Adaptable AI Inference with Vitis AI

Andy Luo

AI Product Maketing

XILINX.

AI Will Transform All the Applications

Power efficient inference along with traditional software

Data Center

5G

Autonomous Driving

Security

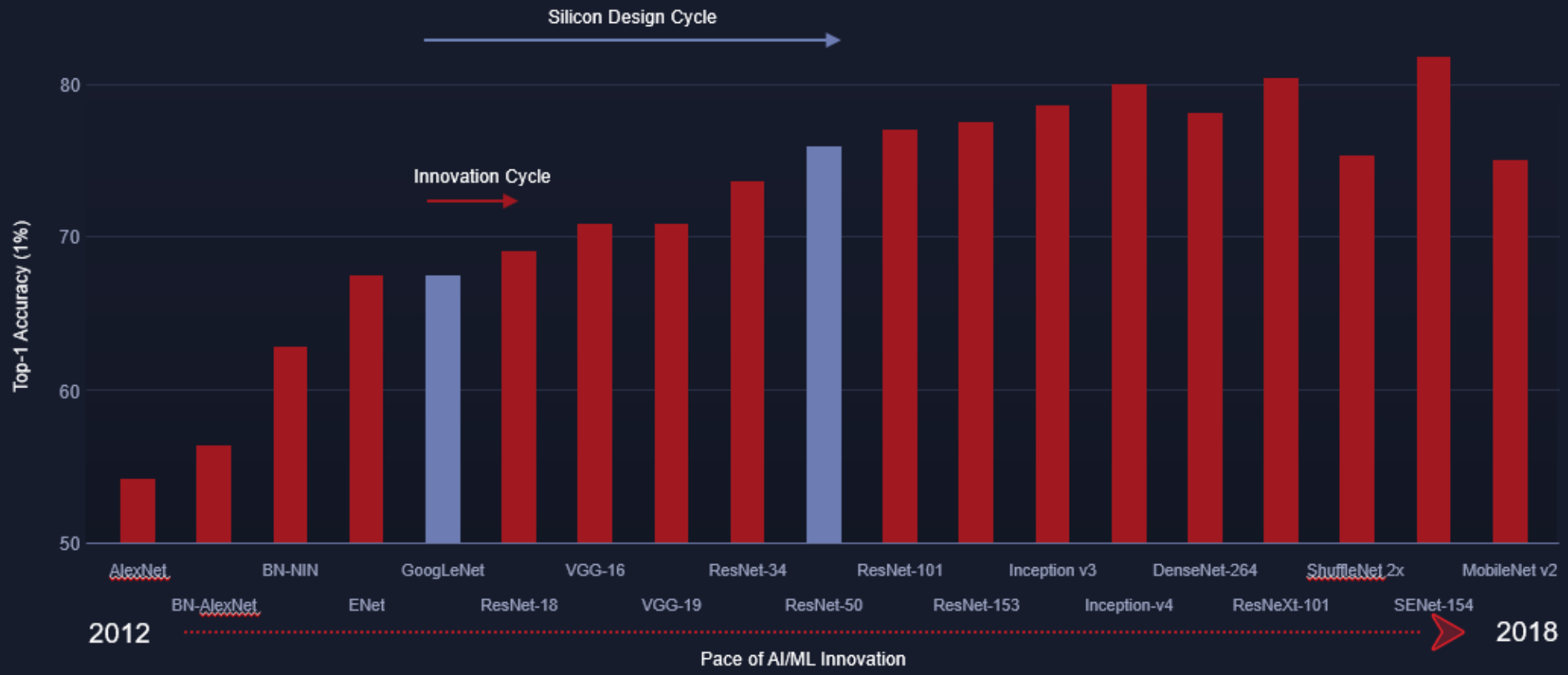
Genomics

Video Analytics

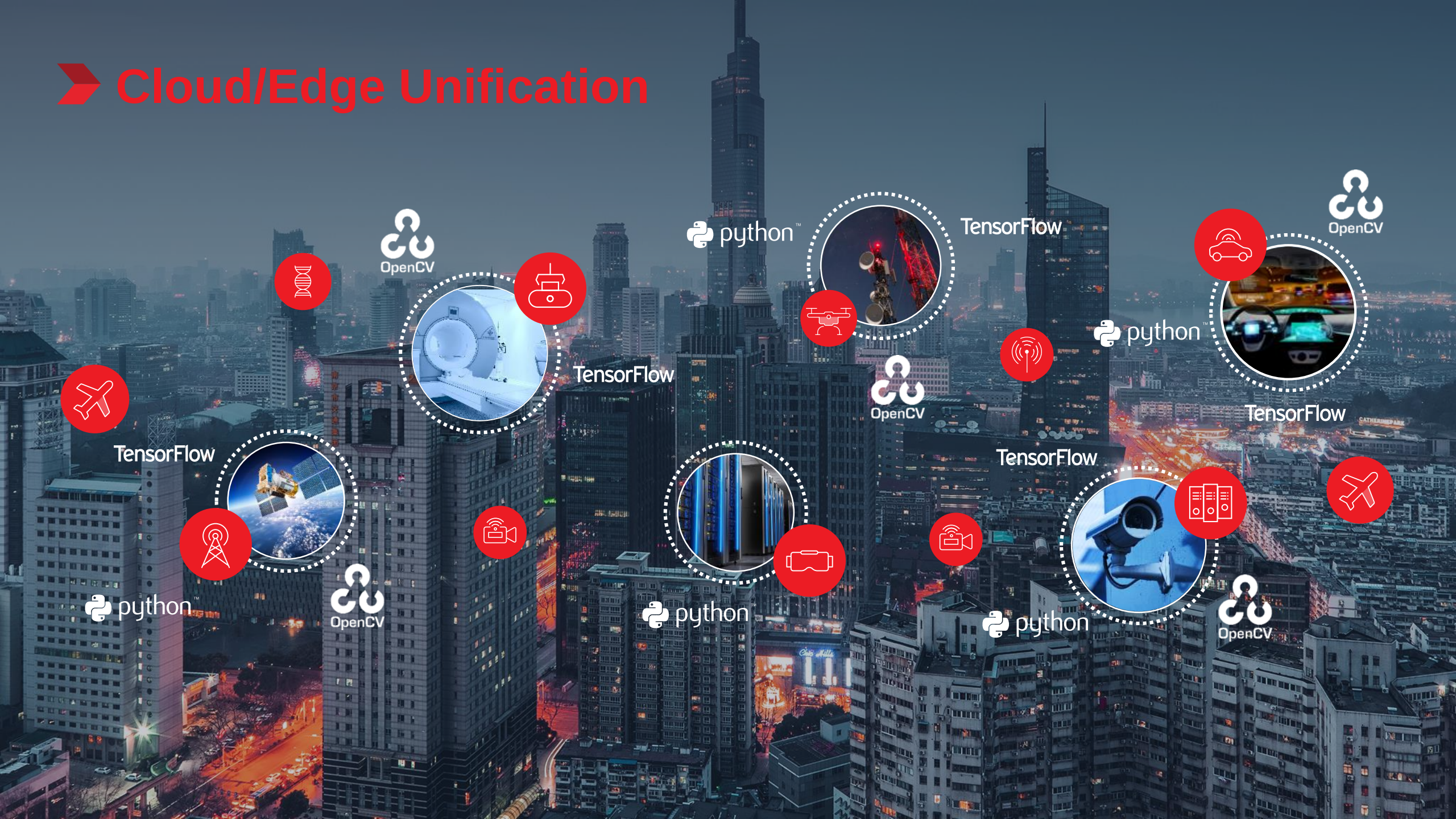
