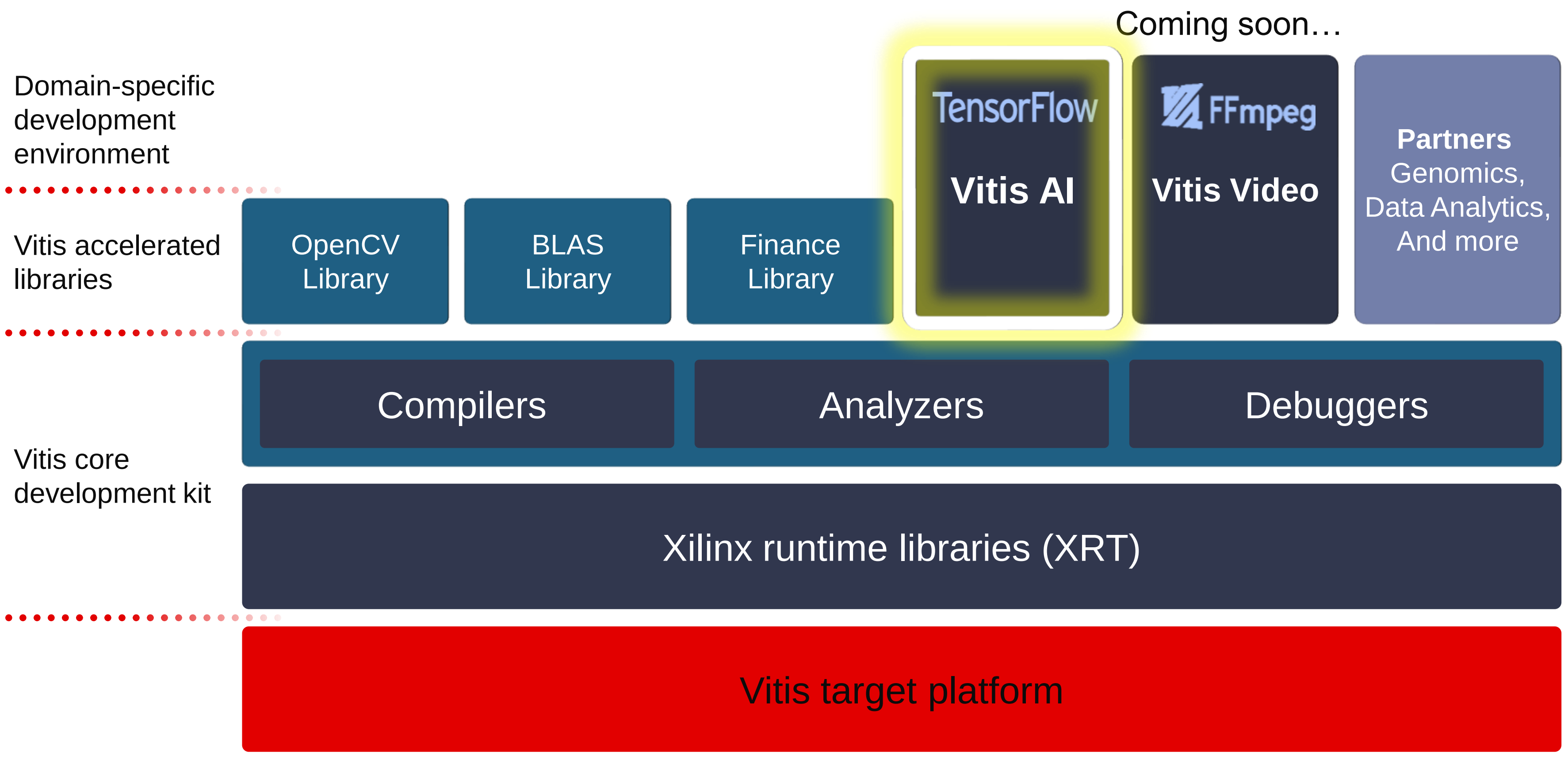
Welcome!

Vitis AI Deepdive

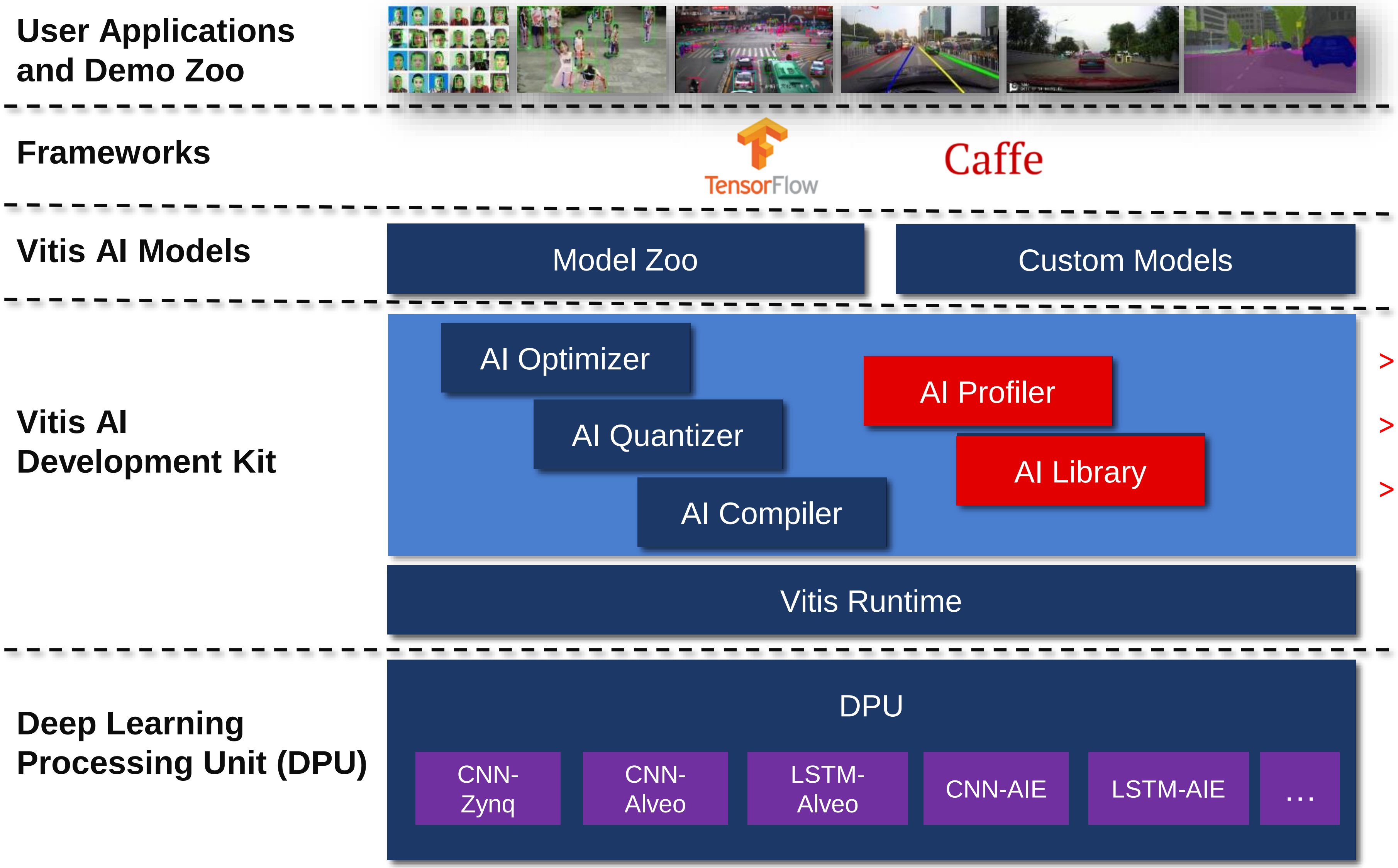
Fan Zhang Ph.D
Software and AI Technical Marketing

Vitis AI: Unified AI Inference Solution Stack



Coming soon…
Domain-specific development environment
Vitis accelerated libraries
Vitis core development kit
OpenCV Library
BLAS Library
Finance Library
TensorFlow
Vitis AI
FFmpeg
Vitis Video
Partners
Genomics, Data Analytics, And more
Compilers
Analyzers
Debuggers
Xilinx runtime libraries (XRT)
Vitis target platform

