backbone	Input Size	OPS per image	Training Set	Val Set	Float (Top1,Top5) /mAP/mIoU	Quantized (Top1,Top5) /mAP/mIoU
1.	Image Classification	resnet50	cf_resnet50_imagenet_224_224_7.7G	caffe	resnet50	224*224	7.7G	ImageNet Train	ImageNet Validation	0.74828 / 0.92135	0.7338 / 0.9130

XILINX.

Getting Started

- Downloading the Vitis AI development kit
  - Available via Docker Hub: https://hub.docker.com/r/xilinx/vitis-ai/tags
  - Consists of the following two packages
    - Tools Docker container xilinx/vitis-ai
    - Vitis AI runtime package for edge
    - https://github.com/Xilinx/Vitis-Tutorials
    - VITIS AI PYTORCH
- Evaluation boards that are supported by Vitis AI development kit v1.1 ~ v2.5
  - Cloud: Xilinx Alveo™ cards U200, U250, and U50
  - Edge: Xilinx MPSoC evaluation boards ZCU102 and ZCU104 and Kria KV260 Kit and Versal AI Core Series VCK190 Kit

1. Install the docker
2. Ensure Linux user is in the group docker
3. Clone the Vitis-AI repository:

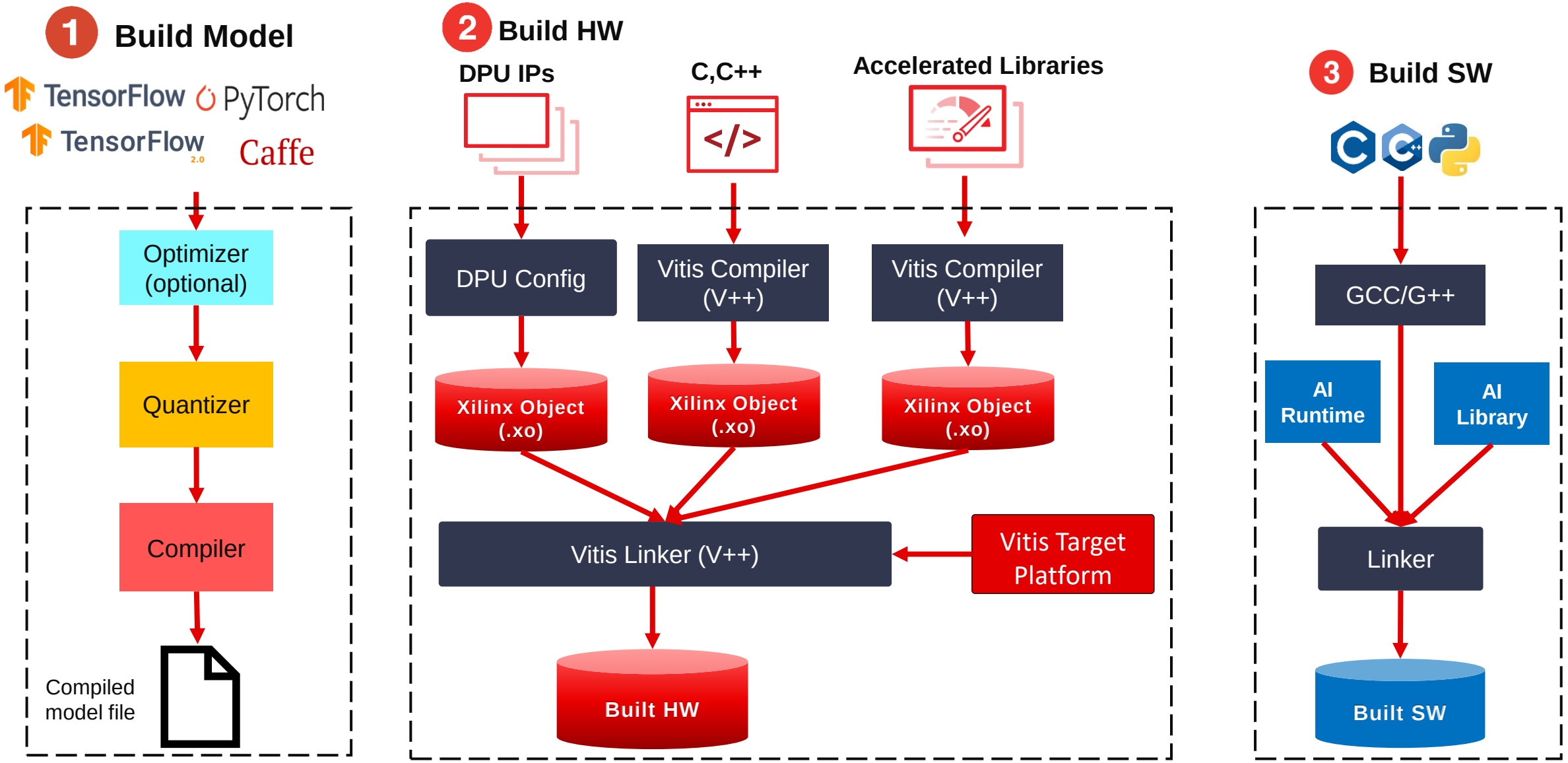
```
git clone
https://github.com/Xilinx/Vitis-AI
cd Vitis-AI
```

4. Run the docker container

```
./docker_run.sh Xilinx/vitis-ai
```

5. Get started with examples

Vitis AI Development Flow



Vitis AI For Edge (KV260) Platform Design Flow

Vitis-Tutorials/Vitis_Platform_Creation/Design_Tutorials/01-Edge-KV260 at 2021.2 · Xilinx/Vitis-Tutorials · GitHub

https://github.com/Xilinx/Vitis-Tutorials/tree/2022.1/Vitis_Platform_Creation/Feature_Tutorials/02_petalinux_customization