Healthcare

Finance

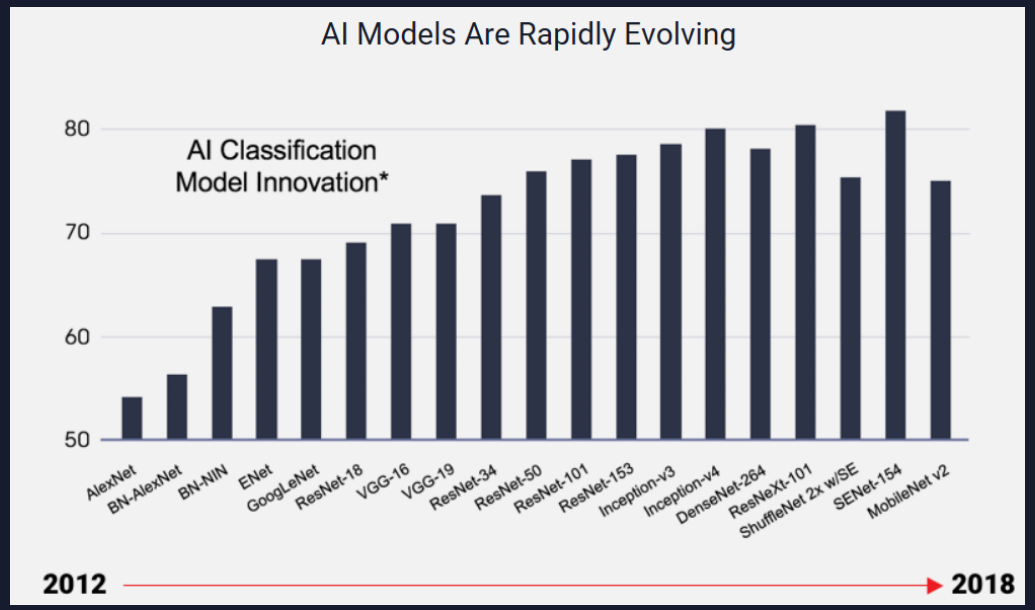
AI is Evolving Rapidly

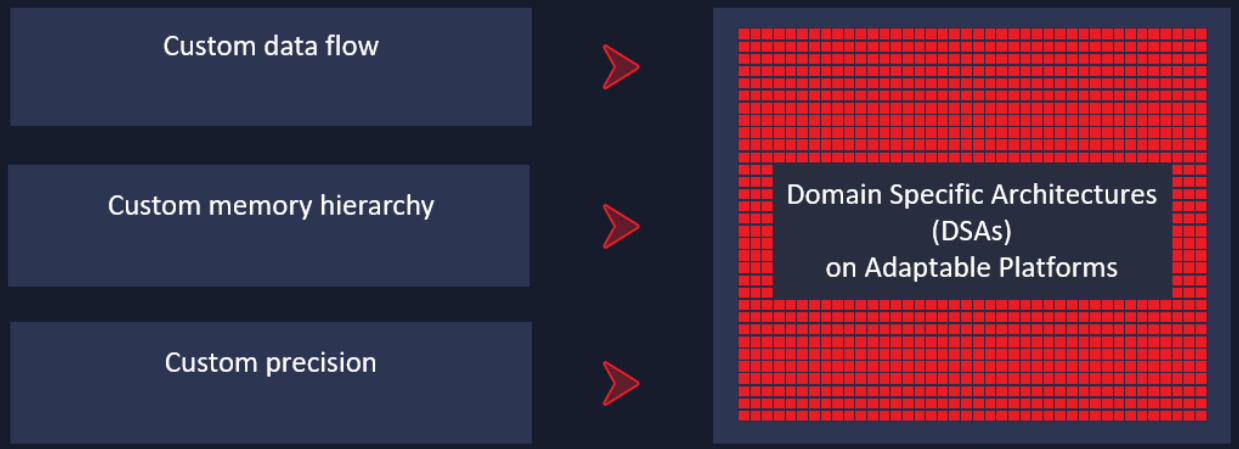


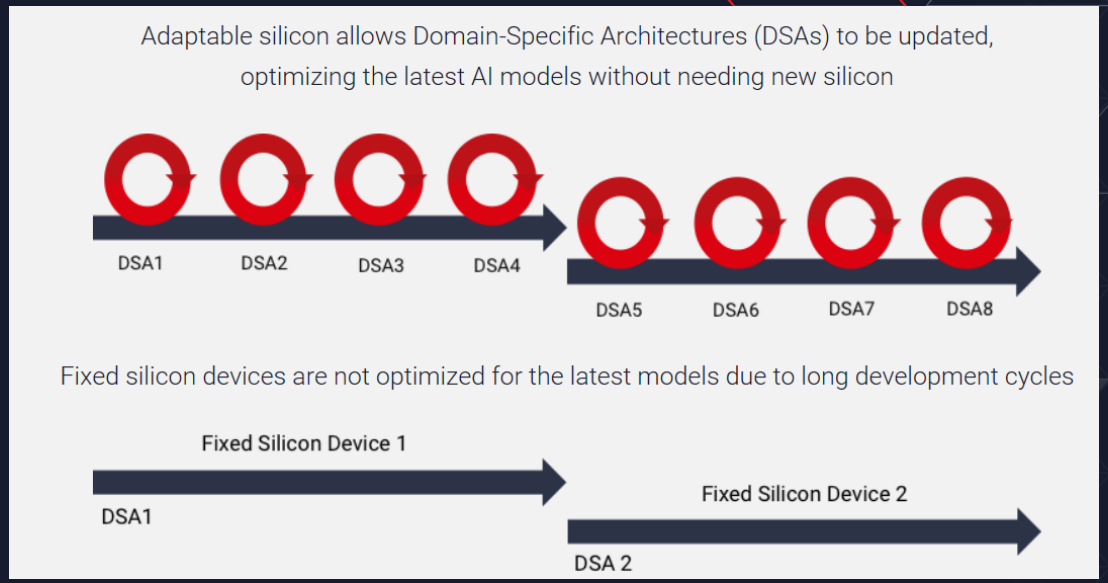
Cloud/Edge Unification



Adapt to Your AI Workloads

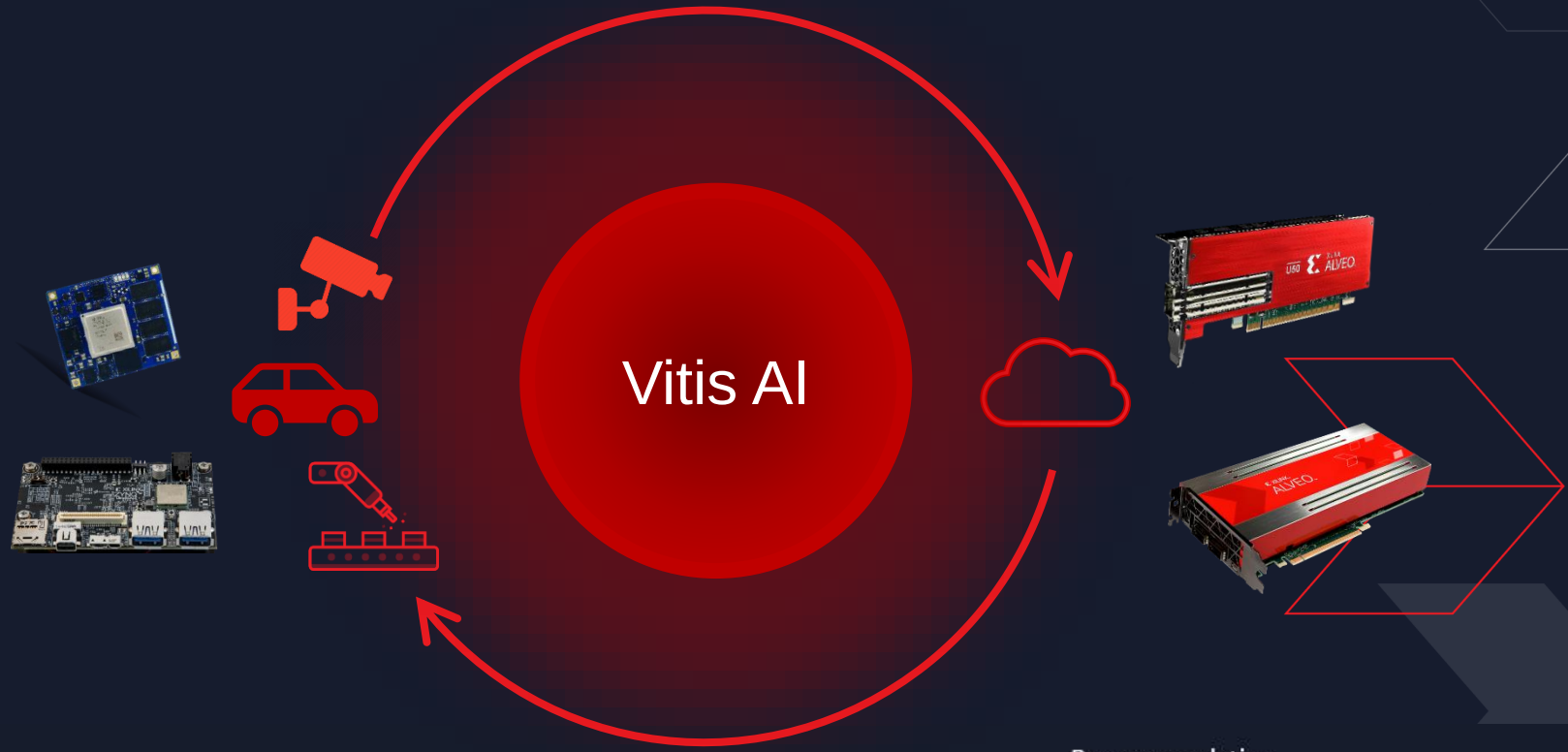






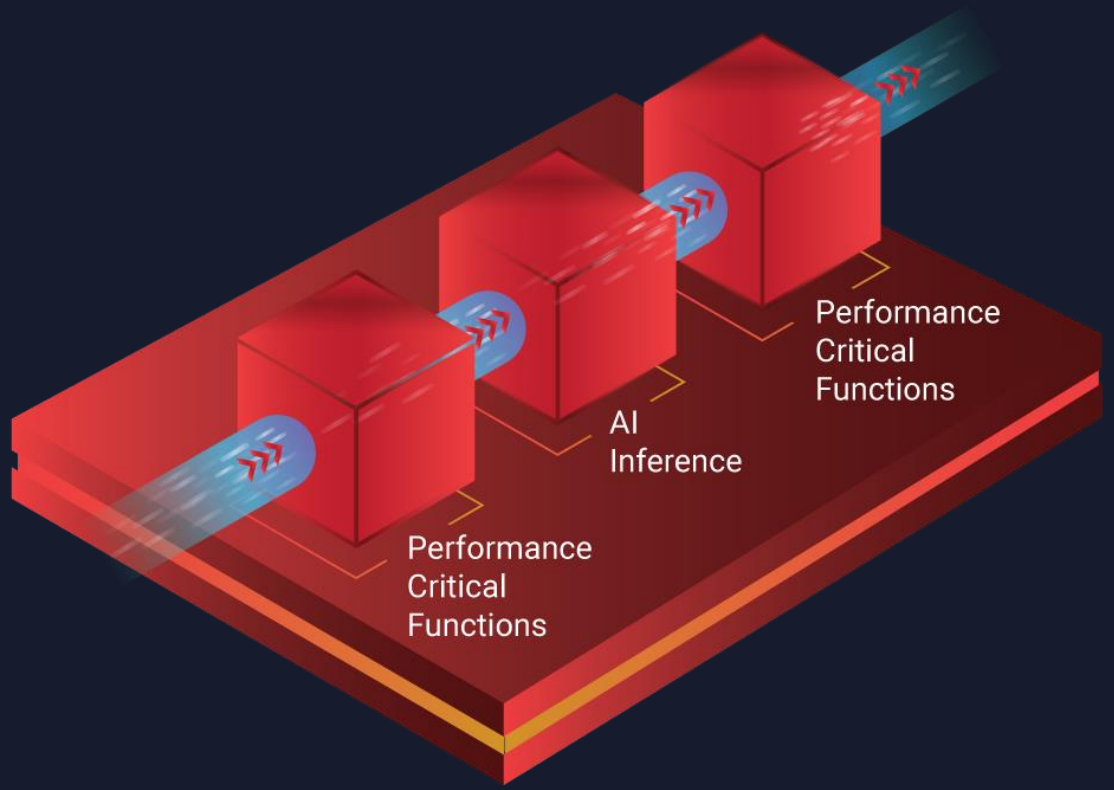