Vitis AI: Unified AI Inference Solution Stack

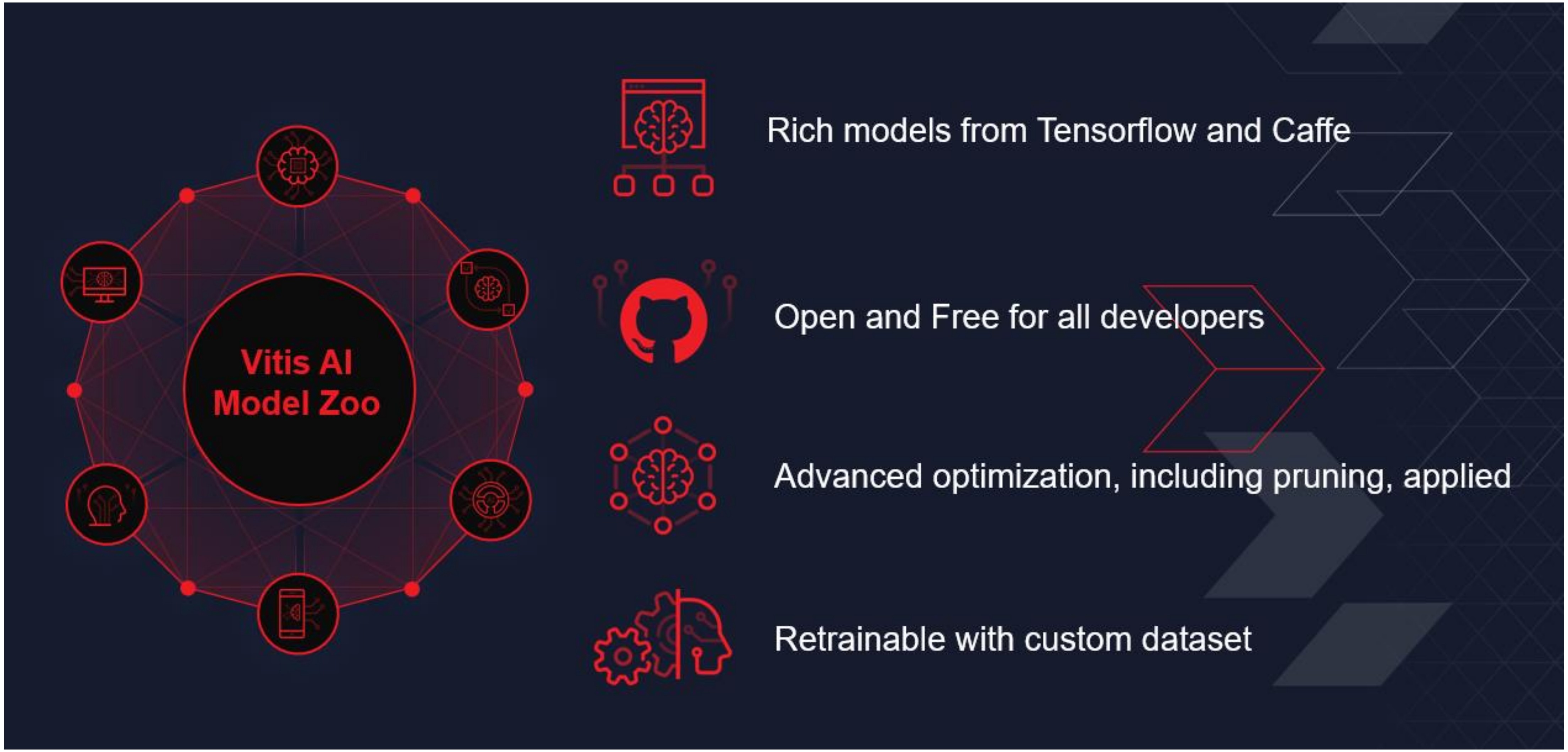


> **Support both edge and cloud**

> **Support AI model zoo**

> **Several releases before this Vitis AI release**

Model Zoo



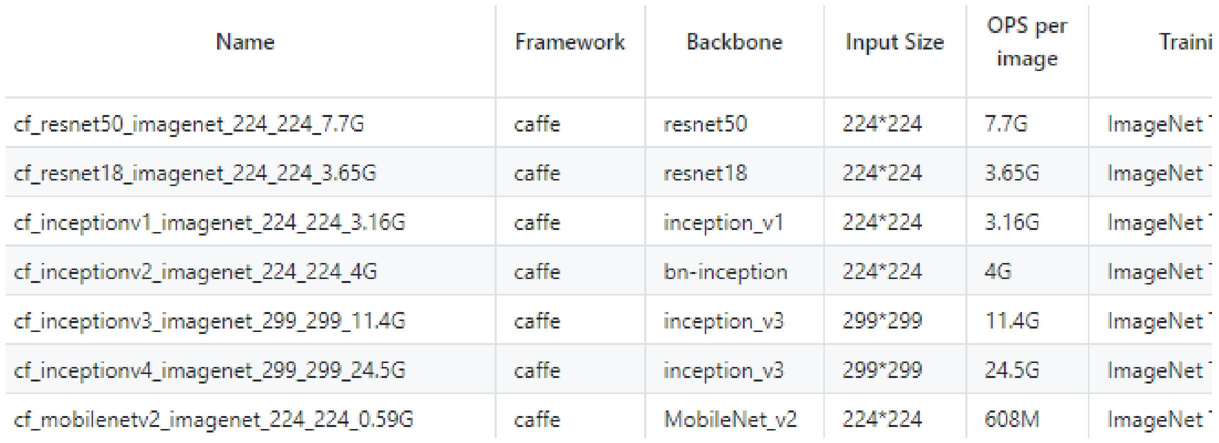
Rich models from Tensorflow and Caffe

Open and Free for all developers

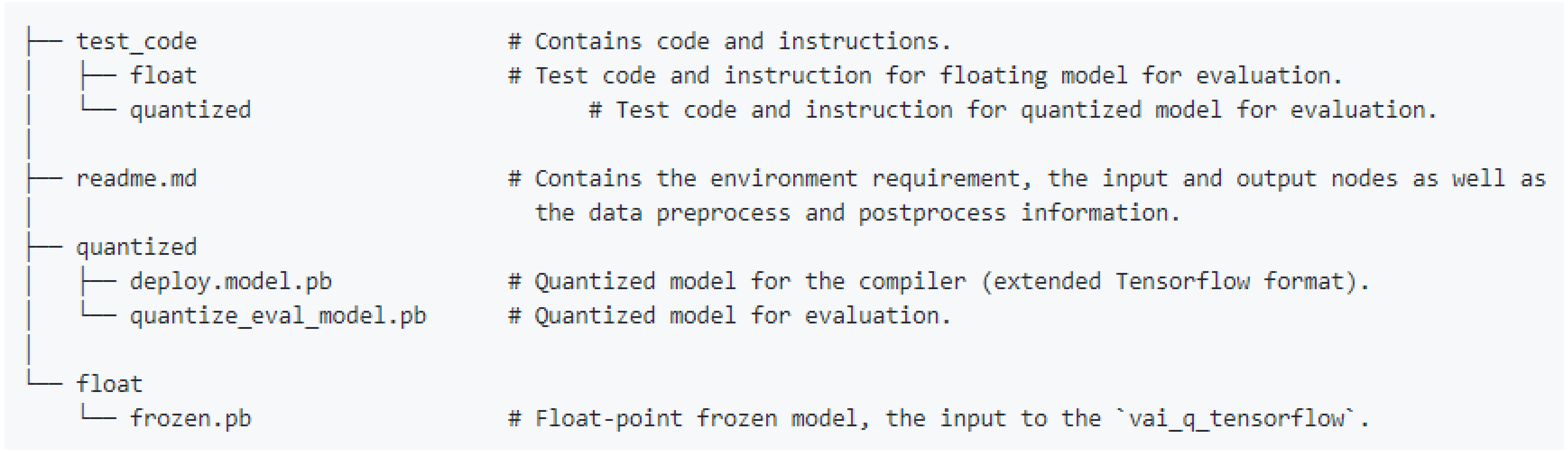
Advanced optimization, including pruning, applied