Vitis-AI/README.md at master · Xilinx/Vitis-AI · GitHub

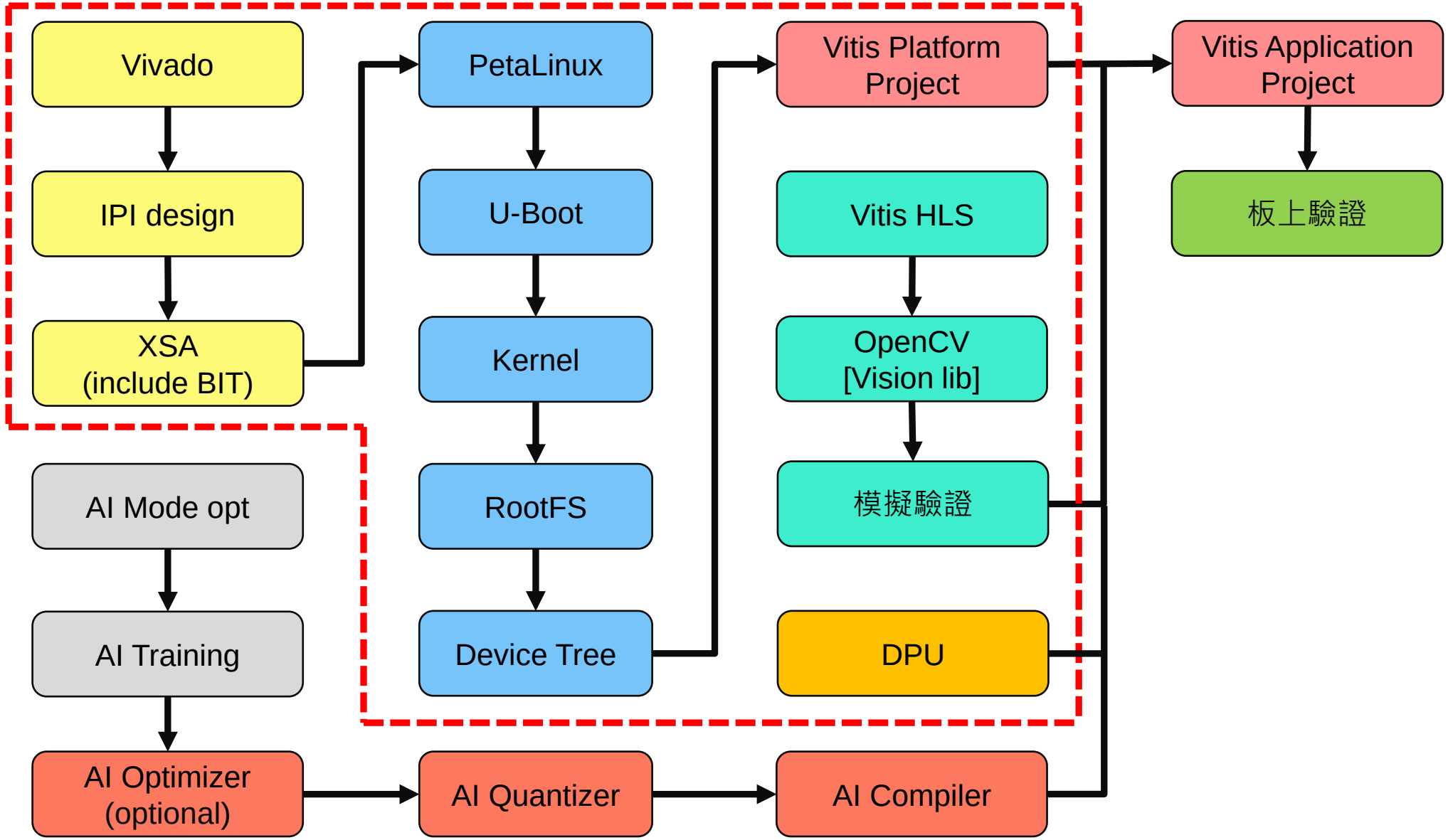
步驟 1 建立 Vivado 硬件設計並生成 XSA

步驟 2 使用 PetaLinux 建立軟件組件

步驟 3 建立 Vitis + DPU Platform

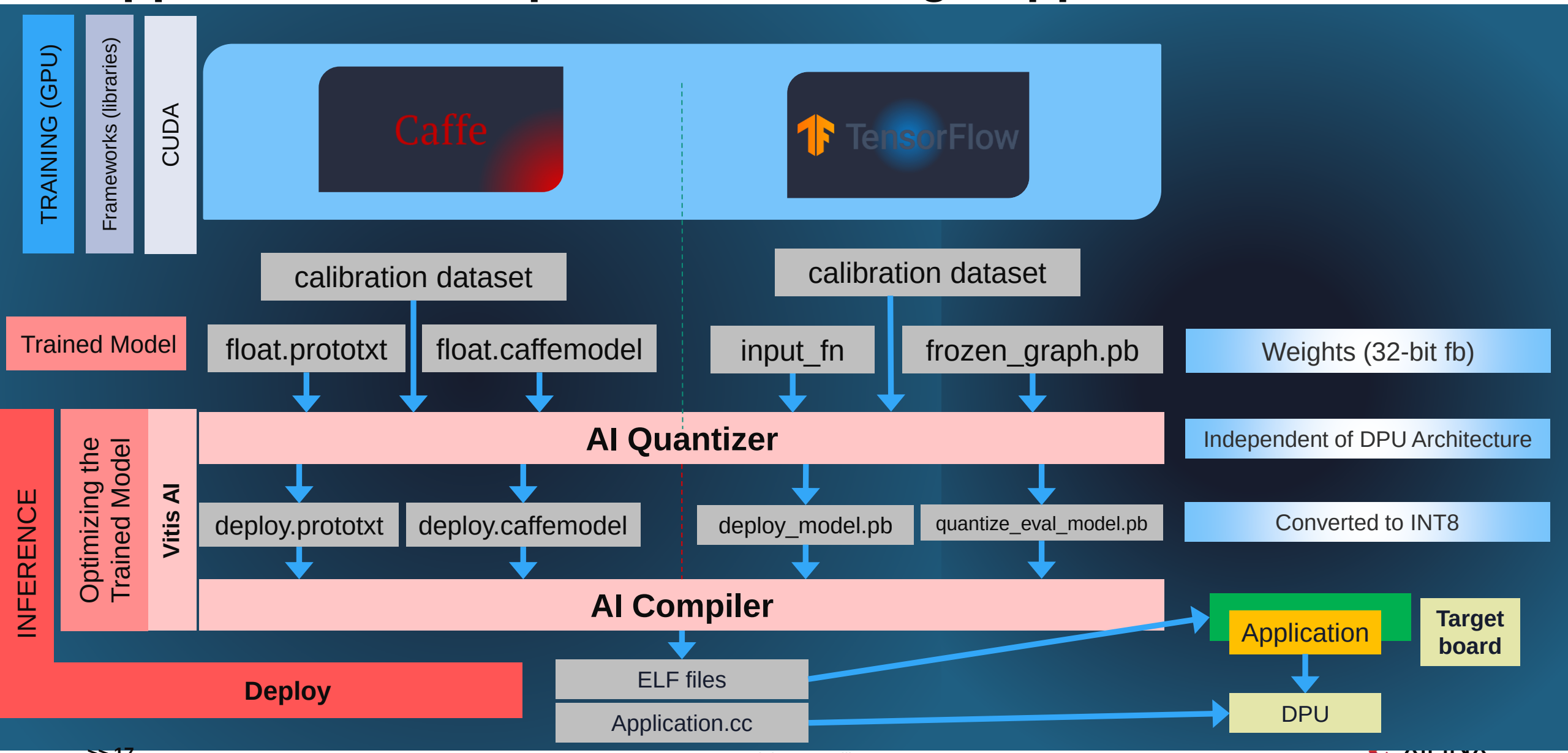
步驟 4 測試平台

Vitis AI For Edge Design Flow



XILINX.

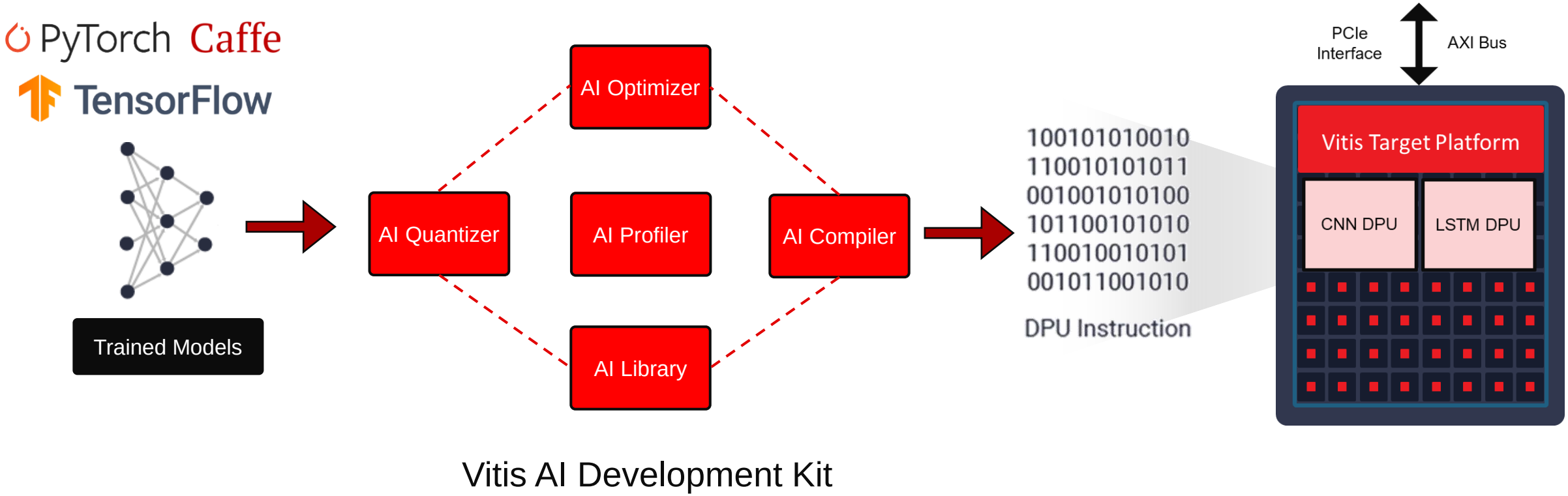
Application Development Flow Using Supported Frameworks



Vitis AI Development Kit – CNN

Optimizer
Quantizer
Compiler

Direct Framework Compilation In Minutes

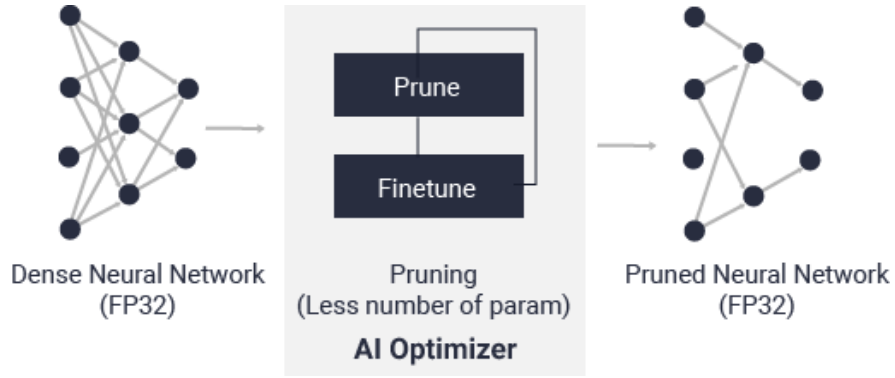


Vitis AI Development Kit

AMD XILINX

Vitis AI Optimizer

- World's leading model compression technology
  - Iterative, coarse-grained pruning
  - Reduce model size 5 – 30x
  - Increase performance 2 – 10x
  - Minimal accuracy loss, <1%
- Support mainstream frameworks
  - Tensorflow 1.x, Tensorflow 2.x, Pytorch, Caffe and DarkNet
- Easy-to-use with simplified APIs
- Commercial license available
  - Optimizer Member Lounge:
    - https://www.xilinx.com/member/ai_optimizer.html
  - Please contact your Xilinx sales representative



Check more optimized models on Github AI Model Zoo

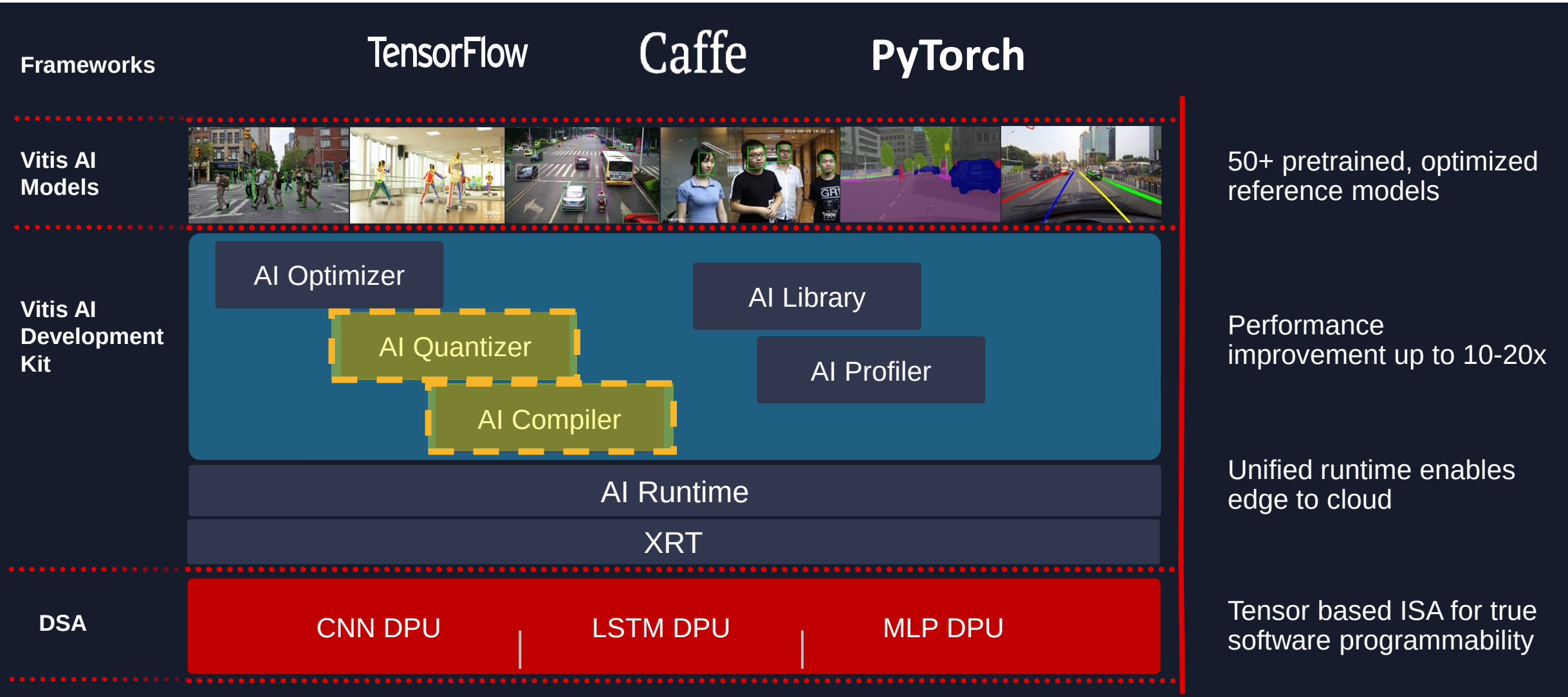
Model	Pruning Ratio	Operation (GOP)	Latency (ms)*	Throughput (FPS)**
RefineDet	-	123	115	18
	80%	25	31	76
	92%	10	16	154
	96%	5	12	228