XILINX.

Unified Methodology Edge to Cloud

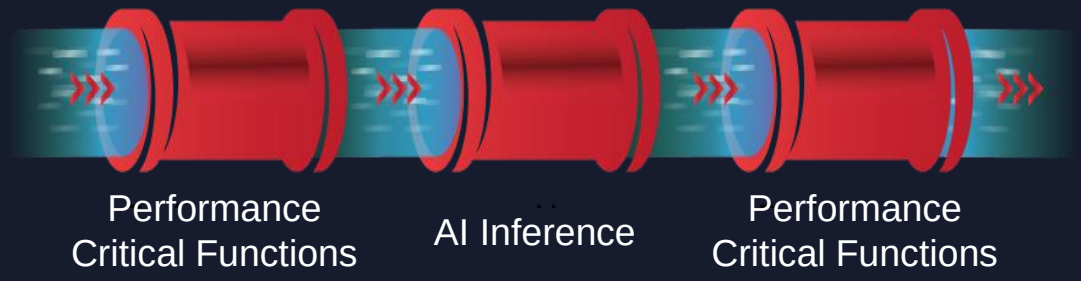


XILINX.

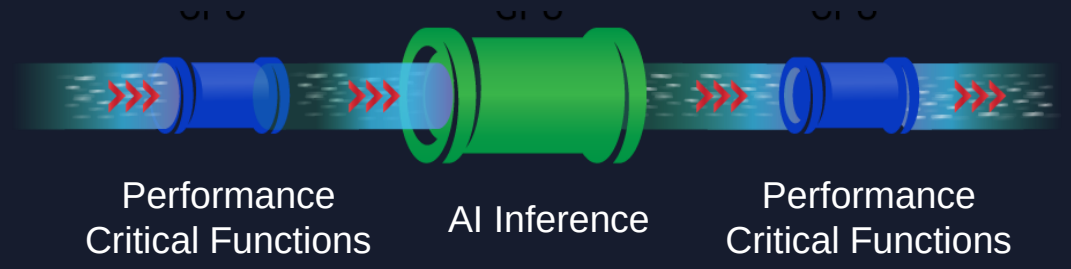
Whole Application Acceleration



Xilinx – Matched Throughput

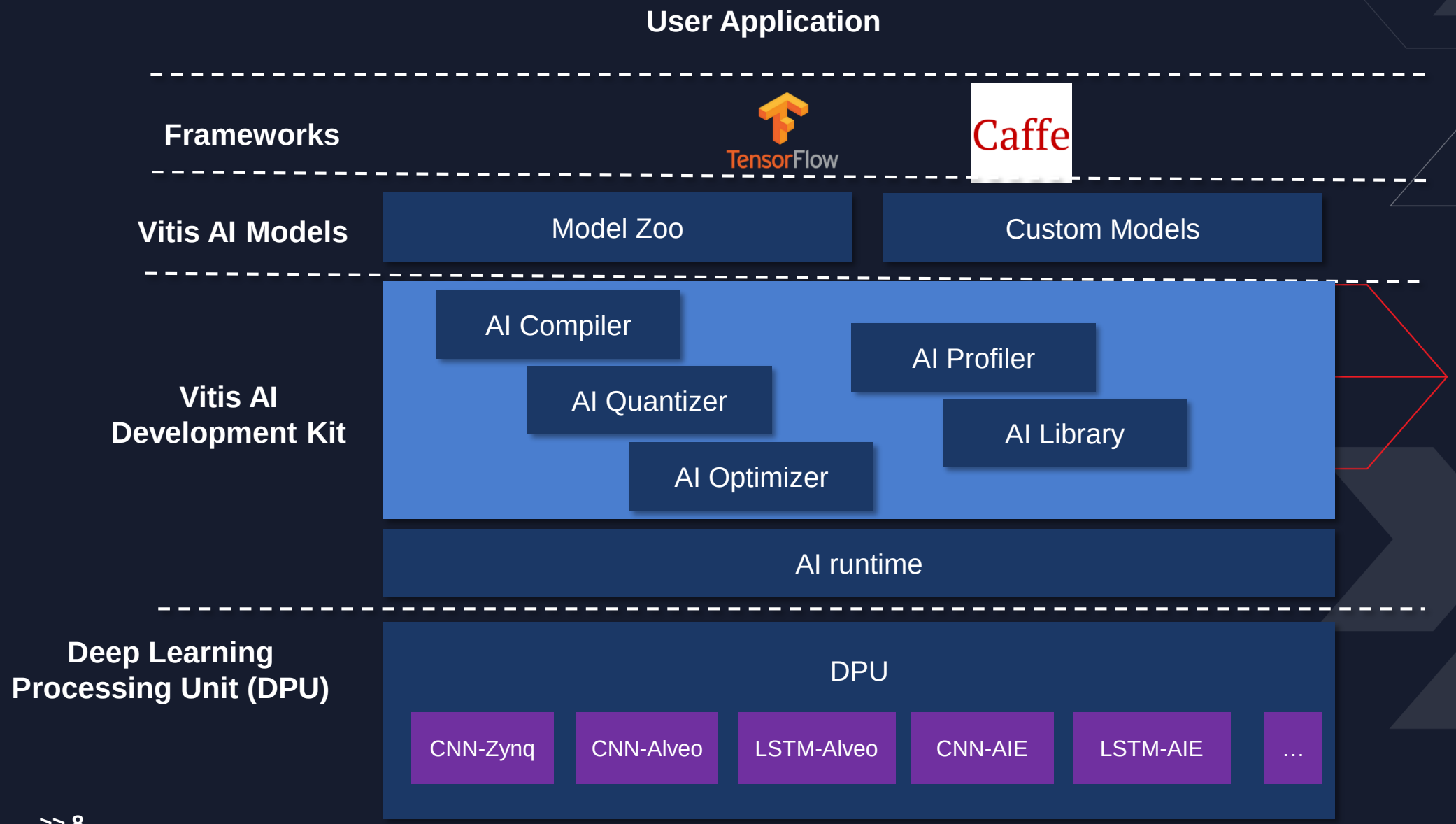


GPU & CPU – Mismatched Throughput



XILINX

What is Vitis AI?