Retrainable with custom dataset

Model Zoo

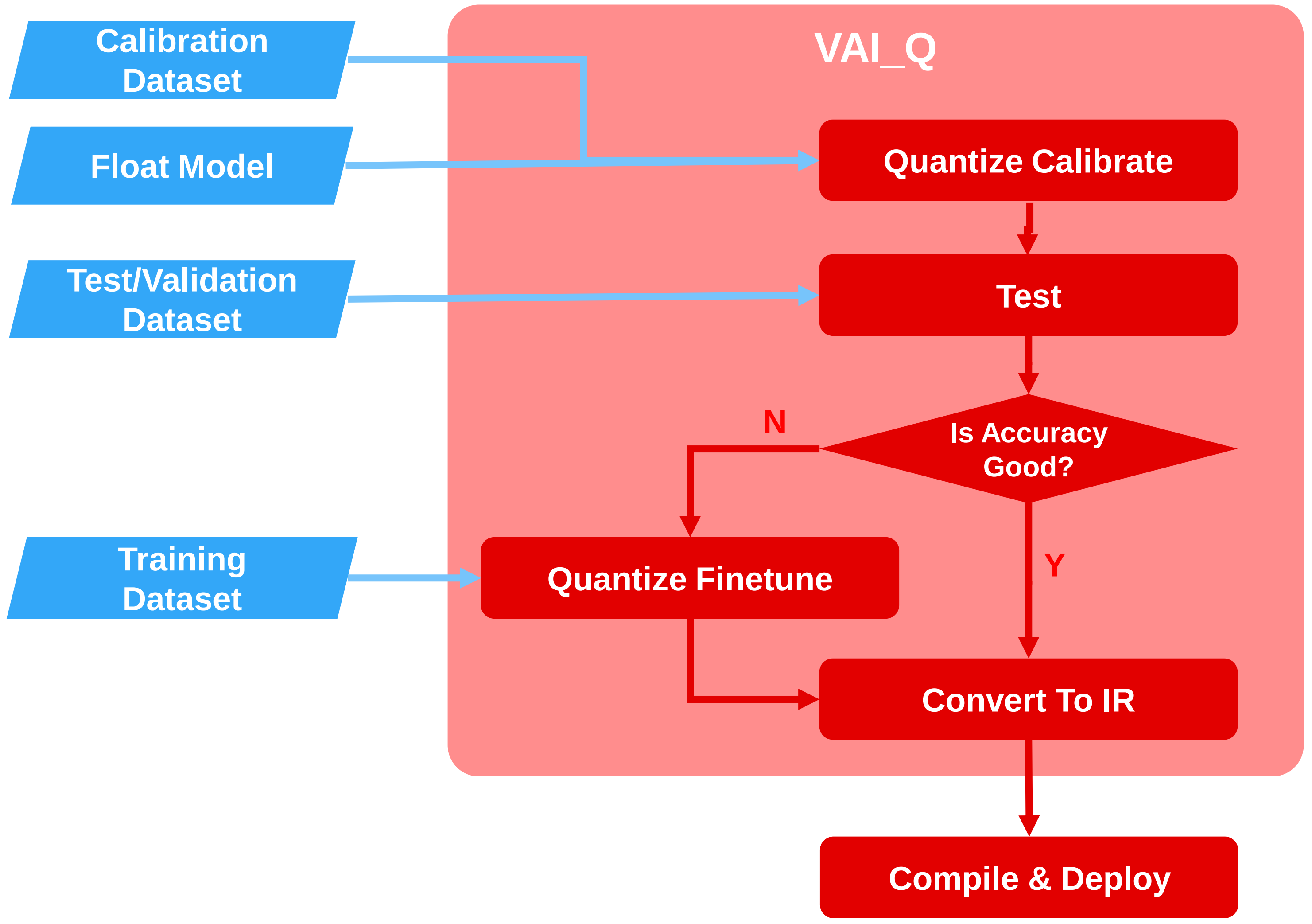


Name	Framework	Backbone	Input Size	OPS per image	Traini
cf_resnet50_imagenet_224_224_7.7G	caffe	resnet50	224*224	7.7G	ImageNet
cf_resnet18_imagenet_224_224_3.65G	caffe	resnet18	224*224	3.65G	ImageNet
cf_inceptionv1_imagenet_224_224_3.16G	caffe	inception_v1	224*224	3.16G	ImageNet
cf_inceptionv2_imagenet_224_224_4G	caffe	bn-inception	224*224	4G	ImageNet
cf_inceptionv3_imagenet_299_299_11.4G	caffe	inception_v3	299*299	11.4G	ImageNet
cf_inceptionv4_imagenet_299_299_24.5G	caffe	inception_v3	299*299	24.5G	ImageNet
cf_mobilenetv2_imagenet_224_224_0.59G	caffe	MobileNet_v2	224*224	608M	ImageNet

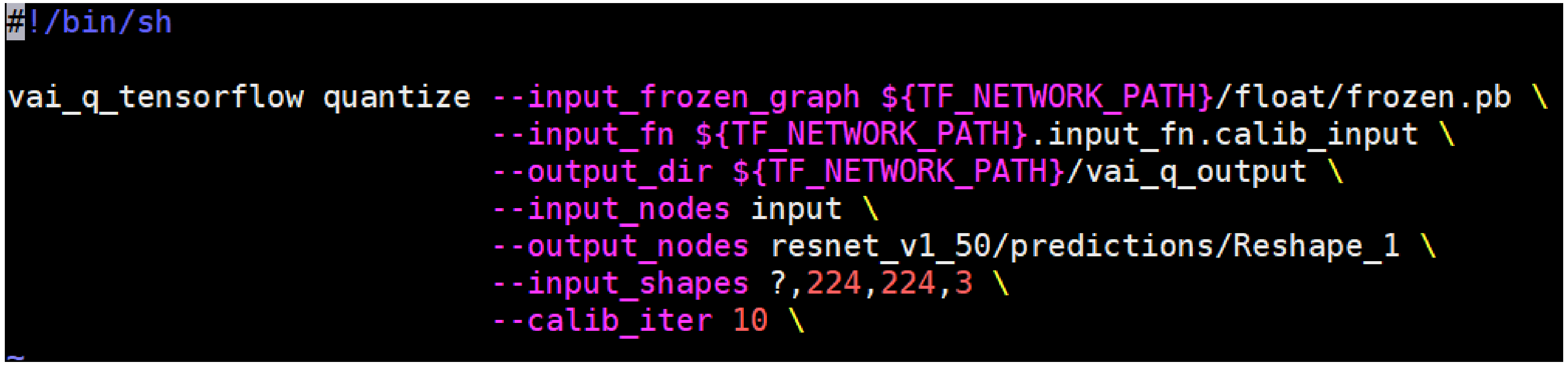


```
├── test_code                       # Contains code and instructions.
│   ├── float                       # Test code and instruction for floating model for evaluation.
│   └── quantized                       # Test code and instruction for quantized model for evaluation.
│
├── readme.md                       # Contains the environment requirement, the input and output nodes as well as
│                                     the data preprocess and postprocess information.
├── quantized
│   ├── deploy.model.pb             # Quantized model for the compiler (extended Tensorflow format).
│   └── quantize_eval_model.pb      # Quantized model for evaluation.
│
└── float
    └── frozen.pb                   # Float-point frozen model, the input to the `vai_q_tensorflow`.
```