* Latency is measured with a single thread
** Throughput is measured on ZCU104, with dual B4096 cores integrated

XILINX

AI Quantizer and AI Compiler

Vitis AI Development Environment



XILINX.

Inference Process

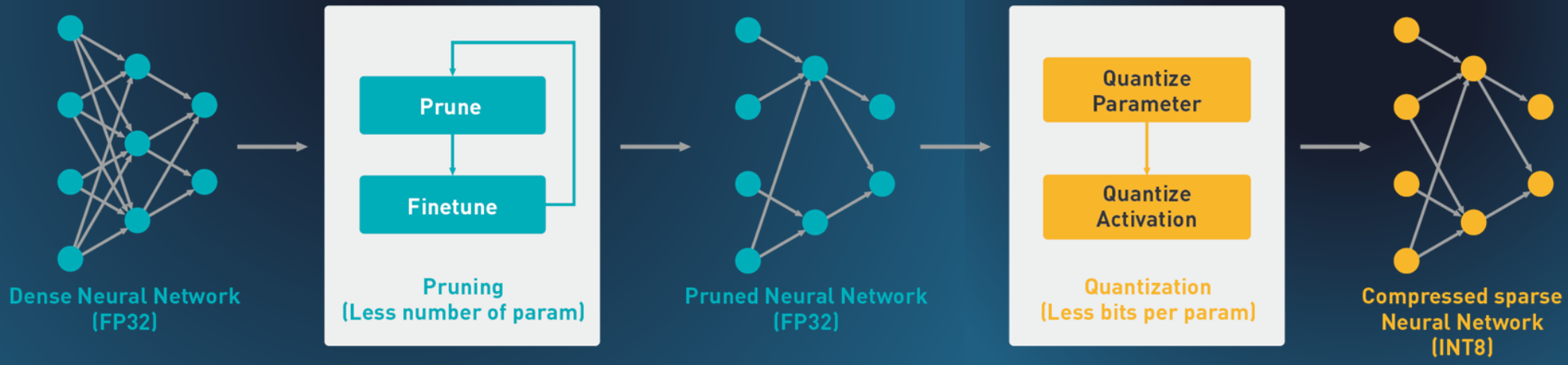
Process of inference:

Computation intensive

High memory bandwidth

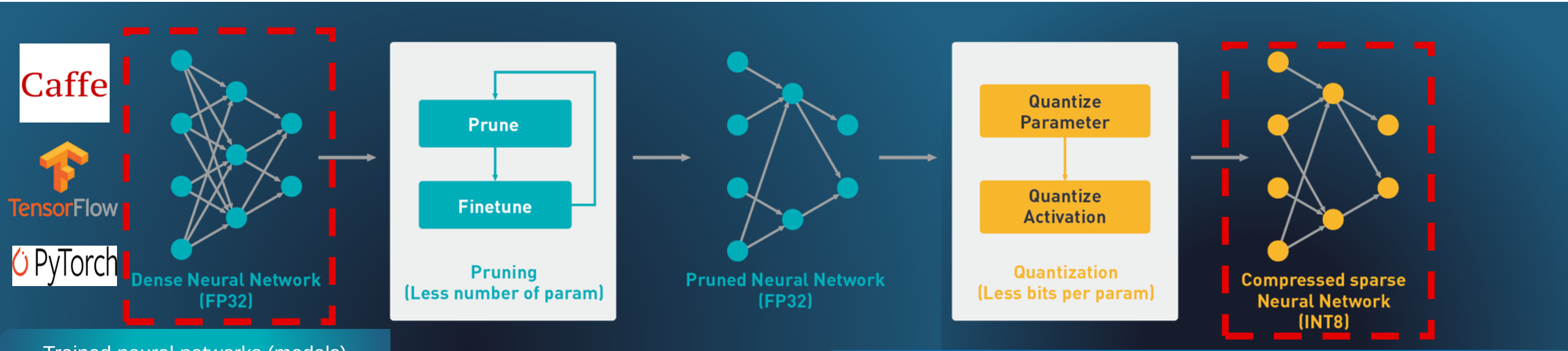
Requires:
Low latency and high throughput

Quantization and pruning techniques:
- Achieve high performance
- High energy efficiency



XILINX.

Quantization



Trained neural networks (models) from frameworks

Converting 32-bit floating-point to 8-bit integer (INT8)

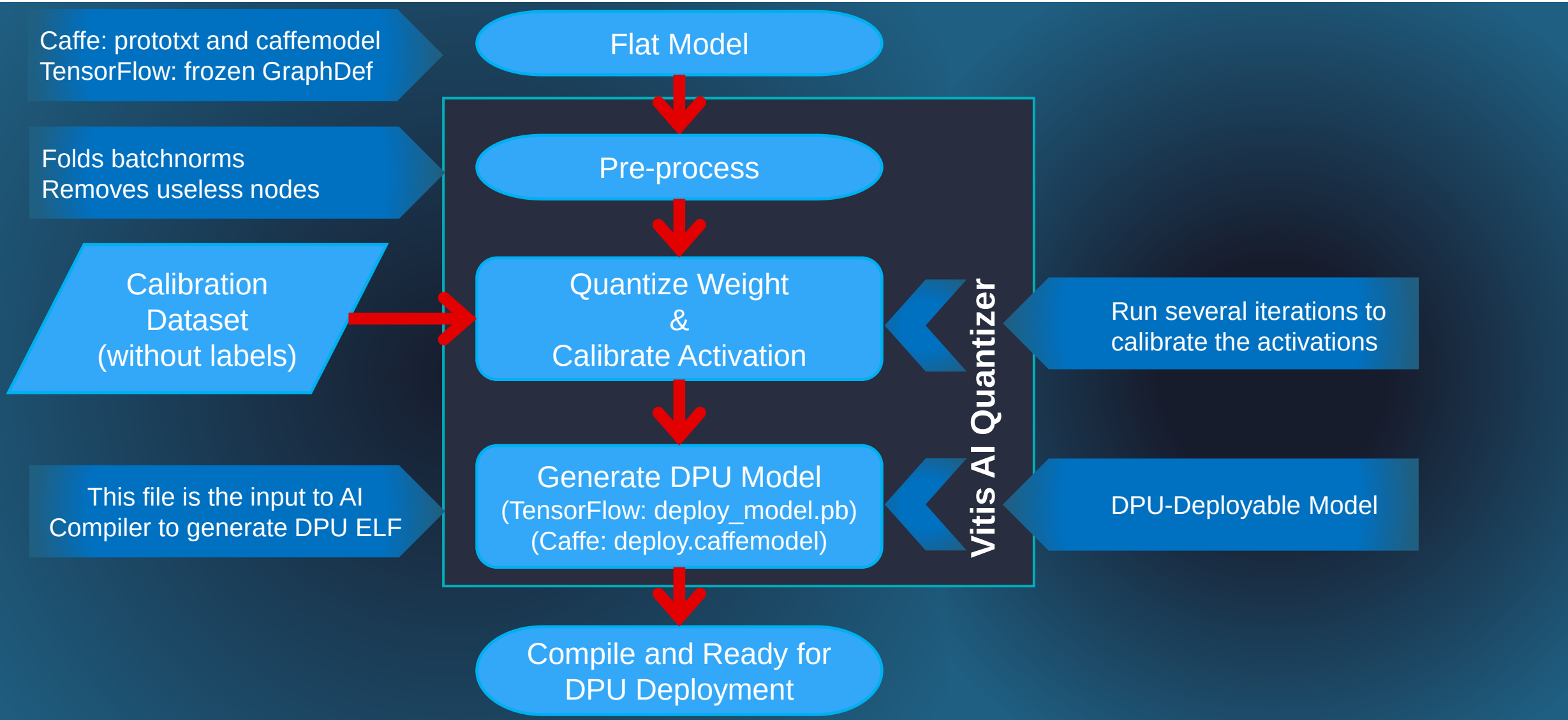
Trained neural networks (model) are in 32-bit floating-point

Vitis AI quantizer supports:

```
TensorFlow (vai_q_tensorflow)
Caffe (vai_q_caffe)
Pytorch (vai_q_pytorch)
```

Vitis AI quantizer can reduce computing complexity without losing prediction accuracy

Vitis AI Quantizer Flow



XILINX.