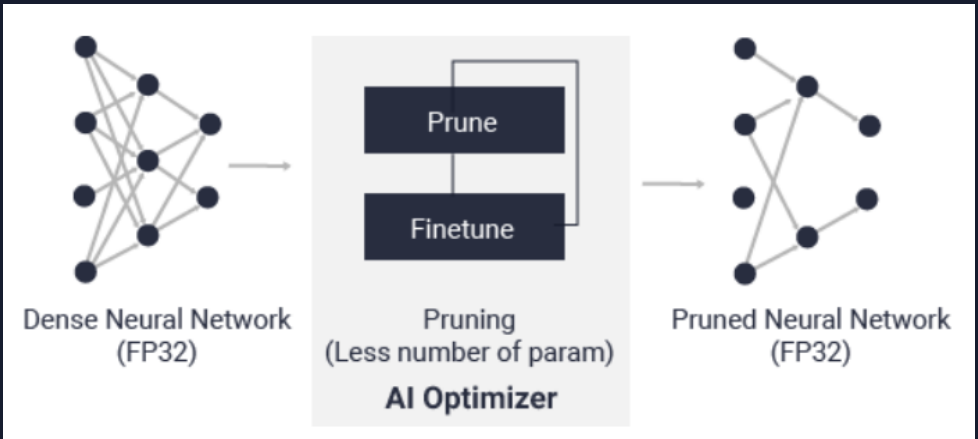
XILINX.

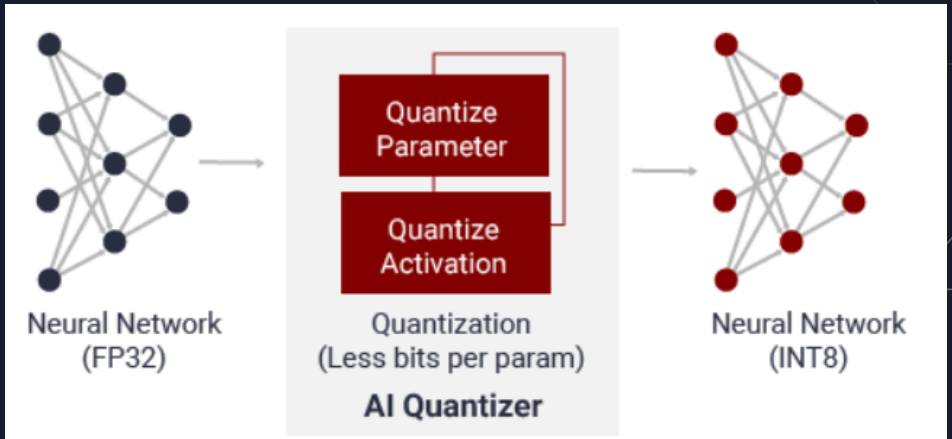
Vitis AI Model Zoo 1.0

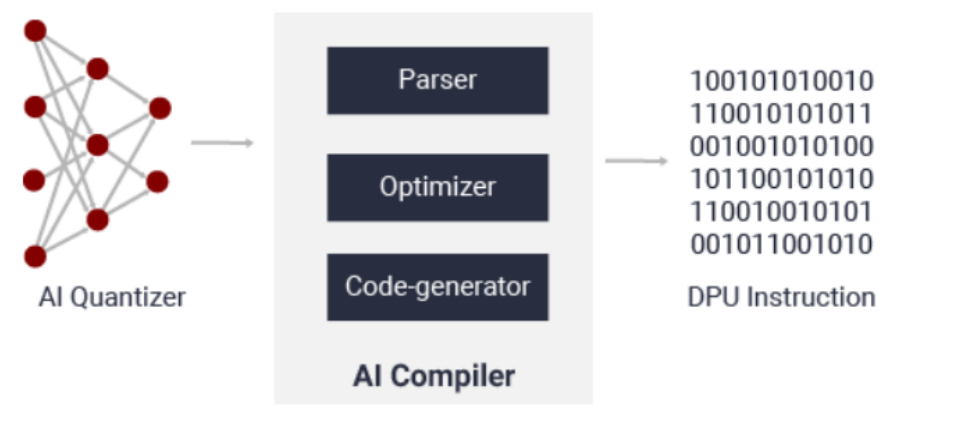
- **Shared Repository of Pre-Trained AI Models**
  - Ready to Deploy, Pre-Optimized Models
  - 50+ Models Support Broad Range of Applications
  - Open and Available on GitHub
- **Leverage Standard Frameworks, Networks, Datasets**
  - Trained Using TensorFlow and Caffe
- **Deploy As-is, Re-Train or Further Optimize**
  - Caffe_Xilinx, a custom distribution of Caffe provided to test & finetune caffe models
  - Training code, test code and train eval instructions provided

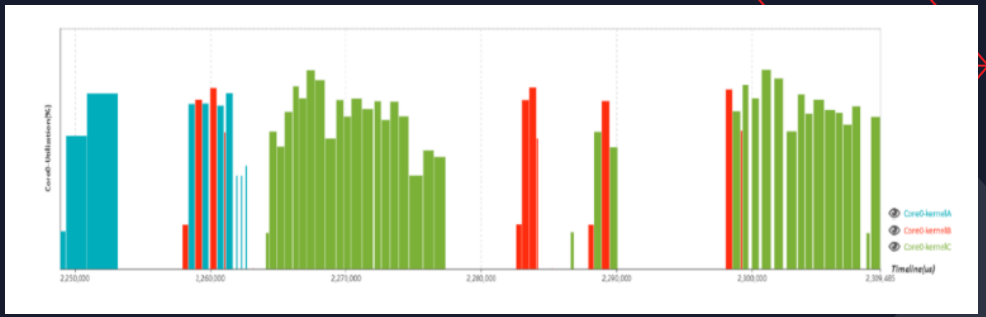
Application	Model
Face	Face detection
	Landmark Localization
	Face recognition
	Face attributes recognition
Pedestrian	Pedestrian Detection
	Pose Estimation
	Person Re-identification
Video Analytics	Object detection
	Pedestrian Attributes Recognition
	Car Attributes Recognition
	Car Logo Detection
	Car Logo Recognition
	License Plate Detection
	License Plate Recognition
ADAS/AD	Object Detection
	3D Car Detection
	Lane Detection
	Traffic Sign Detection
	Semantic Segmentation
	Drivable Space Detection