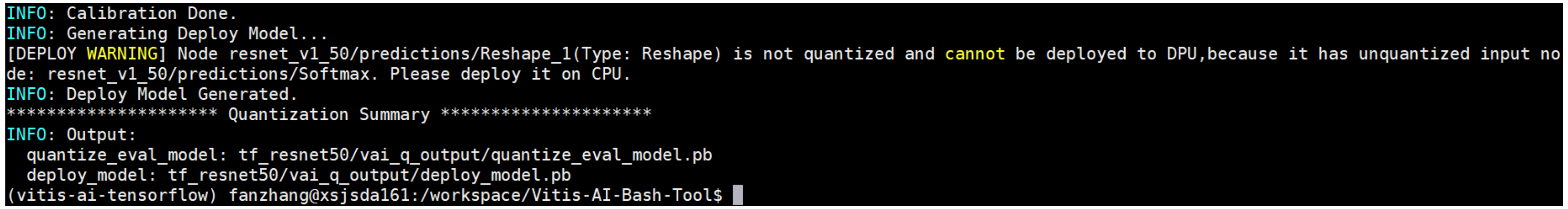
AI Quantizer: Overall Workflow



AI Quantizer

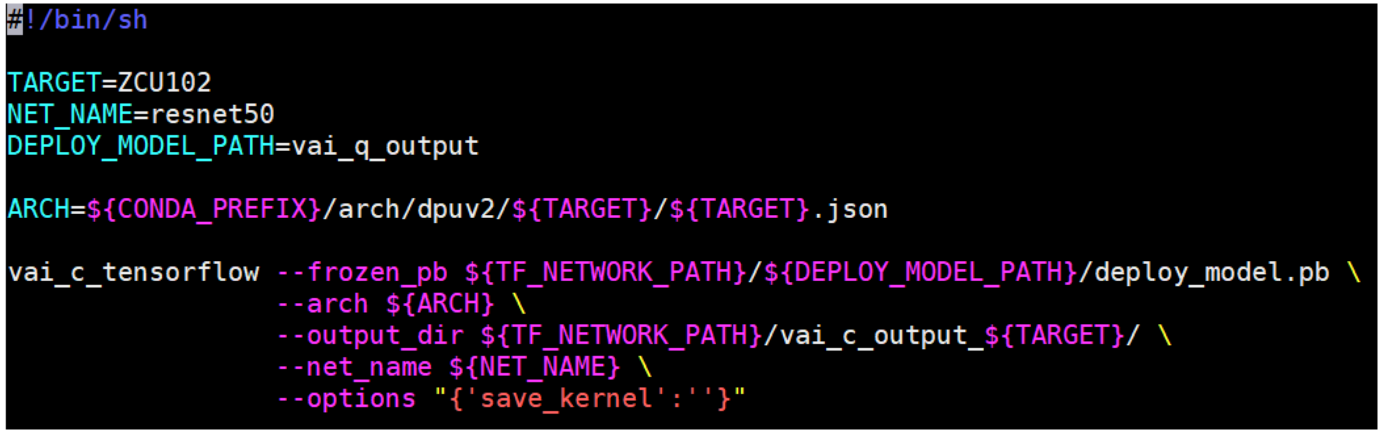


```
#!/bin/sh

vai_q_tensorflow quantize --input_frozen_graph ${TF_NETWORK_PATH}/float/frozen.pb \
                          --input_fn ${TF_NETWORK_PATH}.input_fn.calib_input \
                          --output_dir ${TF_NETWORK_PATH}/vai_q_output \
                          --input_nodes input \
                          --output_nodes resnet_v1_50/predictions/Reshape_1 \
                          --input_shapes ?,224,224,3 \
                          --calib_iter 10 \
~
```



```
INFO: Calibration Done.
INFO: Generating Deploy Model...
[DEPLOY WARNING] Node resnet_v1_50/predictions/Reshape_1(Type: Reshape) is not quantized and cannot be deployed to DPU,because it has unquantized input no
de: resnet_v1_50/predictions/Softmax. Please deploy it on CPU.
INFO: Deploy Model Generated.
******************** Quantization Summary ********************
INFO: Output:
  quantize_eval_model: tf_resnet50/vai_q_output/quantize_eval_model.pb
  deploy_model: tf_resnet50/vai_q_output/deploy_model.pb
(vitis-ai-tensorflow) fanzhang@xsjsda161:/workspace/Vitis-AI-Bash-Tool$
```

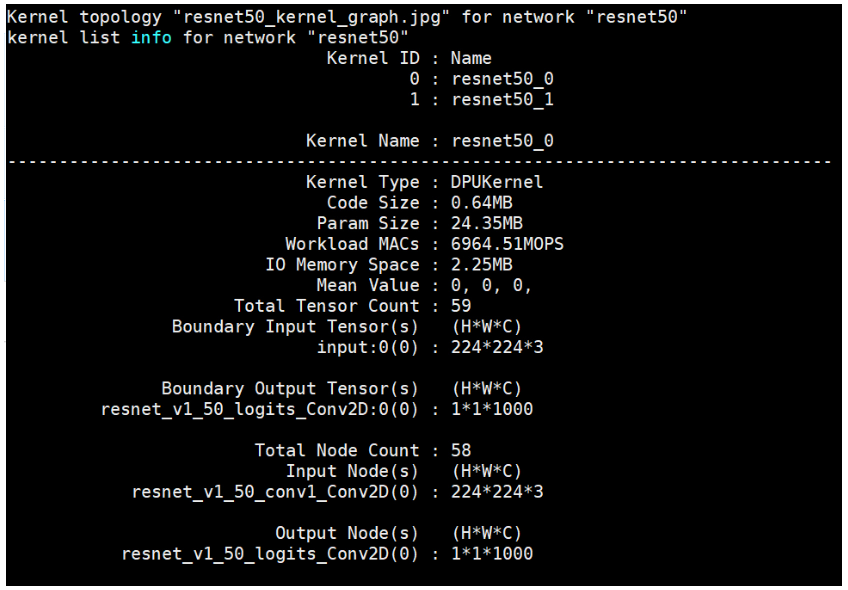
AI Compiler



```
#!/bin/sh

TARGET=ZCU102
NET_NAME=resnet50
DEPLOY_MODEL_PATH=vai_q_output

ARCH=${CONDA_PREFIX}/arch/dpuv2/${TARGET}/${TARGET}.json

vai_c_tensorflow --frozen_pb ${TF_NETWORK_PATH}/${DEPLOY_MODEL_PATH}/deploy_model.pb \
                 --arch ${ARCH} \
                 --output_dir ${TF_NETWORK_PATH}/vai_c_output_${TARGET}/ \
                 --net_name ${NET_NAME} \
                 --options "{'save_kernel':''}"
```

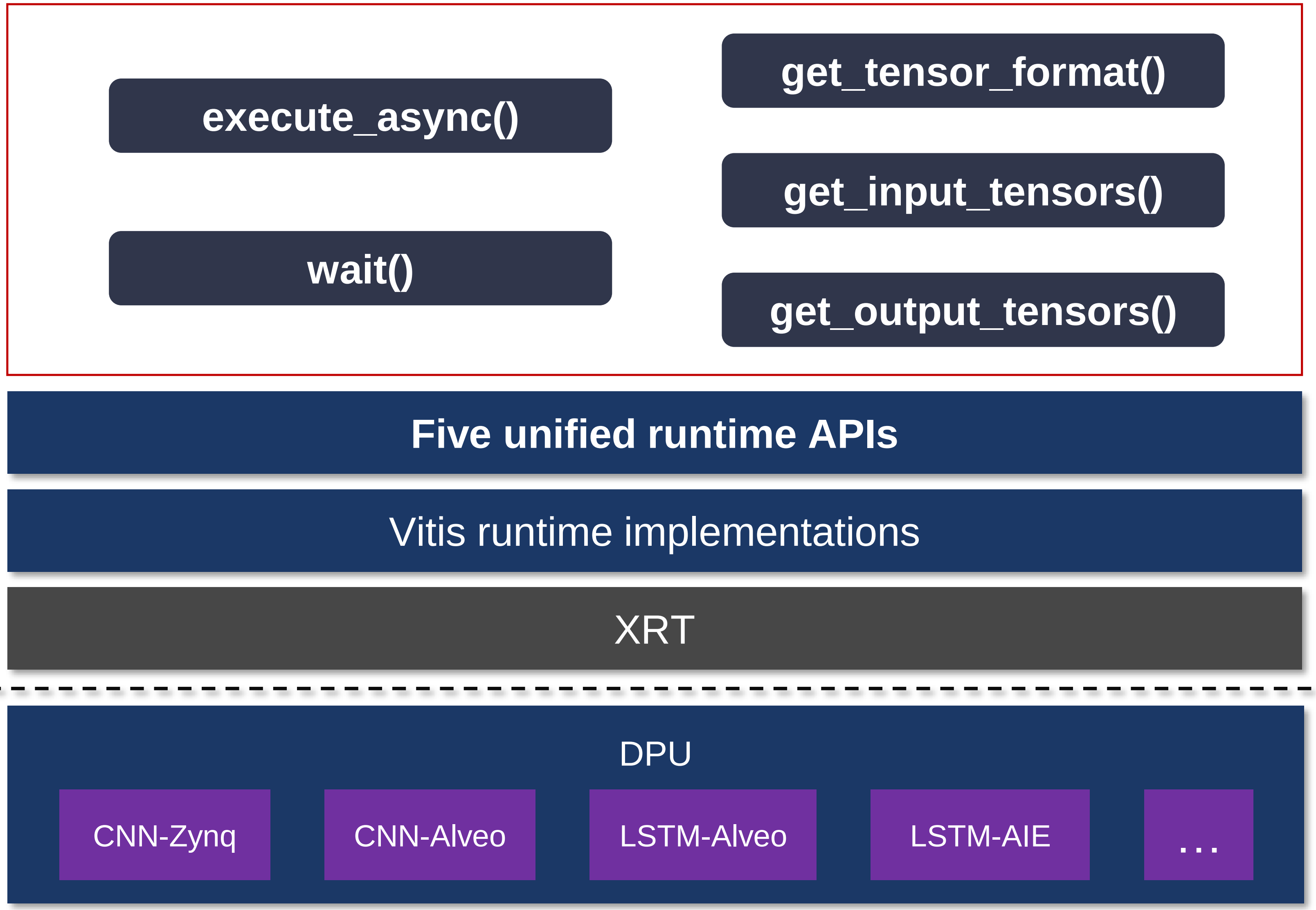
> **Compilation is hardware dependent, need to assign a platform in advance.**