Quantizer for Caffe: vai_q_caffe

Vitis AI quantizer for Caffe — vai_q_caffe

vai_q_caffe: Required input files

- **float.prototxt** — Caffe floating-point network model prototxt file
- **float.caffemodel** — Pre-trained Caffe floating-point network model
- **Calibration dataset** — A subset of the training set

Run AI qua
to generate a

```
vai_q_caffe quantize -model ${CF_NETWORK_PATH}/float/trainval.prototxt \
                     -weights ${CF_NETWORK_PATH}/float/float.caffemodel \
                     -output_dir ${CF_NETWORK_PATH}/vai_q_output \
                     -calib_iter 2 \
                     -test_iter 10 \
                     -auto_test \
```

Quantizer for Caffe: vai_q_caffe

Run AI quantizer for Caffe to generate a fixed-point model

```
vai_q_caffe quantize -model float.prototxt -weights
float.caffemodel [OPTIONS]
```

vai_q_caffe: Output files

- **deploy.prototxt**
- **deploy.caffemodel**
- **quantize_train_test.prototxt**
- **quantize_train_test.caffemodel**

These files are used as input files to the compiler

These files are used to test the accuracy on the GPU/CPU; can be used as input files to quantize finetuning

Quantizer for Caffe: vai_q_caffe Usage

Run AI quantizer for Caffe to generate a fixed-point model

```
vai_q_caffe quantize -model float.prototxt -weights
float.caffemodel [OPTIONS]
```

Name	Type	Optional	Default	Description
model	String	Required	-	Floating-point prototxt file
weights	String	Required	-	Pre-trained floating-point weights
weights_bit	Int32	Optional	8	Bit width for quantized weight and bias
data_bit	Int32	Optional	8	Bit width for quantized activation
method	Int32	Optional	1	Quantization methods, including 0 for non-overflow and 1 for min-diffs
calib_iter	Int32	Optional	100	Maximum iterations for calibration
auto_test	Bool	Optional	FALSE	Run test after calibration, test dataset required
test_iter	Int32	Optional	50	Maximum iterations for testing
output_dir	String	Optional	quantize_result	Output directory for the quantized results
gpu	String	Optional	0	GPU devide ID and calibration and test
ignore_layers	String	Optional	None	List of layers to ignore during quantization
ignore_layers_file	String	Optional	None	Protobuf file which defines the layers to ignore during quantization, starting with ignore_layers

vai_q_caffe Quantize Finetuning

Run AI quantizer for Caffe for finetuning

```
vai_q_caffe quantize finetune -solver solver.prototxt -weights
quantize_results/quantize_train_test.caffemodel -gpu all
```

Assign the training dataset

```
fix_train_test.prototxt
```

Create a solver for finetuning

```
solver.prototxt
```

Run the command to start finetuning

```
vai_q_caffe quantize finetune -solver solver.prototxt -weights
quantize_results/quantize_train_test.caffemodel -gpu all
```

Run the command to deploy the finetuned model

```
vai_q_caffe deploy -model quantize_results/quantize/_train_test.prototxt
-weights finetuned_iter10000.caffemodel -gpu 0 -output-dir deploy_output
```

Quantizer for TensorFlow: vai_q_tensorflow

Vitis AI quantizer for TensorFlow — vai_q_tensorflow

vai_q_tensorflow: Required input files

- **frozen_graph.pb** — Floating-point frozen inference graph
- **input_fn** — Function to convert the calibration dataset
- **Calibration dataset** — A subset of the training set

Run AI quantizer for TensorFlow to generate a fixed-point model

```
vai_q_tensorflow quantize --input_frozen_graph frozen_graph.pb \
                          --input_fn           input_fn \
                          --output_dir         vai_q_output \
                          --input_nodes        ${input_nodes} \
                          --output_nodes       ${output_nodes} \
                          --calib_iter 50 \
                          [options]
```

Quantizer for TensorFlow: vai_q_tensorflow

Run AI quantizer for TensorFlow to generate a fixed-point model

```
vai_q_tensorflow quantize --input_frozen_graph frozen_graph.pb
                          --input_fn [OPTIONS]
```

vai_q_tensorflow: Output files

- **deploy_model.pb** — Quantized model for VAI compiler
- **quantize_eval_model.pb** — Quantized model for evaluation

Run the command to dump the quantize simulation results

```
vai_q_tensorflow dump \
--input_frozen_graph quantize_results/quantize_eval_model.pb \
--input_fn dump_input_fn \
--max_dump_batches 1 \
--dump_float 0 \
--output_dir quantize_results
```

Getting the Frozen Inference Graph

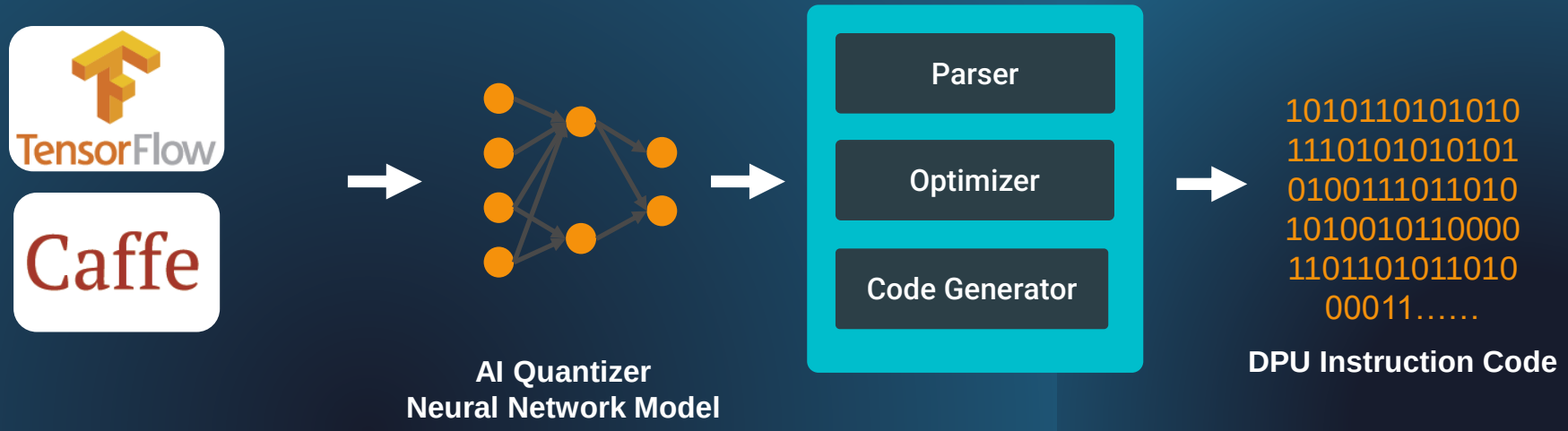
Run the command to dump the quantize simulation results

```
freeze_graph \
--input_graph /tmp/inception_v1_inf_graph.pb \
--input_checkpoint /tmp/checkpoints/model.ckpt-1000 \
--input_binary true \
--output_graph /tmp/frozen_graph.pb \
--output_node_names InceptionV1/Predictions/Reshape_1
```

For more information

```
freeze_graph --help
```

Vitis AI Compiler



Vitis AI compiler is a domain-specific compiler

Vitis AI compiler for Caffe	`vai_c_caffe`
Vitis AI compiler for TensorFlow	`vai_c_tensorflow`

Vitis AI Compiler Common Options for Cloud and Edge

Vitis AI compiler for Caffe	`vai_c_caffe`
Vitis AI compiler for TensorFlow	`vai_c_tensorflow`

Parameters	Description
--arch	DPU architecture configuration file for VAI_C compiler in JSON format. It contains the dedicated options for cloud and edge DPU during compilation.
--prototxt	Path of Caffe prototxt file for the compiler vai_c_caffe. This option is only required while compiling the quantized Caffe model generated by vai_q_caffe.
--caffemodel	Path of Caffe caffemodel file for the compiler vai_c_caffe. This option is only required while compiling the quantized Caffe model generated by vai_q_caffe.
--frozen_pb	Path of TensorFlow frozen protobuf file for the compiler vai_c_tensorflow. This option is only required by the quantized TensorFlow model generated by vai_q_tensorflow.
--output_dir	Path of output directory of vai_c_caffe and vai_c_tensorflow after the compilation process.
--net_name	Name of DPU kernel for network model after compilation by VAI_C.
--options	The list for the extra options for cloud or edge DPU in the format of 'key':'value'. If there are multiple options to be specified, they are separated by ‘,’, and if the extra option has no value, an empty string must be provided. For example: --options "{'cpu_arch':'arm32', 'dcf':'/home/edge-dpu/zynq7020.dcf', 'save_kernel':''}"

Vitis AI Compiler – Cloud Flow: Caffe

DPU-V1 (formerly known as xfDNN) front-end compilers

Four output files:

- compiler.json
- quantizer.json
- weights.h5
- meta.json

The basic Caffe compiler interface comes with simplified help

```
vai_c_caffe -help
**************************************************
* VITIS_AI Compilation - Xilinx Inc.
**************************************************
usage: vai_c_caffe.py [-h] [-p PROTOTXT] [-c CAFFEMODEL] [-a ARCH]
                      [-o OUTPUT_DIR] [-n NET_NAME] [-e OPTIONS]optional
arguments:
-h, --help show this help message and exit
-p PROTOTXT, --prototxt PROTOTXT
                      prototxt
-c CAFFEMODEL, --caffemodel CAFFEMODEL
                      caffe-model
-a ARCH, --arch ARCH json file
-o OUTPUT_DIR, --output_dir OUTPUT_DIR
                      output directory
-n NET_NAME, --net_name NET_NAME
                      prefix-name for the outputs
-e OPTIONS, --options OPTIONS
                      extra options
```

```
vai_c_caffe.py -p MODEL -c WEIGHT -a vai/dpuv1/tools/compile/arch.json
-o WORK -n cmd -e OPTIONS
```

Vitis AI Compiler – Cloud Flow: TensorFlow

DPU-V1 (formerly known as xfDNN) front-end compilers

The basic TensorFlow compiler interface comes with simplified help

```
************************************************
* VITIS_AI Compilation - Xilinx Inc.
************************************************
usage: vai_c_tensorflow.py [-h] [-f FROZEN_PB] [-a ARCH] [-o OUTPUT_DIR]
                           [-n NET_NAME] [-e OPTIONS] [-q]

optional arguments:
-h, --help show this help message and exit
-f FROZEN_PB, --frozen_pb FROZEN_PB
                      prototxt
-a ARCH, --arch ARCH json file
-o OUTPUT_DIR, --output_dir OUTPUT_DIR
                      output directory
-n NET_NAME, --net_name NET_NAME
                      prefix-name for the outputs
-e OPTIONS, --options OPTIONS
                      extra options
-q, --quant_info extract quant info
```

```
vai_c_tensorflow.py --frozen_pb deploy.pb --net_name cmd --options
"{'placeholdershape': {'input_tensor' : [1,224,224,3]}, 'quant_cfgfile':
'fix_info.txt'}" --arch arch.json --output_dir work/temp
```