Vitis AI Development Kit

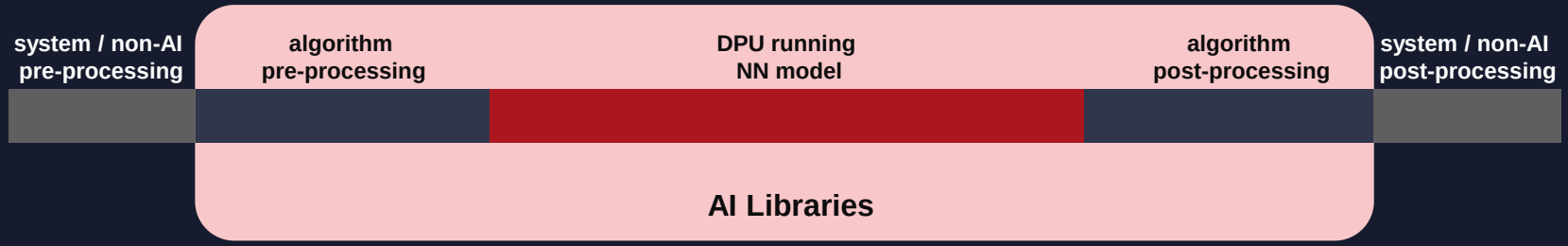






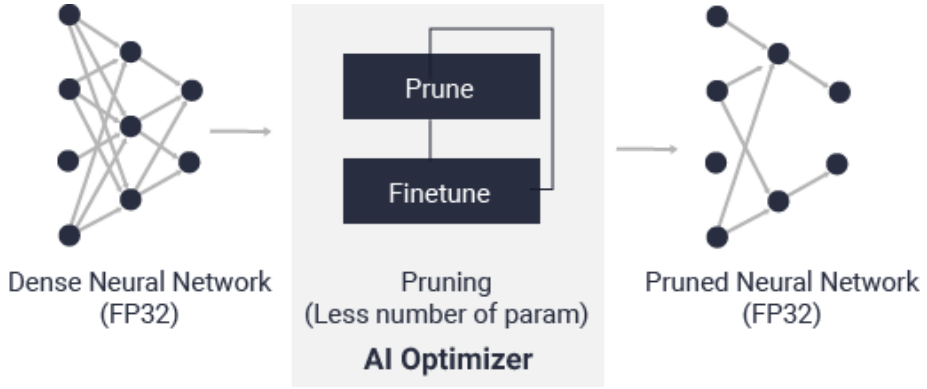


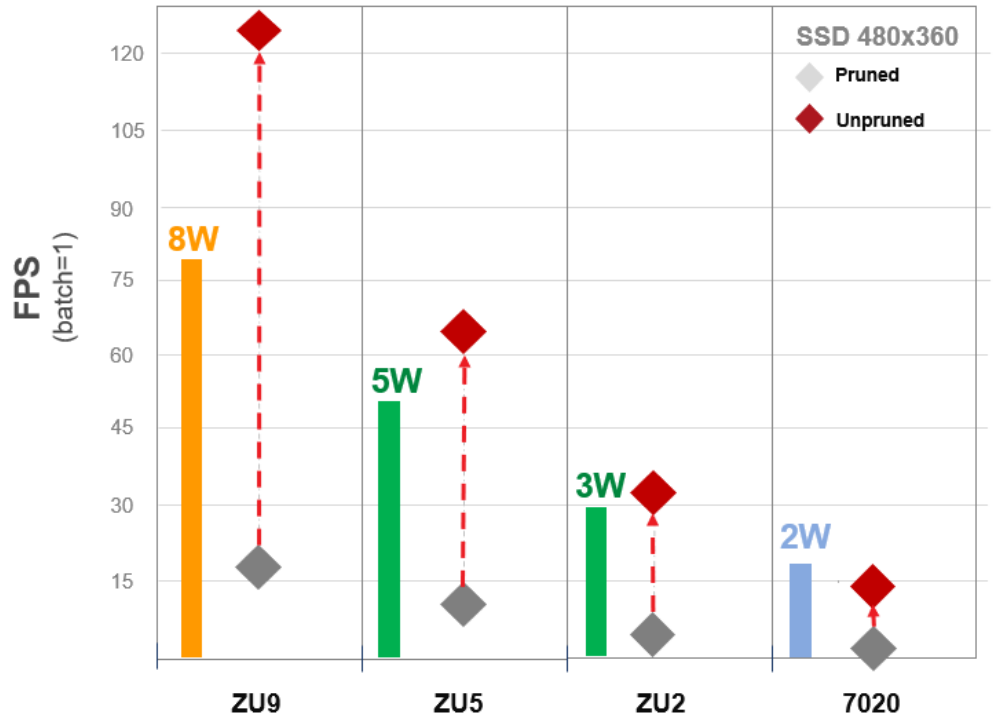
AI Profiler



XILINX.

Vitis AI Optimizer





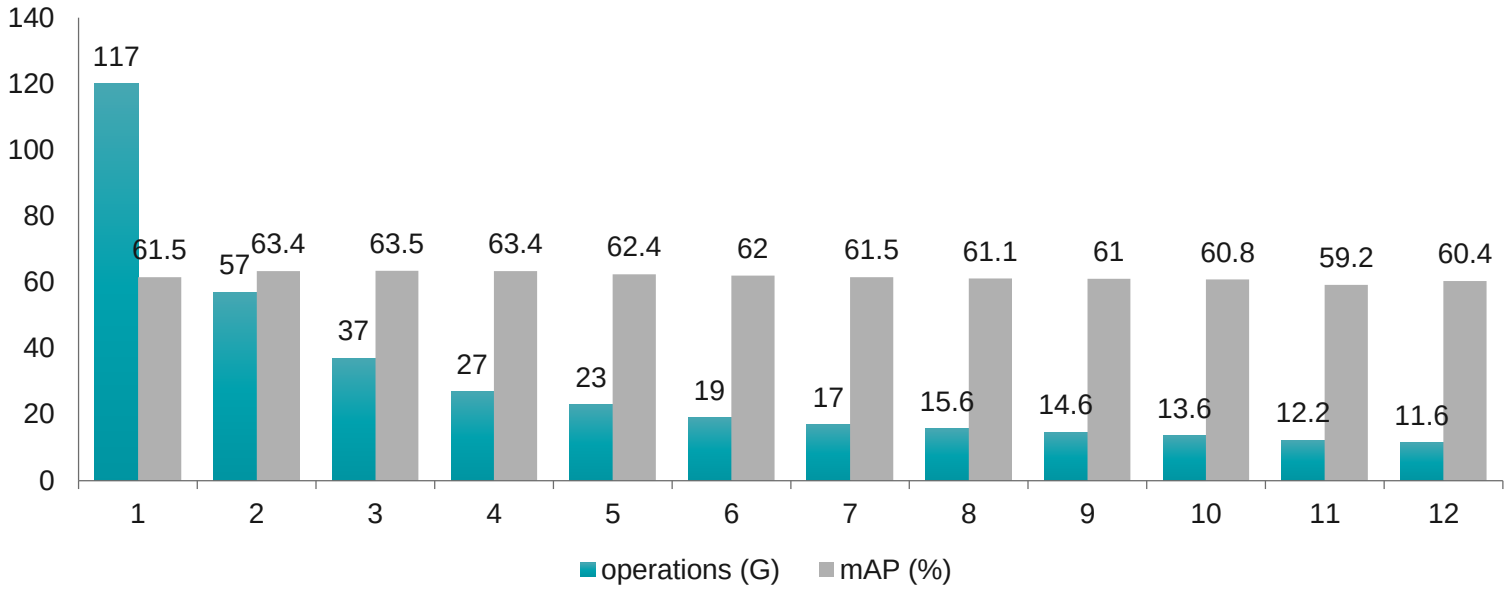
- **World's leading model compression**
  - Iterative, coarse-grained pruning
  - Reduce model size 5 – 30x
  - Increase performance 2 – 10x
  - Minimal accuracy loss, <1%
- **Supported framework**
  - Caffe, Darknet, TensorFlow
- **Commercial license available**
  - Contact Xilinx sales representative

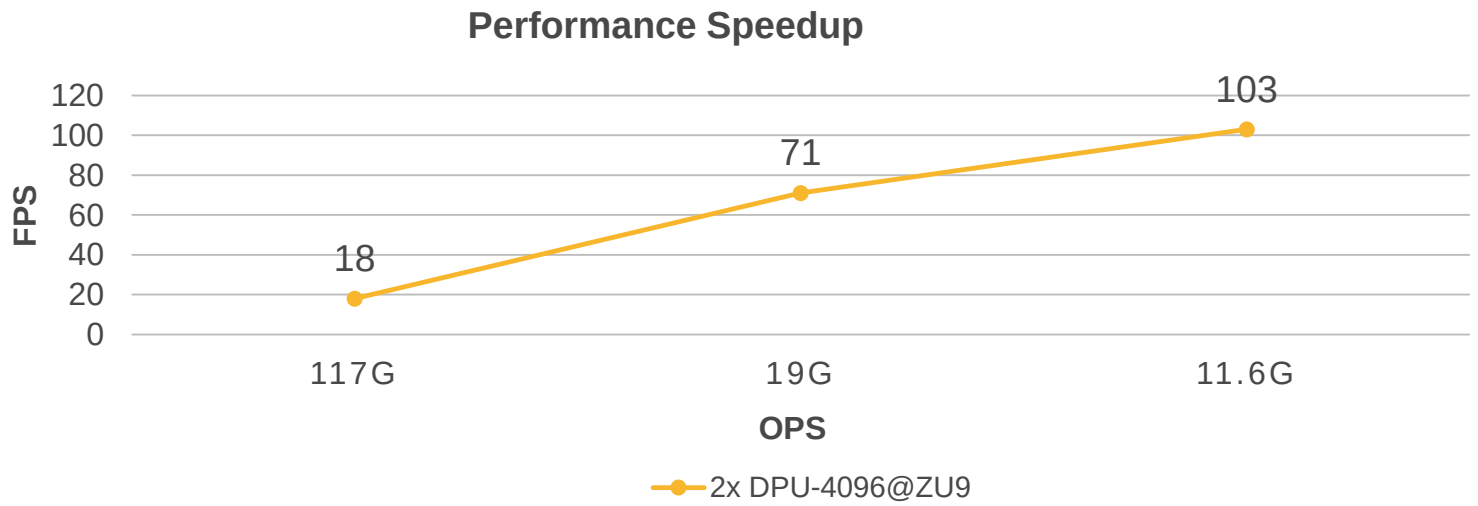
XILINX

AI Optimizer Usage

- **Sensitivity analysis, searching for best pruning strategy**
- **Prune the network iteratively until the network's sparsity meets your needs**
- **Using the pruning tool**
  - Prune the network using a certain pruning ratio
  - Fine-tune the pruned network
  - Increase pruning ratio
  - Prune again
  - Loop the pruning-retraining iteration for a few times
  - Generate a final dense model