> **The output after compilation is the deep learning model itself, pre/post processing are excluded.**



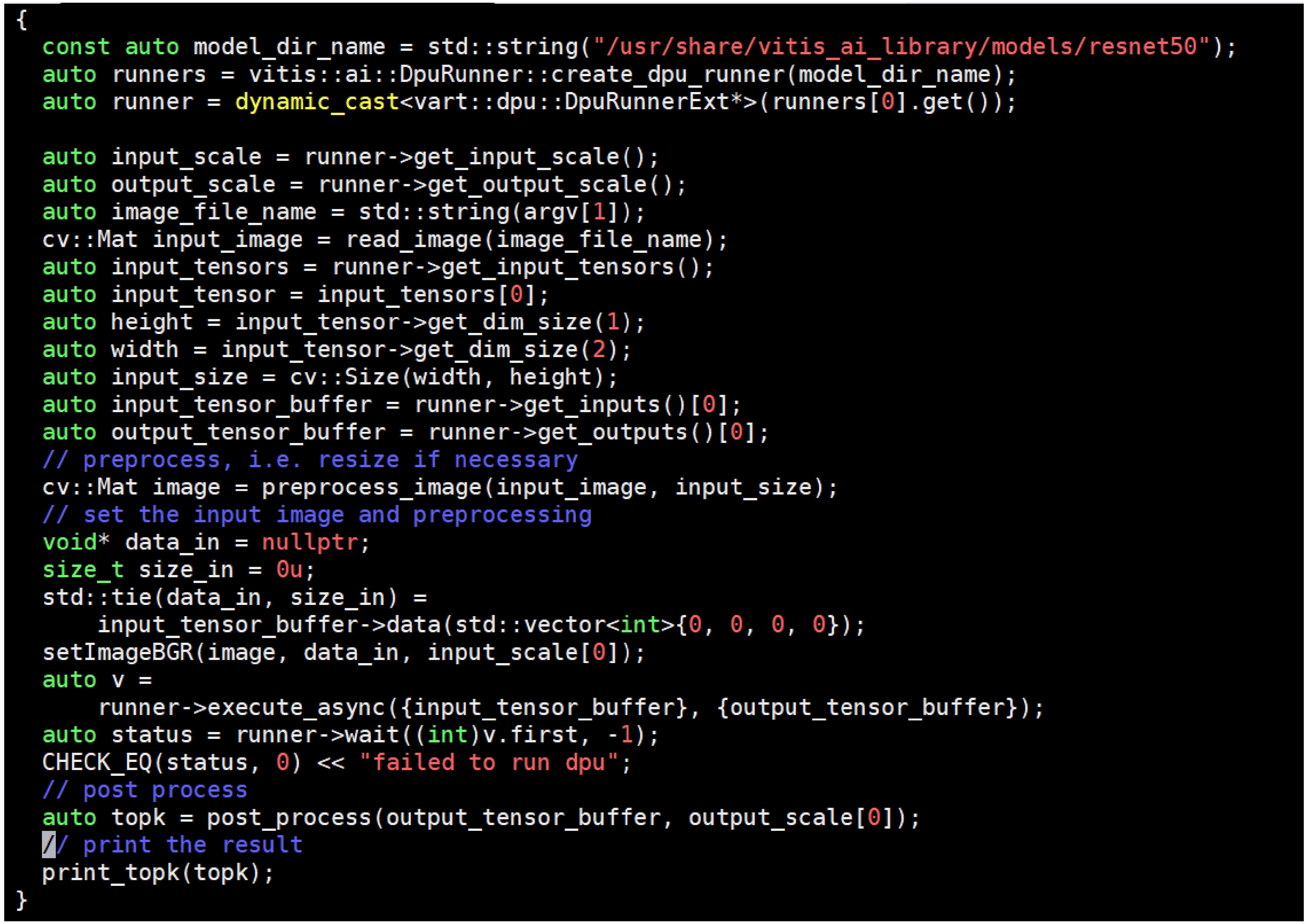
```
Kernel topology "resnet50_kernel_graph.jpg" for network "resnet50"
kernel list info for network "resnet50"
                               Kernel ID : Name
                                       0 : resnet50_0
                                       1 : resnet50_1

                             Kernel Name : resnet50_0
--------------------------------------------------------------------------------
                             Kernel Type : DPUKernel
                               Code Size : 0.64MB
                              Param Size : 24.35MB
                           Workload MACs : 6964.51MOPS
                         IO Memory Space : 2.25MB
                              Mean Value : 0, 0, 0,
                      Total Tensor Count : 59
                Boundary Input Tensor(s)   (H*W*C)
                              input:0(0) : 224*224*3

               Boundary Output Tensor(s)   (H*W*C)
         resnet_v1_50_logits_Conv2D:0(0) : 1*1*1000

                        Total Node Count : 58
                           Input Node(s)   (H*W*C)
            resnet_v1_50_conv1_Conv2D(0) : 224*224*3

                          Output Node(s)   (H*W*C)
           resnet_v1_50_logits_Conv2D(0) : 1*1*1000
```

AI Library：Unified runtime APIs



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

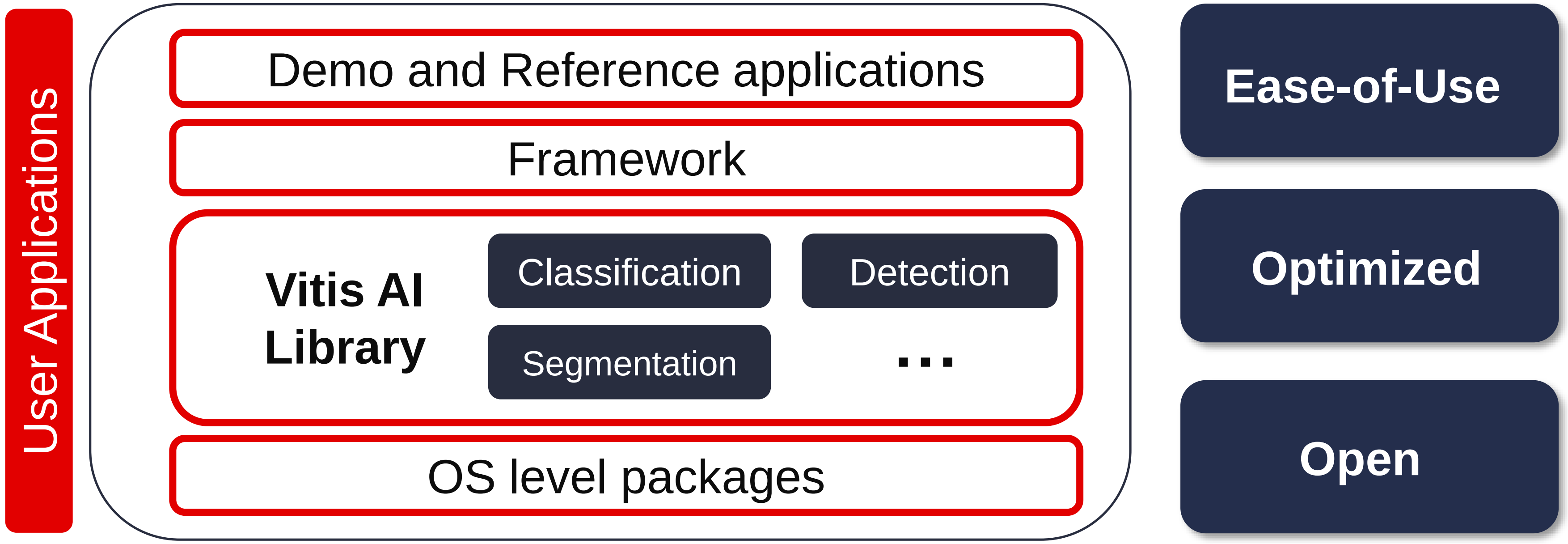
Deploy Resnet50 Using Vitis AI runtime APIs



```
{
  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);
}
```

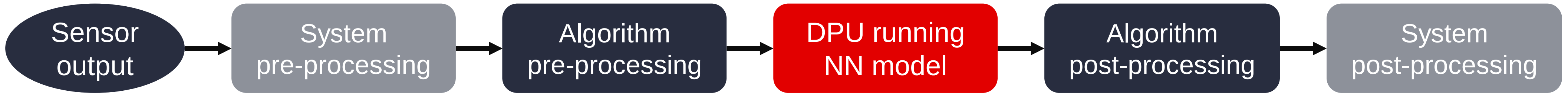
Vitis AI Library: the What?