XILINX

Introduction to the Deep Learning Processor Unit (DPU)

Vitis AI: Trained Model to Deployment in Minutes

Full AI Software Stack Support

Frameworks

PyTorch | TensorFlow | TensorFlow 2.0 | Caffe

Vitis AI models

108 pretrained, optimized reference models

Vitis AI development kit

AI Optimizer | AI Quantizer | AI Compiler | AI Profiler | AI Library

Xilinx runtime library (XRT)

Supports deploying custom AI models to Xilinx devices

Deep Learning Processing Unit (DPU)

CNN DPU | LSTM DPU | MLP DPU

Optimized ”processor-like” IP for groups of AI workloads

Deep Learning Processor Unit (DPU)

Deep Learning Processor Unit (DPU)

> Optimized engine for deep neural networks

> Parameterizable IP cores

> Pre-implemented on the hardware

> Accelerates the computing workloads of deep learning inference algorithms

Scalable DPU IPs for Various Xilinx Platforms

DPUv4E, Throughput optimized
- VCK5000, 384AIE-core, max 8PE
- Popular CNN models, MLPerf
- Higher throughput with low latency to compete against T4

DPU for Versal ACAP – New in Vitis AI 1.4
- Higher performance, lower latency
- Configurable IPs
- Shared AI quantizer, compiler, VART, models, etc.

DPUv3INT8, Throughput optimized
- Alveo U200 / U250
- Optimized for 100% INT8 models and ResNet/Inception
- High Throughput at medium latency

xvDPU, Latency optimized
- VCK190, 32 to 256 cores, 1-6 batch
- Popular CNN models,
- Scalable with power & cost

DPUv3E, Throughput optimized (HBM)
- Alveo U280 / U50
- Optimized for ResNet/Inception models and HBM devices
- Low-cost and high power efficiency solution

DPUv3ME, Latency optimized (HBM)
- Alveo U280 /U50
- Optimized for MobileNet-like models and HBM devices
- Extreme low latency with high-resolution images

DPUv2, Scalability optimized
- Zynq-7000, ZU+ MPSoC, Kria SoM
- Popular CNN models
- Scalable with power & cost

High

Throughput

Low

7nm

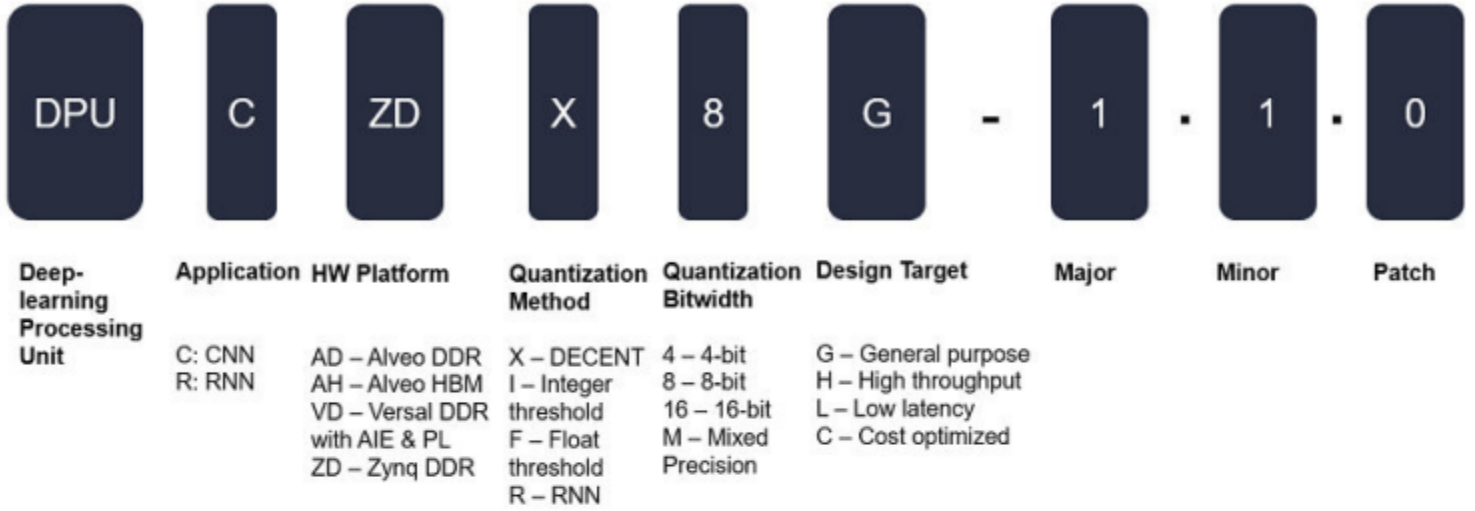
Latency

16/28nm

High

AMD XILINX

DPU Naming Convention



Example	DPU	Application	Hardware Platform	Quantization Method	Quantization Bitwidth	Design Target	Major	Minor	Patch	DPU Name
DPUv1	DPU	C	AD	X	8	G	3	0	0	DPUCADX8G-3.0.0
DPUv2	DPU	C	ZD	X	8	G	1	4	1	DPUCZDX8G-1.4.1
DPUv3e	DPU	C	AH	X	8	H	1	0	0	DPUCAHX8H-1.0.0
DPUv3me	DPU	C	AH	X	8	L	1	0	0	DPUCAHX8L-1.0.0
DPUv3int8	DPU	C	AD	F	8	H	1	0	0	DPUCADF8H-1.0.0
XRNN	DPU	R	AH	R	16	L	1	0	0	DPURAHR16L-1.0.0

Use Cases

Deep Learning Processor Unit (DPU)

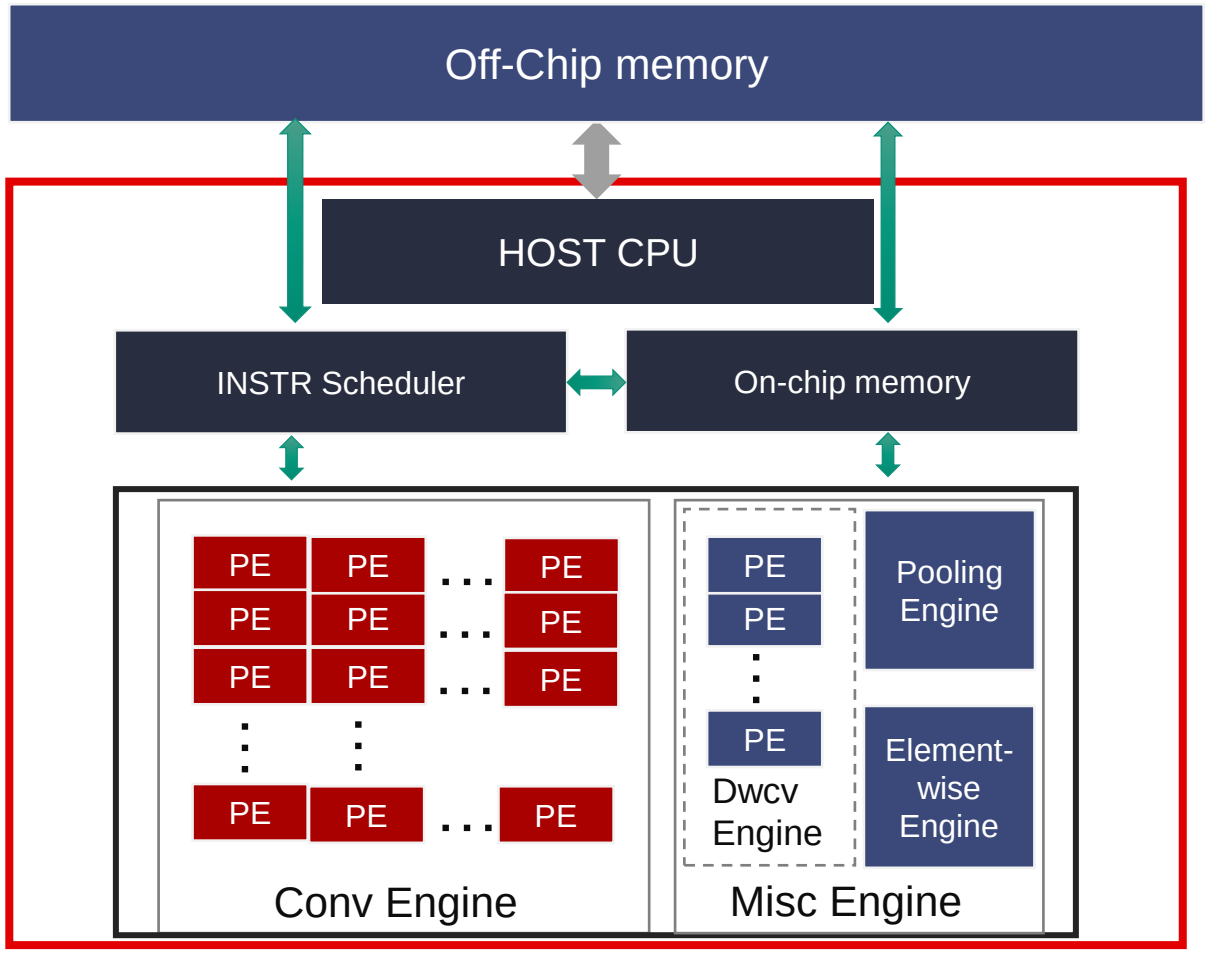
Automated Driving

- Ultra-low latency DPU
- High hardware parallelism for convolution and depth-wise convolution

Data Center

- Flexible pipeline with optimized architecture
- Multi-engine organization for super logic region (SLR) implementation

CNN DPU for Zynq SoC / MPSoC



- Flexible and Configurable DPU
  - Configurable hardware architecture includes：Z7020 to Z7100, ZU2 to ZU11
  - Relu, Relu6, LeakyRelu
  - Max/Average pooling 2x2~8x8
  - Ram usage for higher performance or lower resource utilization
  - Core number, Bram or Uram, More DSP or less DSP
  - Support channel augmentation to improve performance
  - Support low power consumption feature