Example Using Vitis AI Optimizer

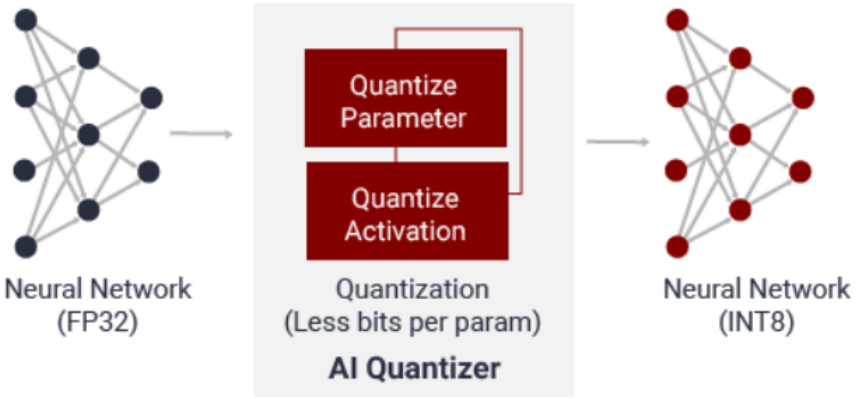


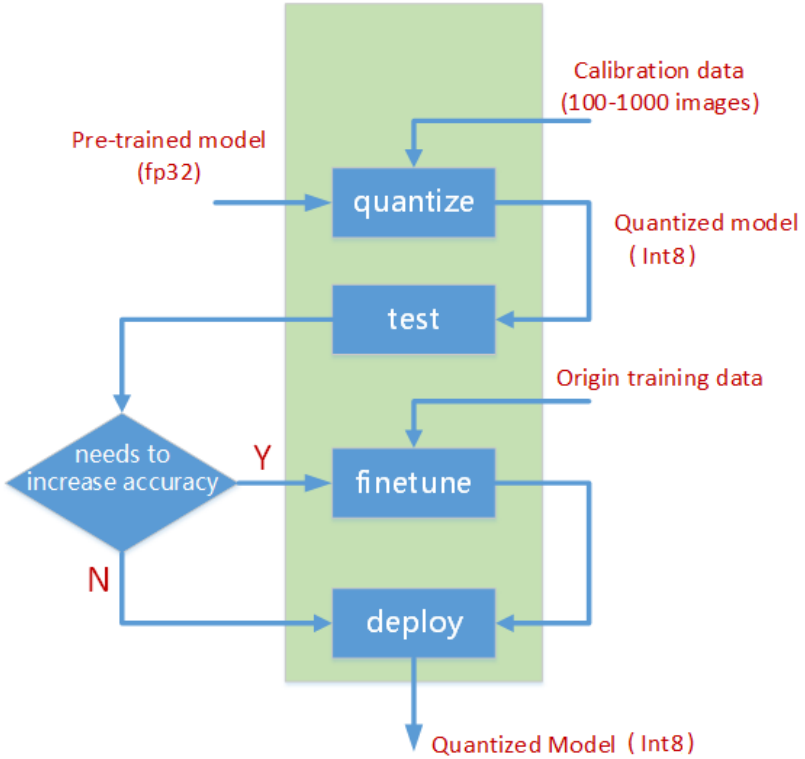


SSD+VGG @ Surveillance 4 Classes

XILINX.

Vitis AI Quantizer





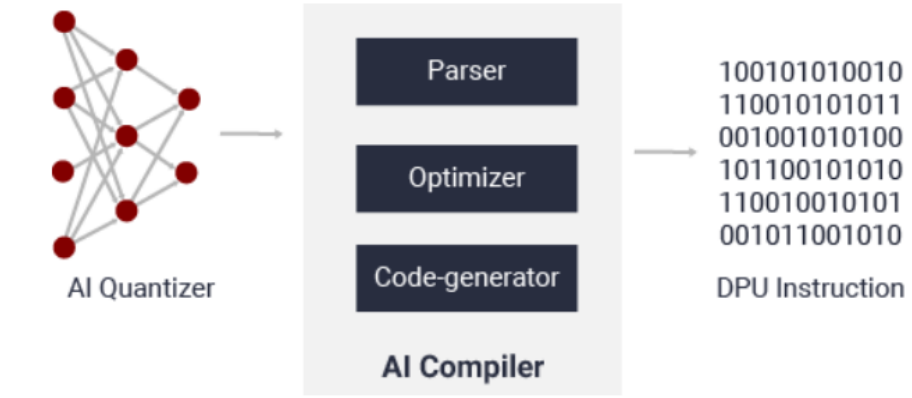
- **Uniform symmetric quantization**
  - 8bit for both weights and activation
- **Support both calibration and finetune**
  - Calibration – A small set of training data
  - Finetune – Original training data, further increase accuracy
- **Support framework**
  - Caffe, Tensorflow
  - Pytorch (Q2, 2020)
- **Have both GPU and CPU version**
  - GPU version is 10x faster than CPU version

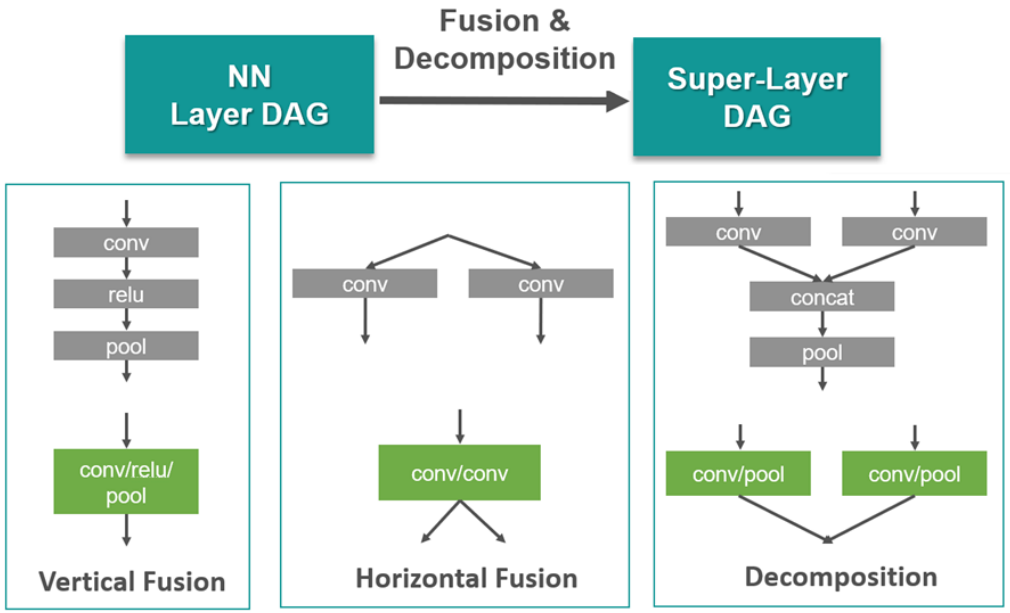
Quantization Result

Classification Networks	Float		8 bit quantized				After quantized finetune			
	Top1	Top5	Top1	ΔTop1	Top5	ΔTop5	Top1	ΔTop1	Top5	ΔTop5
Inception_v1	66.90%	87.68%	66.54%	-0.36%	87.58%	-0.10%	66.62%	-0.28%	87.58%	-0.10%
Inception_v2	72.78%	91.04%	71.93%	-0.85%	90.58%	-0.46%	72.40%	-0.38%	90.82%	-0.23%
Inception_v3	77.01%	93.29%	76.26%	-0.75%	92.85%	-0.44%	76.56%	-0.45%	93.00%	-0.29%
Inception_v4	79.74%	94.80%	79.04%	-0.70%	94.53%	-0.27%	79.42%	-0.32%	94.64%	-0.16%
ResNet-50	74.76%	92.09%	73.74%	-1.02%	91.44%	-0.65%	74.59%	-0.17%	91.95%	-0.14%
VGG16-3fc-float	70.97%	89.85%	70.67%	-0.30%	89.72%	-0.13%	70.74%	-0.23%	89.79%	-0.06%
MobileNet_v1	70.61%	89.63%	68.01%	-2.60%	88.14%	-1.49%	69.71%	-0.90%	89.06%	-0.57%