- Vitis AI Library provides 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



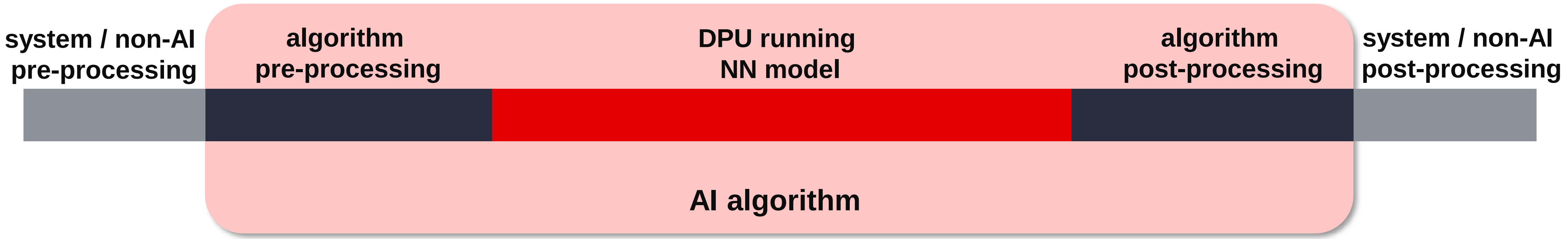
AI Application General Processing Flow

- **A typical abstraction of processing flow:**



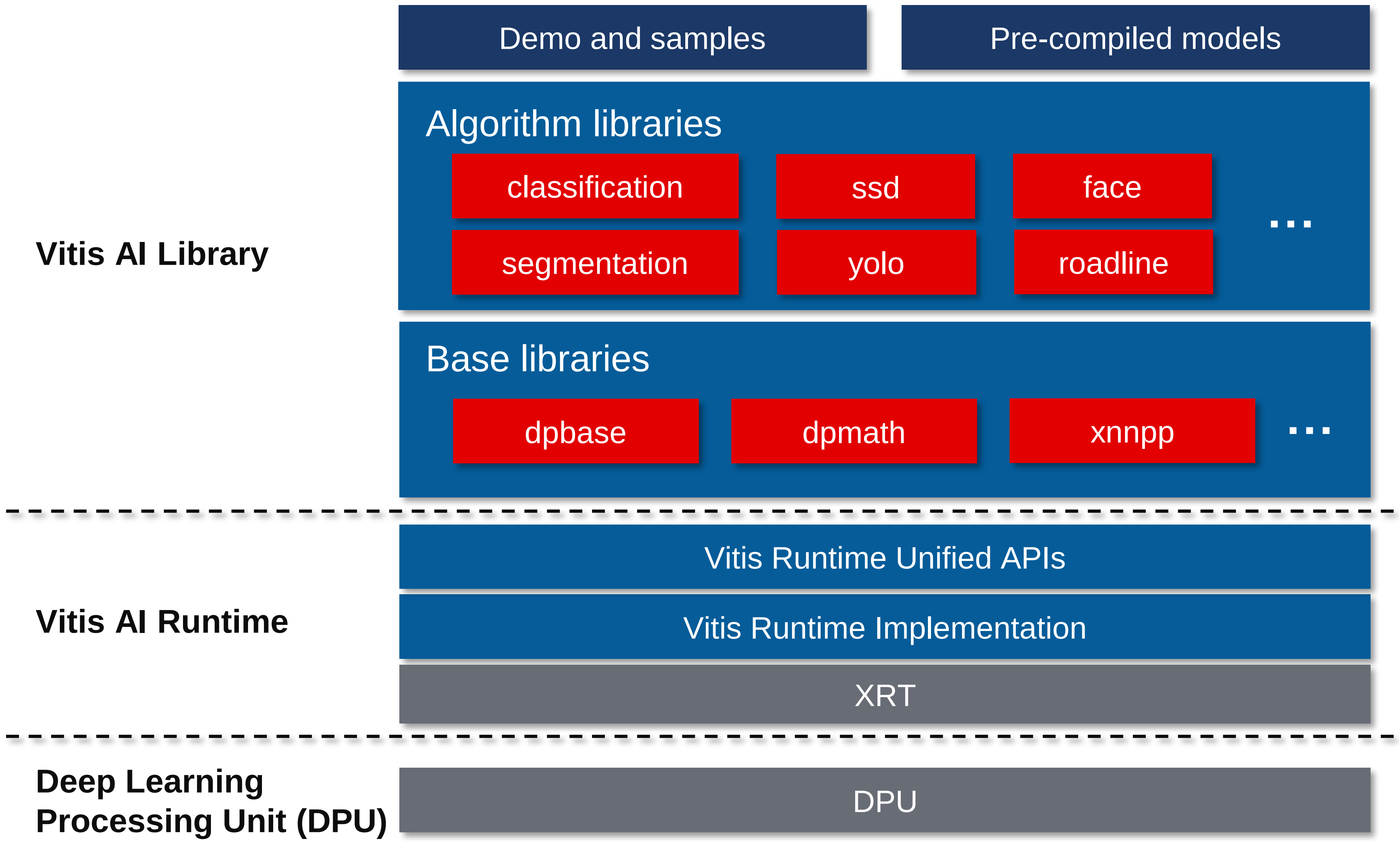
- **Algorithm-level processing**
  - » Data normalization before sending to DPU
  - » Post processing (e.g. bounding boxes decoding in detection)
- **Additional system-level workloads for AI inference**
  - » Color conversion / resizing
  - » Path planning / control / status update

What Vitis AI Library Provides


system / non-AI pre-processing
algorithm pre-processing
DPU running NN model
algorithm post-processing
system / non-AI post-processing
AI algorithm

- **AI Library offers libraries for**
  - Algorithm-level optimization
  - Open and easy to extend
  - Directly support models in AI Model Zoo