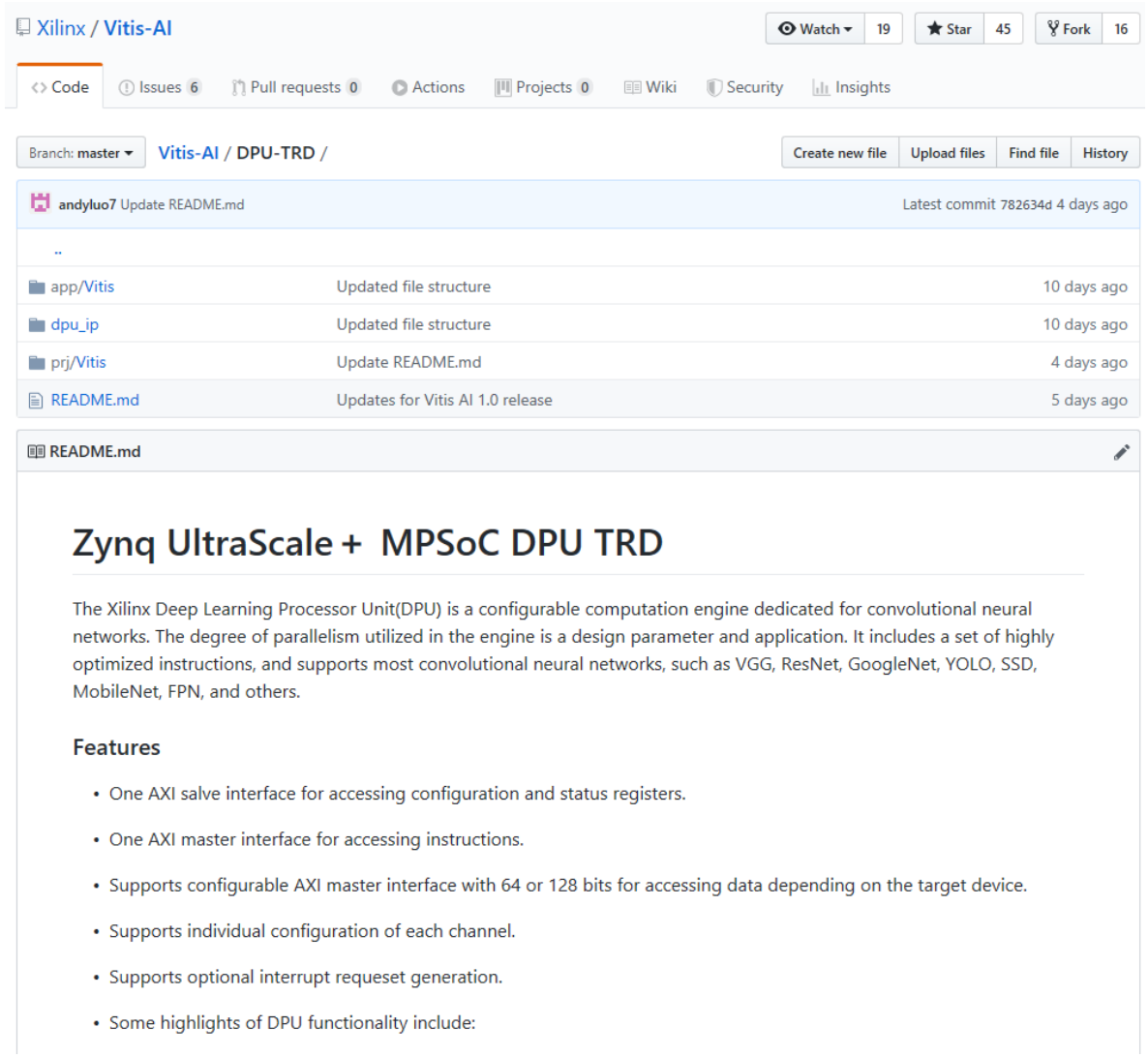
DPU for Zynq Ultrascale+ MPSoC Product Guide

DPU_EU:Preference

DPU LUTS and resisters

Arch	LUTs	Registers	BRAM	DSP
B512	21171	33572	69.5	66
B800	22900	33752	87	102
B1024	26341	49823	101.5	130
B1152	25250	49588	117.5	146
B1600	29270	60739	123	202
B2304	32684	72850	161.5	290
B3136	35797	86132	203.5	394
B4096	41412	99791	249.5	514

DPU TRD



Xilinx / Vitis-AI

Watch 19 | Star 45 | Fork 16

Code | Issues 6 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights

Branch: master ▾ Vitis-AI / DPU-TRD / Create new file | Upload files | Find file | History

andyluo7 Update README.md — Latest commit 782634d 4 days ago

..		
app/Vitis	Updated file structure	10 days ago
dpu_ip	Updated file structure	10 days ago
prj/Vitis	Update README.md	4 days ago
README.md	Updates for Vitis AI 1.0 release	5 days ago

README.md

Zynq UltraScale + MPSoC DPU TRD

The Xilinx Deep Learning Processor Unit(DPU) is a configurable computation engine dedicated for convolutional neural networks. The degree of parallelism utilized in the engine is a design parameter and application. It includes a set of highly optimized instructions, and supports most convolutional neural networks, such as VGG, ResNet, GoogleNet, YOLO, SSD, MobileNet, FPN, and others.

Features

- One AXI salve interface for accessing configuration and status registers.
- One AXI master interface for accessing instructions.
- Supports configurable AXI master interface with 64 or 128 bits for accessing data depending on the target device.
- Supports individual configuration of each channel.
- Supports optional interrupt requeset generation.
- Some highlights of DPU functionality include:

https://github.com/Xilinx/Vitis-AI/tree/master/DPU-TRD

Introduction to Vitis AI Library

XILINX.

Vitis AI API

- A set of high-level API based libraries across different vision tasks: classification, detection, segmentation and etc.
  - Reference applications to help customers' fast prototyping
  - Optimized codes used in AI applications and products

User Applications

Demo and Reference applications

Framework

Vitis AI Library

Classification

Detection

Segmentation

...

OS level packages

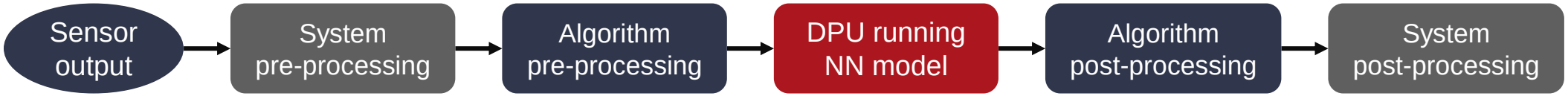
Ease-of-Use

Optimized

Open

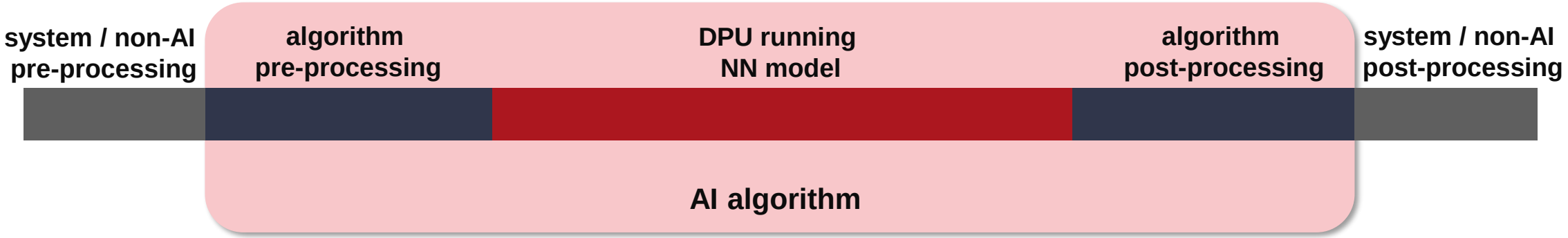
General Processing Flow

- A typical abstraction of processing flow:



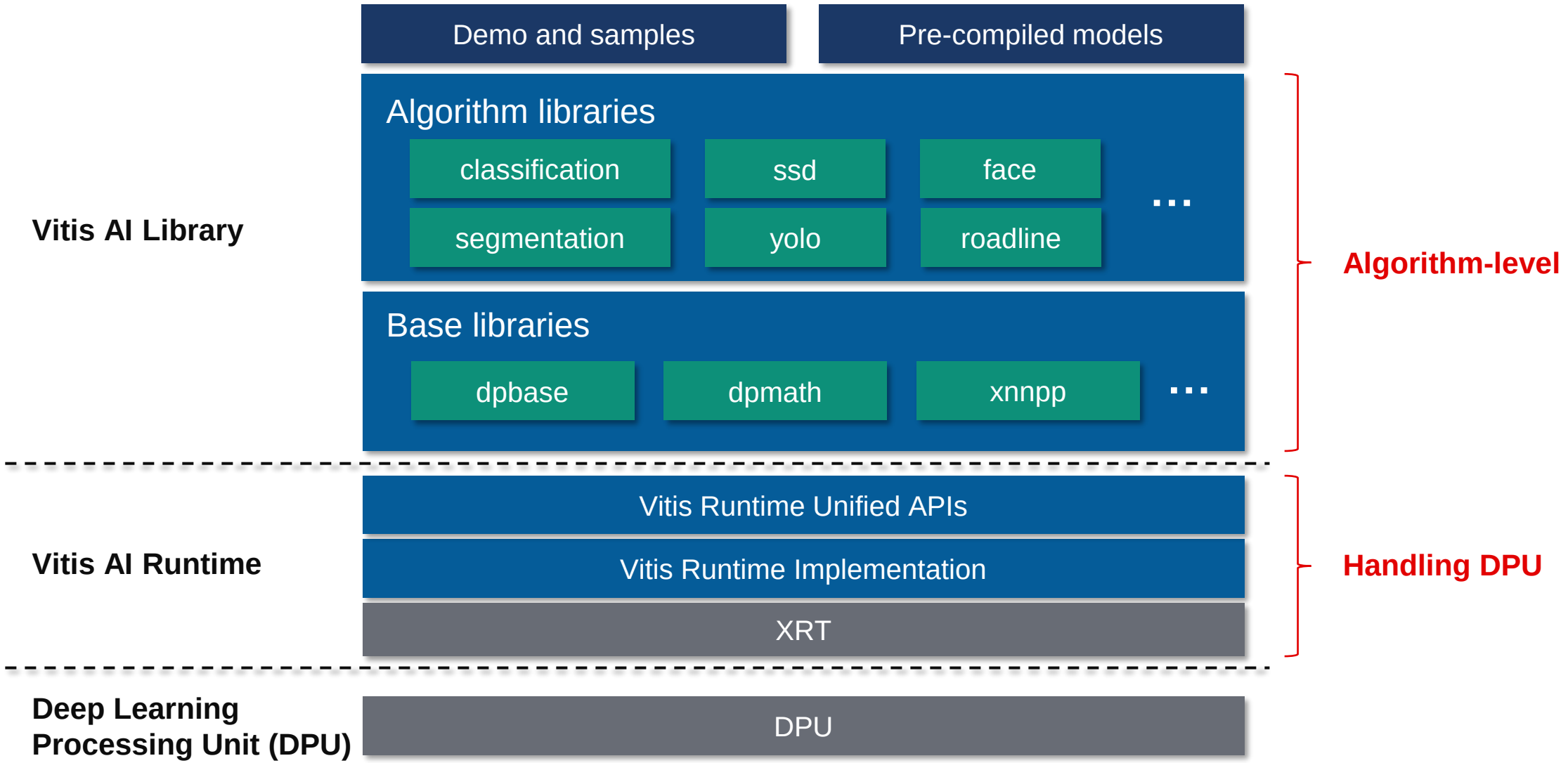
- Algorithm-level processing
  - Data normalization before sending to DPU
  - Post processing (e.g. bounding boxes decoding in detection)
- Additional system-level workloads for AI inference
  - Color conversion / resizing
  - Path planning / control / status update