Detection Networks	Dataset	Float mAP	8 bit quantized mAP	ΔmAP
SSD_VGG	VOC 21 classes	76.47%	76.27%	-0.20%
SSD_MobileNet_v2	BDD100k 11 classes	30.80%	29.70%	-1.10%
SSDLite_MobileNet_v2	Customer's data	20.28%	20.12%	-0.16%

Vitis AI Compiler

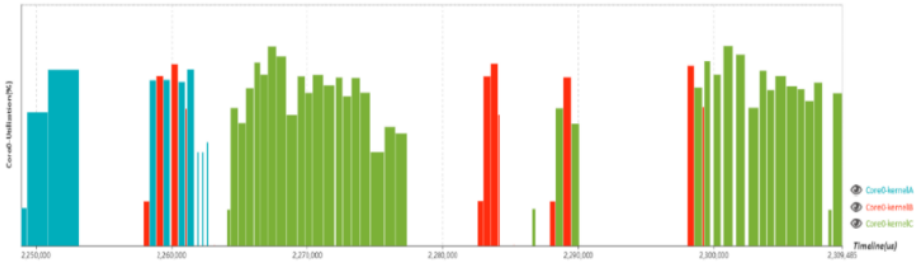




- **Maps the quantized model to instruction set and data flow**
  - High-efficient tensor-level DPU instruction set
- **Performs sophisticated optimizations**
  - Layer fusion and decomposition
  - Instruction scheduling
  - Reuses on-chip memory as much as possible
- **Support framework**
  - Caffe, Tensorflow
  - Pytorch (Q2, 2020)

Vitis AI Profiler

Layer-by-layer Profiling



ID	NodeName	Workload(MOP)	RunTime(ms)	Perf(GOPS)	Utilization	MB/S
0	conv1	236.0	5.28	44.66	19.4%	69.7
1	res2a_branch1	102.8	0.96	107.49	46.7%	1083.9
2	res2a_branch2a	25.7	0.23	111.70	48.6%	1779.4
3	res2a_branch2b	231.2	1.35	171.90	74.7%	328.7
4	res2a_branch2c	102.8	1.63	63.08	27.4%	643.8
5	res2b_branch2a	102.8	0.65	158.83	69.1%	1586.7
6	res2b_branch2b	231.2	1.34	172.03	74.8%	328.9
7	res2b_branch2c	102.8	1.64	62.74	27.3%	640.3
8	res2c_branch2a	102.8	0.65	159.07	69.2%	1589.1
9	res2c_branch2b	231.2	1.35	171.90	74.7%	328.7
...						
47	res5b_branch2a	102.8	1.00	102.35	44.5%	1180.3
48	res5b_branch2b	231.2	1.77	130.33	56.7%	1363.8
49	res5b_branch2c	102.8	1.30	78.80	34.3%	928.5
50	res5c_branch2a	102.8	1.01	101.74	44.2%	1173.3
51	res5c_branch2b	231.2	1.76	131.59	57.2%	1377.0
52	res5c_branch2c	102.8	1.27	80.66	35.1%	949.6
Total Nodes In Avg:	All	7711.9	66.43	116.10	50.5%	668.8

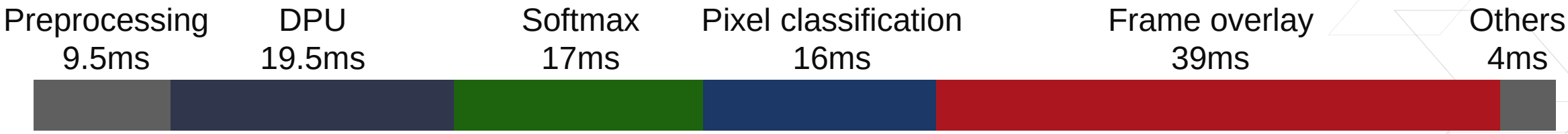
End-to-end Profiling

Task Name	Start Time (us)	End Time (us)	Duration (us)
xilinx::ai::yolov3_post_process	1677531	1677990	458
xilinx::ai::YOLOv3Imp::run	1678003	1804787	126783
xilinx::ai::ConfigurableDpuTaskImp::setInputImageRGB	1678012	1680140	2128
xilinx::ai::DpuTaskImp::setImageRGB_1	1678018	1680139	2120
xilinx::ai::DpuTaskImp::setImageRGB	1678025	1680137	2111
xilinx::ai::NormalizeInputDataRGB	1678036	1680121	2085
xilinx::ai::ConfigurableDpuTaskImp::run	1680145	1804366	124220
xilinx::ai::DpuTaskImp::run	1680183	1804361	124178
dpu	1740888	1804325	63436
xilinx::ai::yolov3_post_process	1804380	1804782	401
xilinx::ai::YOLOv3Imp::run	1804804	1931732	126927
xilinx::ai::ConfigurableDpuTaskImp::setInputImageRGB	1804817	1807077	2260

XILINX.

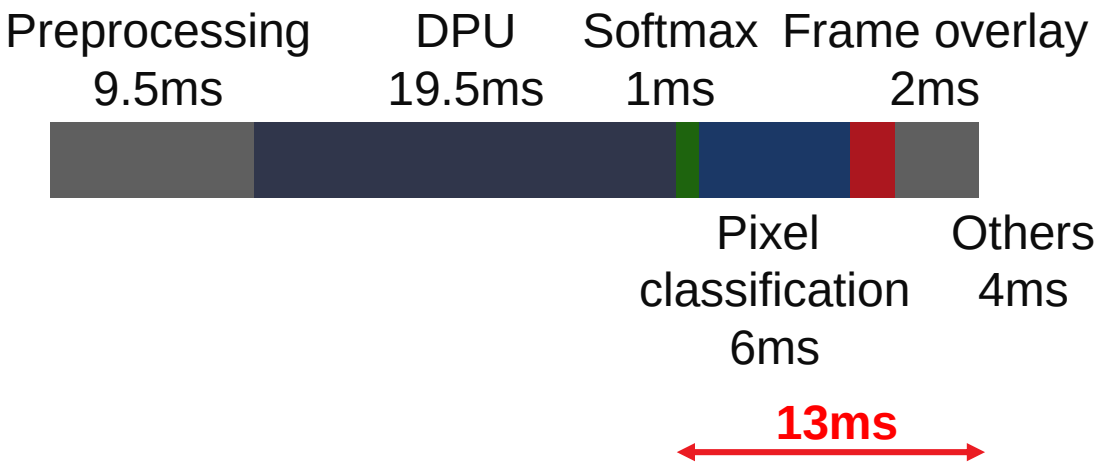
Performance Optimization with End-to-end Profiling

> **Before optimization**



76ms

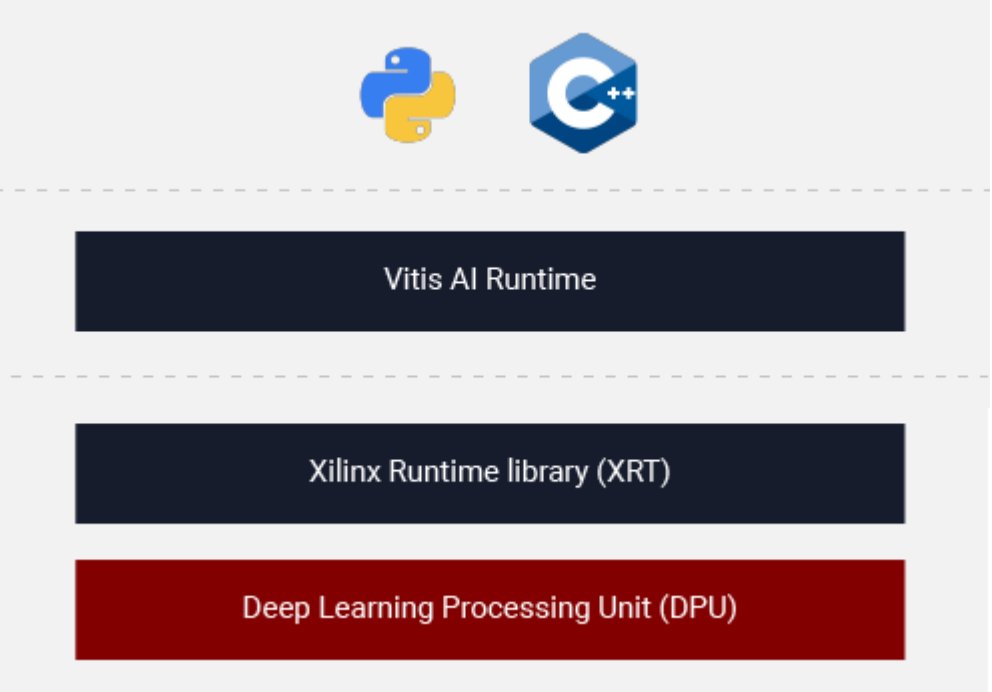
> **After optimization**



13ms

- >> Use Vitis libraries / custom hardware IPs
- >> Use NEON to accelerate the computing
- >> Use Eigen/BLAS library to perform linear superposition