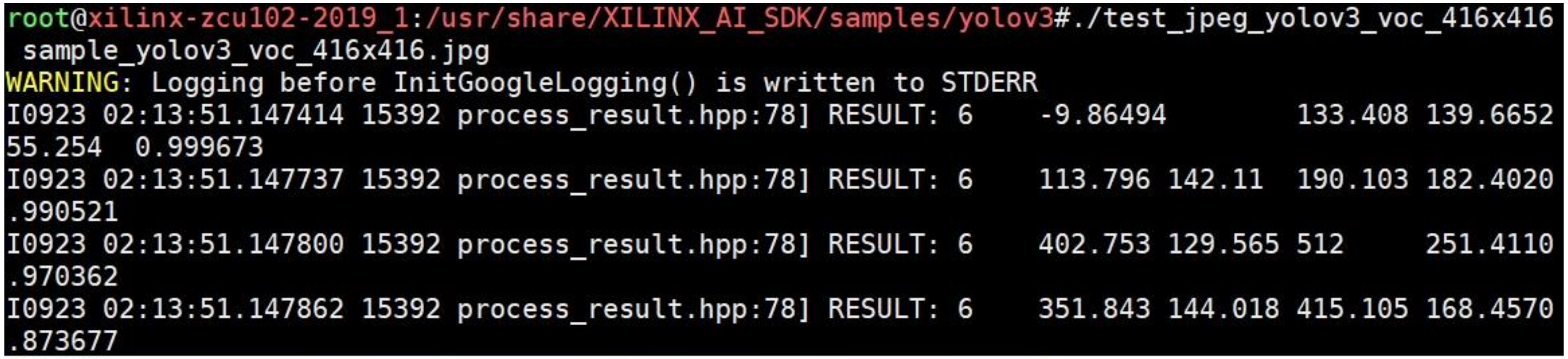
AI Library Deepdive



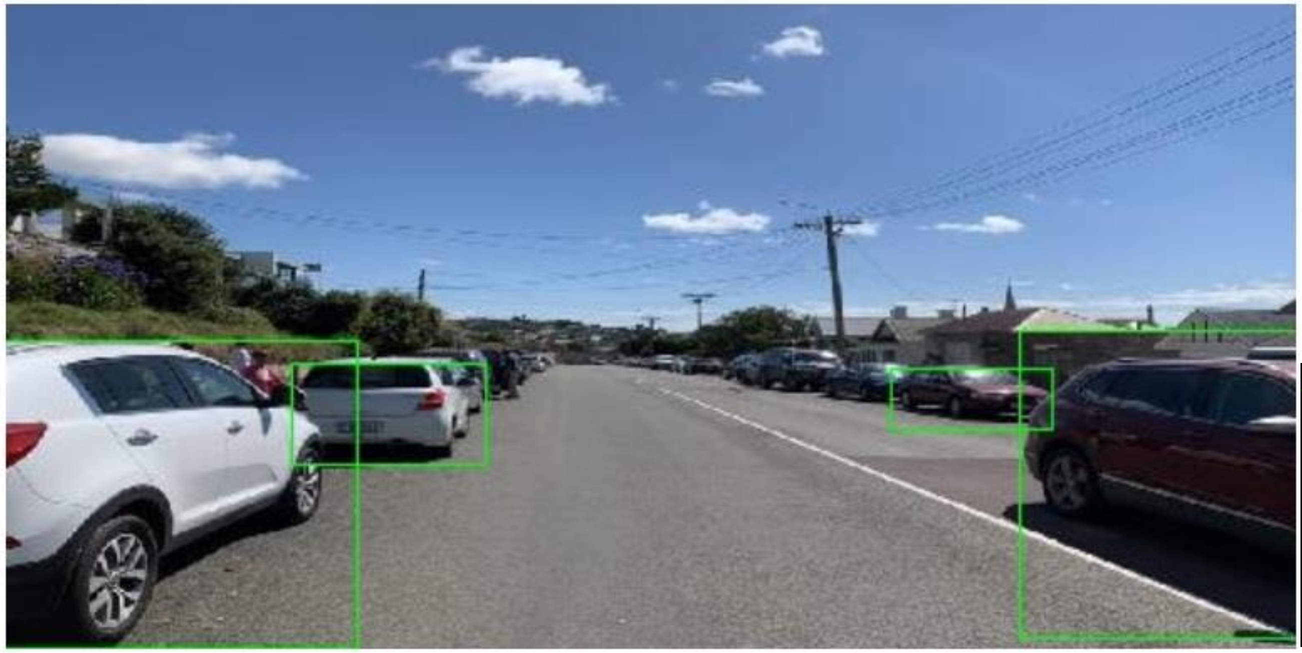
AI Library Samples

- The Vitis AI Library provides image test samples ,video test samples, performance test samples for all the above networks. Each sample has the following four kinds of test sample.
  - test_jpeg_[model type]
  - test_video_[model type]
  - test_performance_[model type]
  - test_accuracy_[model type]
- In addition, the kit provides the corresponding performance test program. For video based testing, we recommend to use raw video for evaluation. Because decoding by software libraries on Arm® CPU may have inconsistent decoding time, which may affect the accuracy of evaluation.

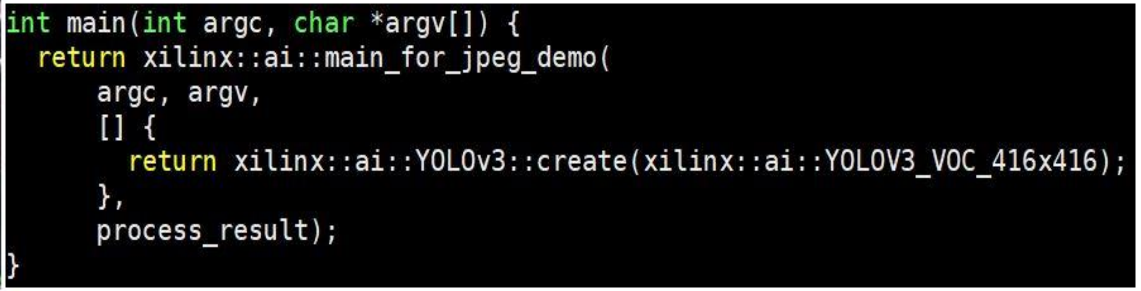
AI Library Samples: test_jpeg_yolov3



```
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);
}
```

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**

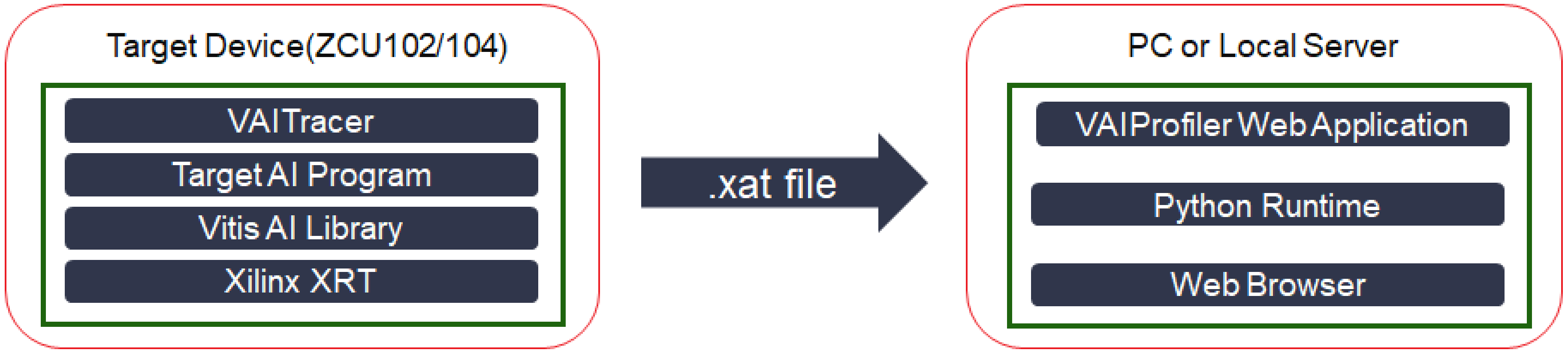
- No need to consider algorithm-level processing and DPU running codes

4 **Support multiple deploying approaches**

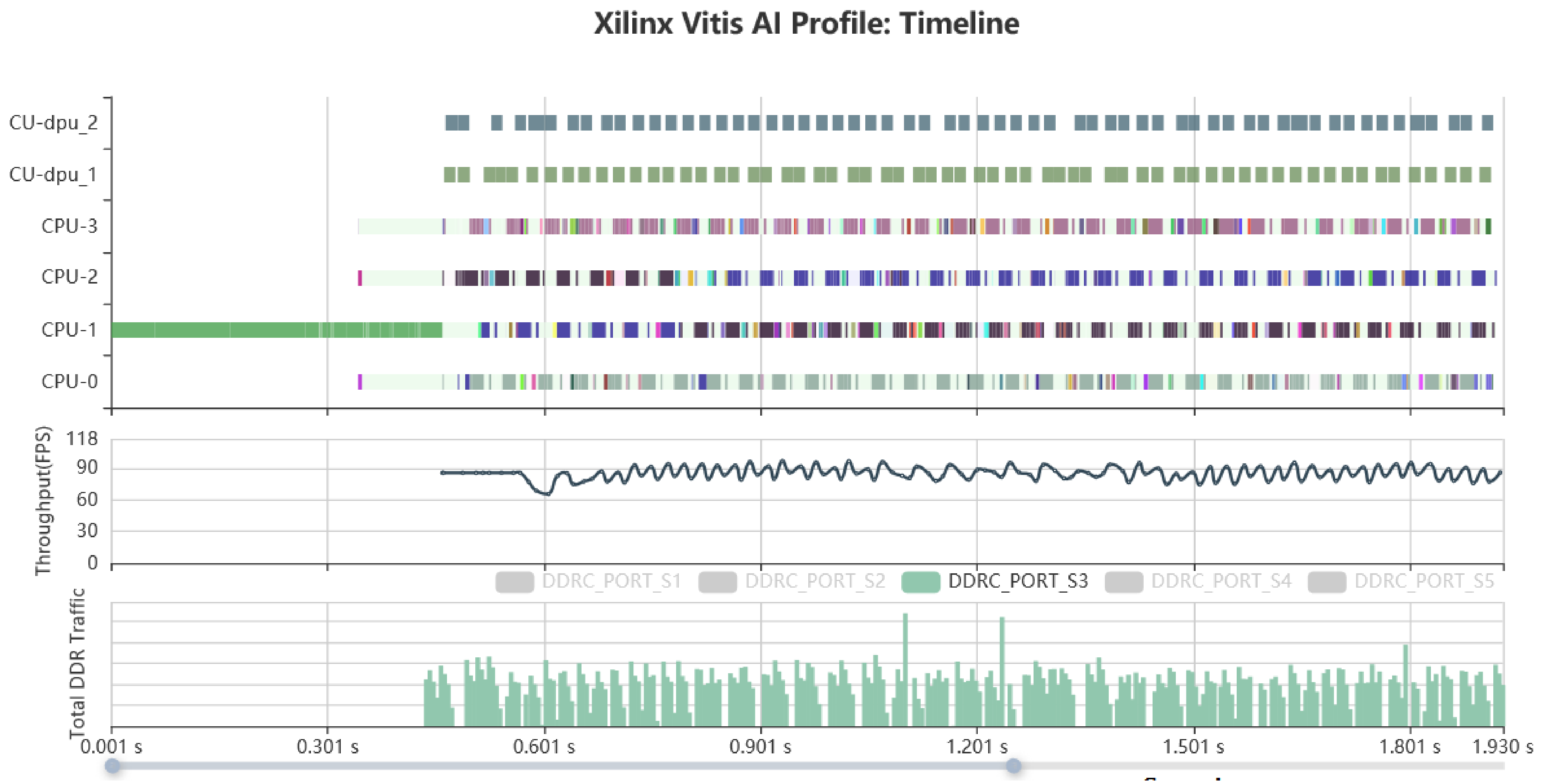
- Besides suggested high-level APIs, DPU running can be also controlled by users