What Vitis AI Library Provides



- Algorithm-level optimization
- Open and easy to extend
- Directly support models in Vitis AI Model Zoo

Vitis AI Library



XILINX.

Installation and Support Evaluation Boards

- Vitis AI Library package
  - Freely downloaded from GitHub
    - https://github.com/Xilinx/Vitis-AI/blob/master/demo/Vitis-AI-Library/README.md
    - https://github.com/Xilinx/Vitis_Libraries
    - Vitis AI Library User Guide : https://docs.xilinx.com/viewer/book-attachment/qcgEHA~q4hM768dpOQHTKg/IdNhwhuxB19ntMheAlWgFQ
  - Edge Evaluation boards supported
    - Xilinx ZCU102 Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit (xilinx.com)
    - Xilinx ZCU104 Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit (xilinx.com)
    - Xilinx Kria™ KV260 Kria KV260 Vision AI Starter Kit (xilinx.com)
    - Xilinx VCK190 Versal AI Core Series VCK190 Evaluation Kit (xilinx.com)
  - Cloud Board supported
    - Xilinx Alveo U50LV/U200/U250/U55C Data Center development kit Alveo (xilinx.com)
    - Versal AI Core series VCK5000 Data Center development kit VCK5000 Versal Development Card (xilinx.com)

Easy-to-Use APIs to Deploy Full Algorithm

1. **Seamlessly compatible with AI Model Zoo**
   - Classification, detection, segmentation and others
2. **Samples for fast prototyping**
   - Every algorithm has several samples, image, video and performance benchmarking
   - Complicated samples can be refer to AI Demo Zoo which is also built on AI LIbrary
3. **High-level APIs to deploy algorithm**
   - No need to consider algorithm-level processing and DPU running codes
4. **Support multiple deploying approaches**
   - In addition to high-level APIs, DPU running can be also controlled by users

Algorithm libraries: Unified API Interface

Get an instance of derived class

Get width and height for required by Algorithm

DPU run and get results

```
class YOLOv3 {
public:
  static std::unique_ptr<YOLOv3> create(const std::string &model_name,
                                        bool need_preprocess = true);
protected:
  explicit YOLOv3();
  YOLOv3(const YOLOv3 &) = delete;
public:
  virtual ~YOLOv3();
public:
  virtual int getInputWidth() const = 0;
  virtual int getInputHeight() const = 0;

  virtual YOLOv3Result run(const cv::Mat &image) = 0;

};
```


XILINX.

Implement YOLOv3 in AI Library

- Most important is to implement run() method

Pre-processing

Running DPU

Post-processing

```
YOLOv3Result YOLOv3Imp::run(const cv::Mat &input_image) {
  cv::Mat image;
  int sWidth = getInputWidth();
  int sHeight = getInputHeight();
  auto mAP = configurable_dpu_task_->getConfig().yolo_v3_param().test_map();
  LOG_IF(INFO, false) << "tf_flag_ " << tf_flag_ << " " //
                      << "mAp " << mAP << " "           //
                      << std::endl;
  if (mAP) {
    if (!tf_flag_) {
      int channel = configurable_dpu_task_->getInputTensor()[0][0].channel;
      float scale = xilinx::ai::tensor_scale(
          configurable_dpu_task_->getInputTensor()[0][0]);
      int8_t *data =
          (int8_t *)configurable_dpu_task_->getInputTensor()[0][0].data;
      LOG_IF(INFO, false) << "scale " << scale << " "     //
                          << "sWidth " << sWidth << " "   //
                          << "sHeight " << sHeight << " " //
                          << std::endl;
      yolov3::convertInputImage(input_image, sWidth, sHeight, channel, scale, data);
    } else {
      image = yolov3::letterbox_tf(input_image, sWidth, sHeight).clone();
      configurable_dpu_task_->setInputImageRGB(image);
    }
  } else {
    auto size = cv::Size(sWidth, sHeight);
    if (size != input_image.size()) {
      cv::resize(input_image, image, size, 0, 0, cv::INTER_LINEAR);
    } else {
      image = input_image;
    }
    //  convert_RGB(image);
    __TIC__(YOLOV3_SET_IMG)
    configurable_dpu_task_->setInputImageRGB(image);
    __TOC__(YOLOV3_SET_IMG)
  }
  __TIC__(YOLOV3_DPU)
  configurable_dpu_task_->run(0);
  __TOC__(YOLOV3_DPU)

  __TIC__(YOLOV3_POST_ARM)

  auto ret = xilinx::ai::yolov3_post_process(
      configurable_dpu_task_->getInputTensor()[0],
      configurable_dpu_task_->getOutputTensor()[0],
      configurable_dpu_task_->getConfig(), input_image.cols, input_image.rows);

  __TOC__(YOLOV3_POST_ARM)
  return ret;
}
```

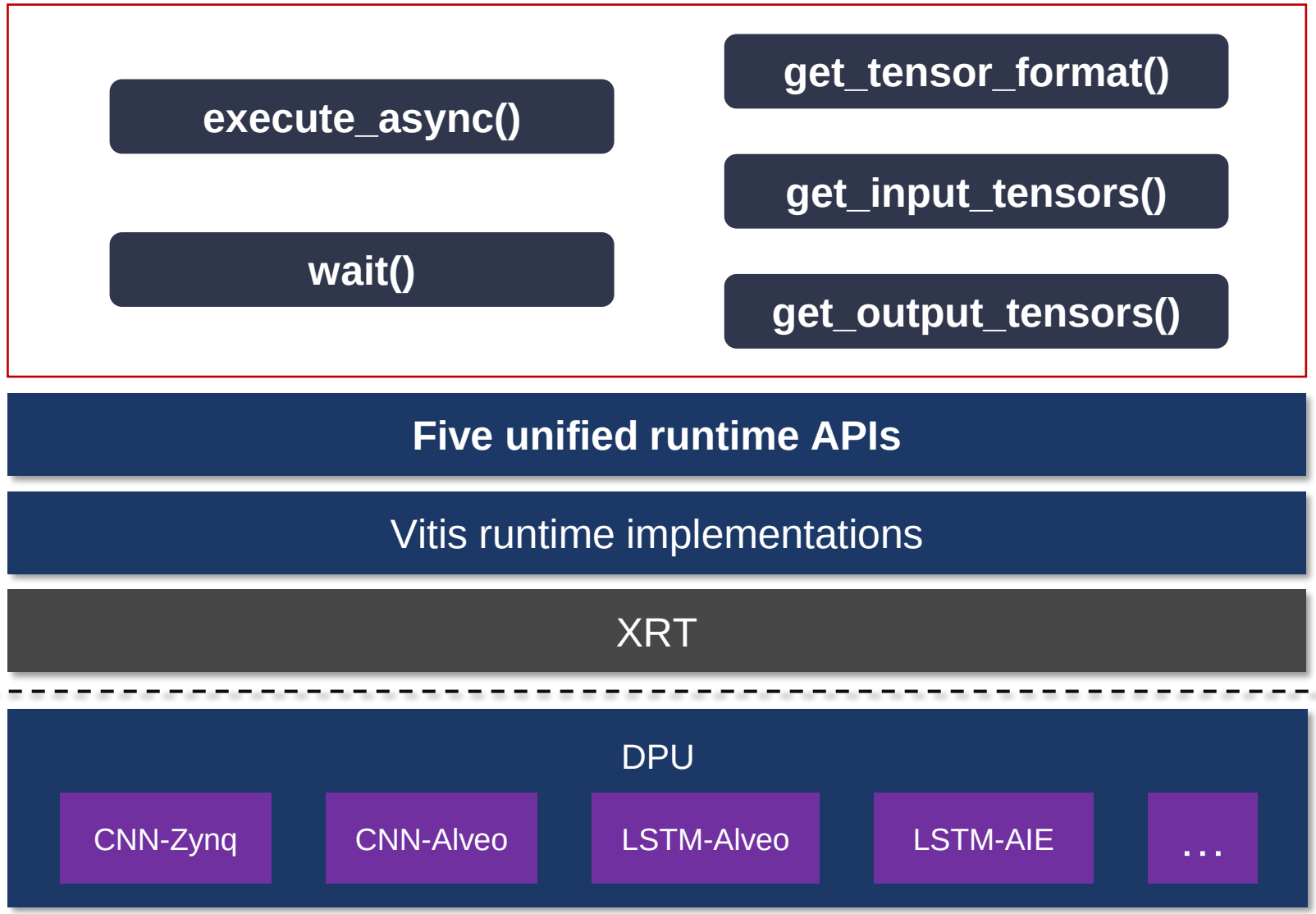

XILINX.