AI Profiler

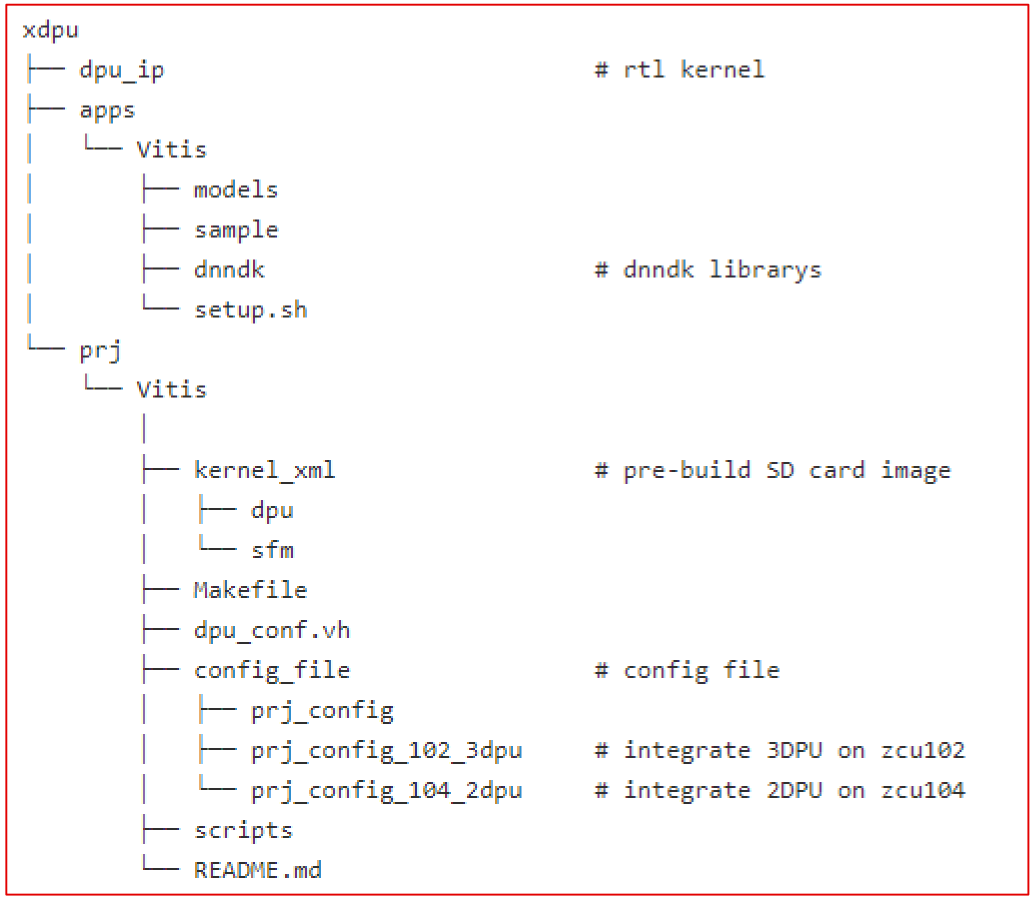
- The Vitis AI profiler tools is a set of tools that helps profile and visualize AI applications based on the Vitis AI Library:
  - Easy to use; requires neither the change in user’s code nor re-compilation of the program
  - Figuring out hot spots or bottlenecks of preference at a glance
  - Illustrating the running state of difference computing units



AI Profiler


Xilinx Vitis AI Profile: Timeline
CU-dpu_2
CU-dpu_1
CPU-3
CPU-2
CPU-1
CPU-0
Throughput(FPS)
118
90
60
30
0
DDRC_PORT_S1
DDRC_PORT_S2
DDRC_PORT_S3
DDRC_PORT_S4
DDRC_PORT_S5
Total DDR Traffic
0.001 s
0.301 s
0.601 s
0.901 s
1.201 s
1.501 s
1.801 s
1.930 s

DPU Target Reference Design



```
xdpu
├── dpu_ip                              # rtl kernel
├── apps
│   └── Vitis
│       ├── models
│       ├── sample
│       ├── dnndk                       # dnndk librarys
│       └── setup.sh
└── prj
    └── Vitis
        │
        ├── kernel_xml                  # pre-build SD card image
        │   ├── dpu
        │   └── sfm
        ├── Makefile
        ├── dpu_conf.vh
        ├── config_file                 # config file
        │   ├── prj_config
        │   ├── prj_config_102_3dpu     # integrate 3DPU on zcu102
        │   └── prj_config_104_2dpu     # integrate 2DPU on zcu104
        ├── scripts
        └── README.md
```

> **DPU IP**

> **Model and related libraries**

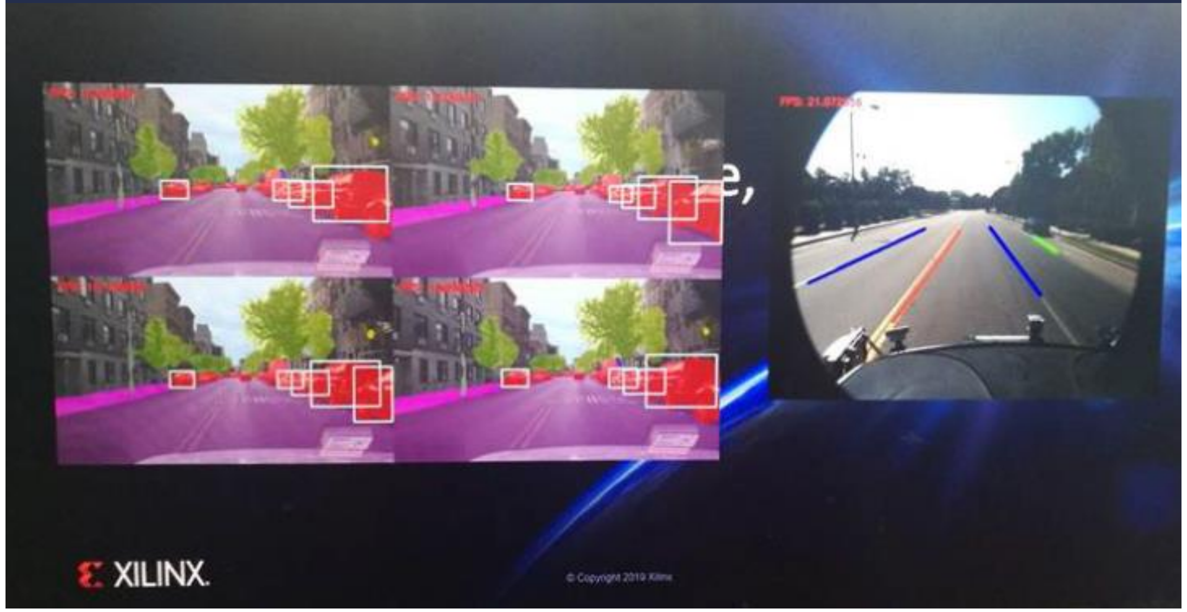
> **Prebuild image**

> **Config files**

Demo Zoo

- ✓ Completely build from AI model zoo and Vitis AI Library
- ✓ Compatible with the latest DPU overlays and Vitis Tools
- ✓ Designed for multiple scenario and applications

ADAS demo



Multi-task and pose detection demo

Vitis AI v1.1 Available NOW!

https://github.com/Xilinx/Vitis-AI

XILINX®

Thank You

Xilinx Core Values

- Excellence
  - Question, learn, and innovate for exceptional results
- Teamwork
  - Work together in the best interest of Xilinx
  - Embrace diversity of thought and experience
  - Collaborate effectively and respectfully
- Accountability
  - Own commitments to their full conclusion
  - Deal with the unexpected quickly and professionally
  - Be transparent about issues, see them as opportunities, and learn from them

XILINX.