Deploy Yolov3 Using AI Library

```
cv::Mat img = cv::imread(argv[2]);
auto yolo = xilinx::ai::YOLOv3::create(argv[1], true);

int width = yolo->getInputWidth();
int height = yolo->getInputHeight();
cv::Mat img_resize;
cv::resize(img, img_resize, cv::Size(width, height), 0, 0, cv::INTER_LINEAR);

auto results = yolo->run(img_resize);

for (auto &box : results.bboxes) {

  int label = box.label;
  float x = box.width * img.cols;
  float y = box.height * img.rows;
  float confidence = box.score;

  cout << "RESULT: " << label << "\t" << xmin << "\t" << ymin << "\t" << xmax
       << "\t" << ymax << "\t" << confidence << "\n";
  rectangle(img, Point(xmin, ymin), Point(xmax, ymax), Scalar(0, 255, 0), 1,
            1, 0);
}
```

XILINX.

Unified Vitis AI runtime APIs



Unified Vitis AI runtime with same five APIs across edge and cloud

XILINX.

Deploy Resnet50 Using Vitis AI runtime extension APIs

Create extension runner and get information about the model

Pre-processing

Dpu running

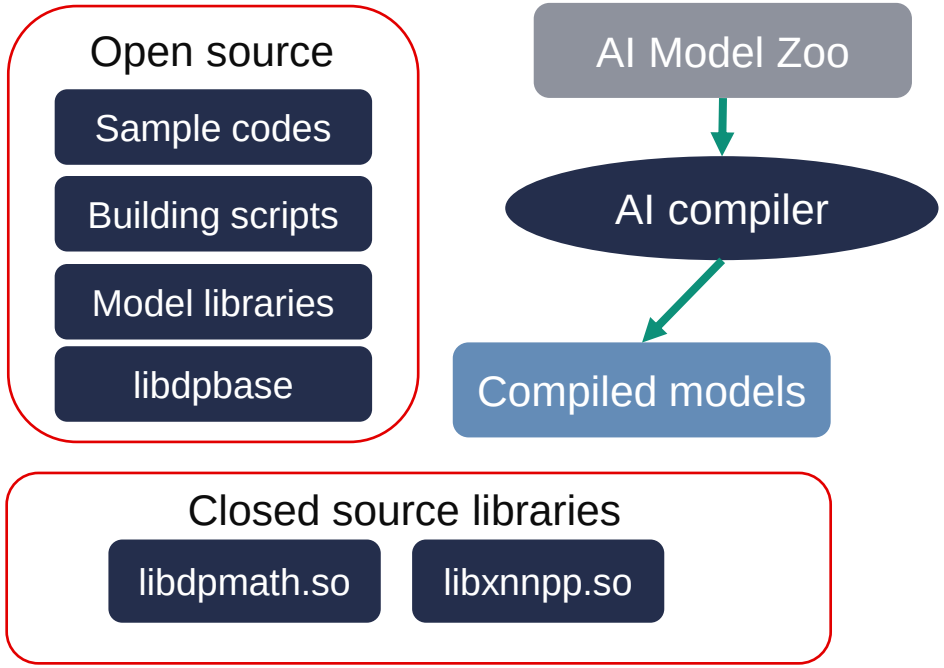
Post-processing

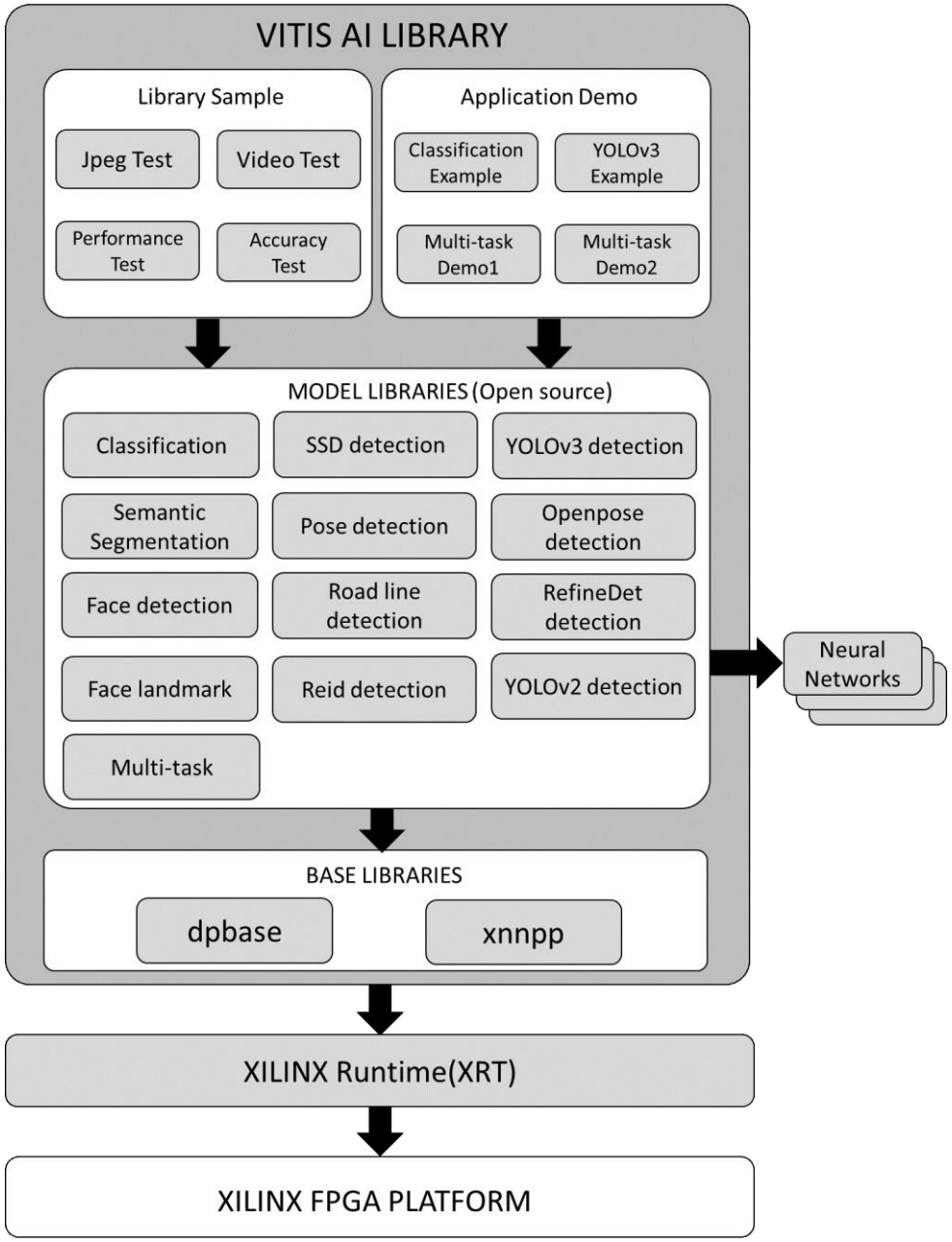
```
{
  const auto model_dir_name = std::string("/usr/share/vitis_ai_library/models/resnet50");
  auto runners = vitis::ai::DpuRunner::create_dpu_runner(model_dir_name);
  auto runner = dynamic_cast<vart::dpu::DpuRunnerExt*>(runners[0].get());

  auto input_scale = runner->get_input_scale();
  auto output_scale = runner->get_output_scale();
  auto image_file_name = std::string(argv[1]);
  cv::Mat input_image = read_image(image_file_name);
  auto input_tensors = runner->get_input_tensors();
  auto input_tensor = input_tensors[0];
  auto height = input_tensor->get_dim_size(1);
  auto width = input_tensor->get_dim_size(2);
  auto input_size = cv::Size(width, height);
  auto input_tensor_buffer = runner->get_inputs()[0];
  auto output_tensor_buffer = runner->get_outputs()[0];
  // preprocess, i.e. resize if necessary
  cv::Mat image = preprocess_image(input_image, input_size);
  // set the input image and preprocessing
  void* data_in = nullptr;
  size_t size_in = 0u;
  std::tie(data_in, size_in) =
      input_tensor_buffer->data(std::vector<int>{0, 0, 0, 0});
  setImageBGR(image, data_in, input_scale[0]);
  auto v =
      runner->execute_async({input_tensor_buffer}, {output_tensor_buffer});
  auto status = runner->wait((int)v.first, -1);
  CHECK_EQ(status, 0) << "failed to run dpu";
  // post process
  auto topk = post_process(output_tensor_buffer, output_scale[0]);
  // print the result
  print_topk(topk);
}
```


XILINX.

Open-Source Architecture

- Model libraries are open-source, and users can easily replace models or write a new algorithm
- Users can port to other platforms easily
- Support both edge and cloud platforms





XILINX

Vitis AI Library Samples: test_jpeg_yolov3

```
root@xilinx-zcu102-2019_1:/usr/share/XILINX_AI_SDK/samples/yolov3#./test_jpeg_yolov3_voc_416x416
 sample_yolov3_voc_416x416.jpg
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0923 02:13:51.147414 15392 process_result.hpp:78] RESULT: 6    -9.86494        133.408 139.6652
55.254  0.999673
I0923 02:13:51.147737 15392 process_result.hpp:78] RESULT: 6    113.796 142.11  190.103 182.4020
.990521
I0923 02:13:51.147800 15392 process_result.hpp:78] RESULT: 6    402.753 129.565 512     251.4110
.970362
I0923 02:13:51.147862 15392 process_result.hpp:78] RESULT: 6    351.843 144.018 415.105 168.4570
.873677
```

Fast implementation of YOLOv3 demo by very simple code

```
int main(int argc, char *argv[]) {
  return xilinx::ai::main_for_jpeg_demo(
      argc, argv,
      [] {
        return xilinx::ai::YOLOv3::create(xilinx::ai::YOLOV3_VOC_416x416);
      },
      process_result);
}
```


XILINX.

Thank You