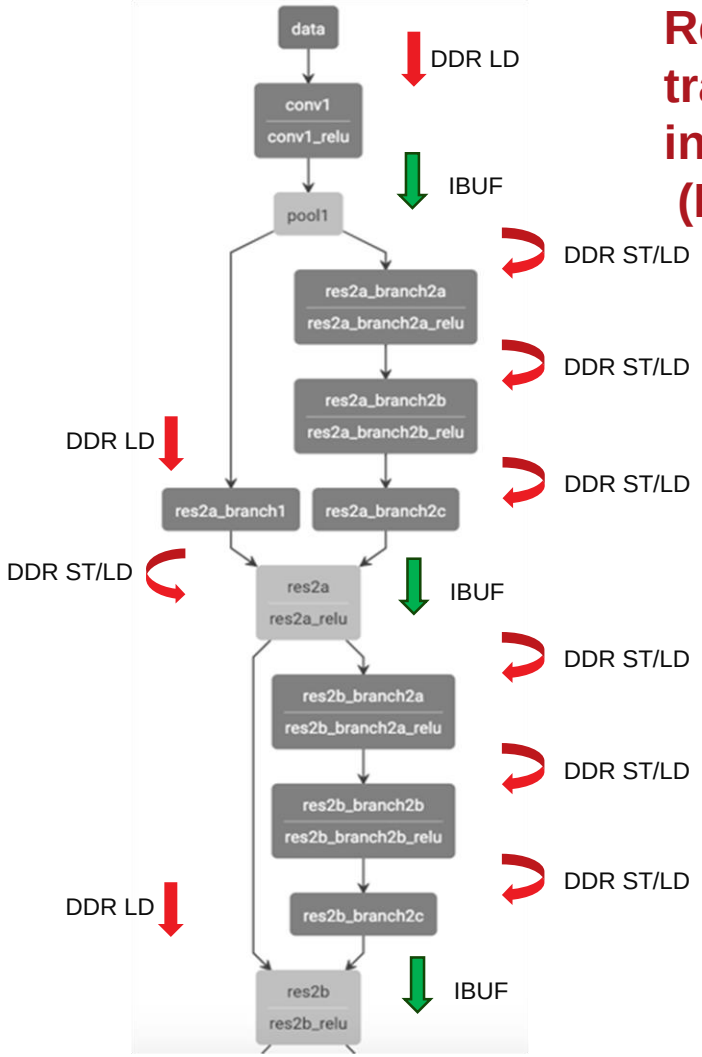
Vitis AI Runtime



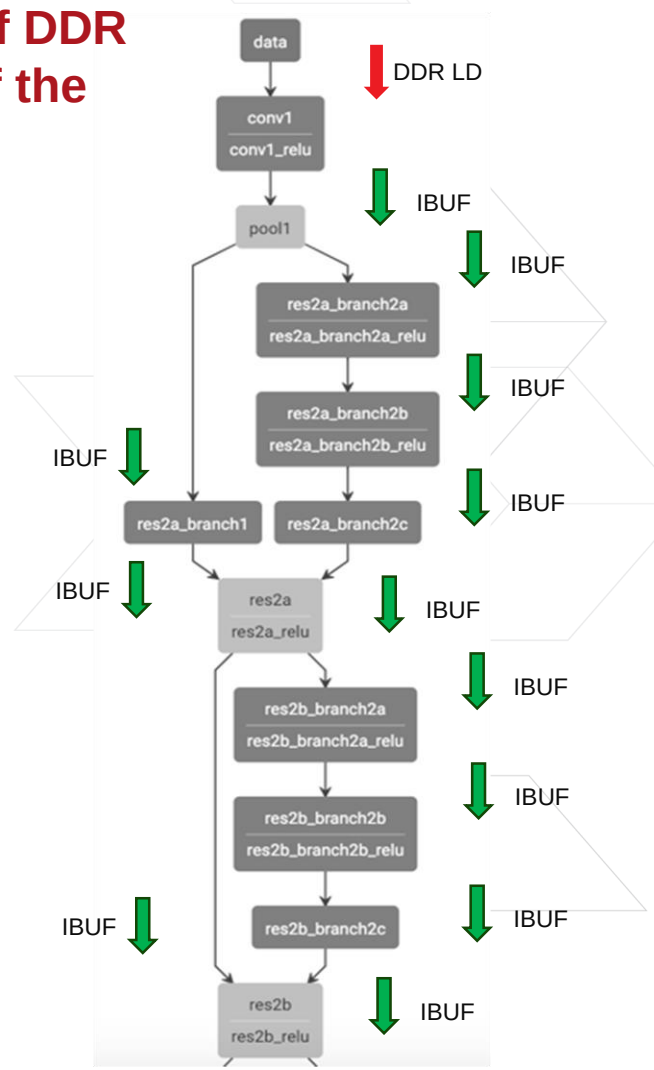
- Vitis AI Runtime
  - Efficient task scheduling
  - Memory management
  - Multi-threading
  - Interrupt handling
- Vitis AI Runtime API
  - A lightweight set of high-level C++ & Python APIs
  - Common for both embedded and Alveo/Cloud

Vitis AI Runtime Features

- Cacheable memory management
- QoS based scheduling
- IFM/OFM buffer detached from intermediate buffer
- Optimize DPU scheduling based on model topology

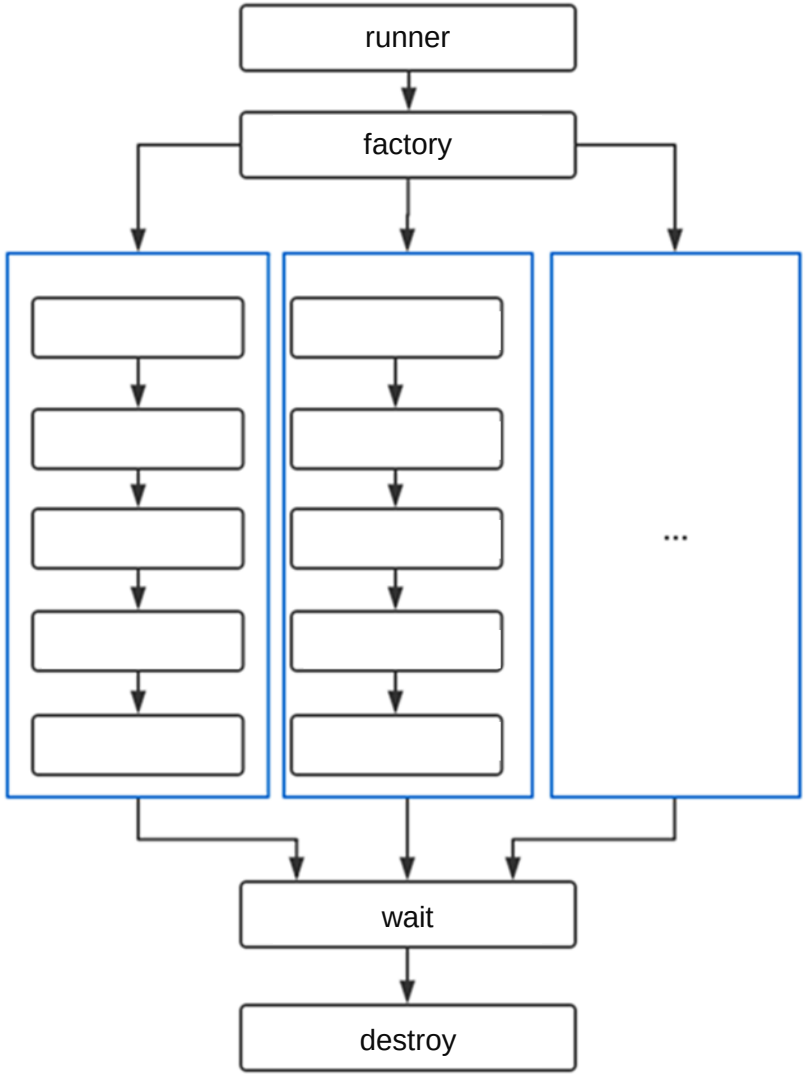


Reduce 80.37% of DDR traffic and 88% of the instructions! (ResNet-50)



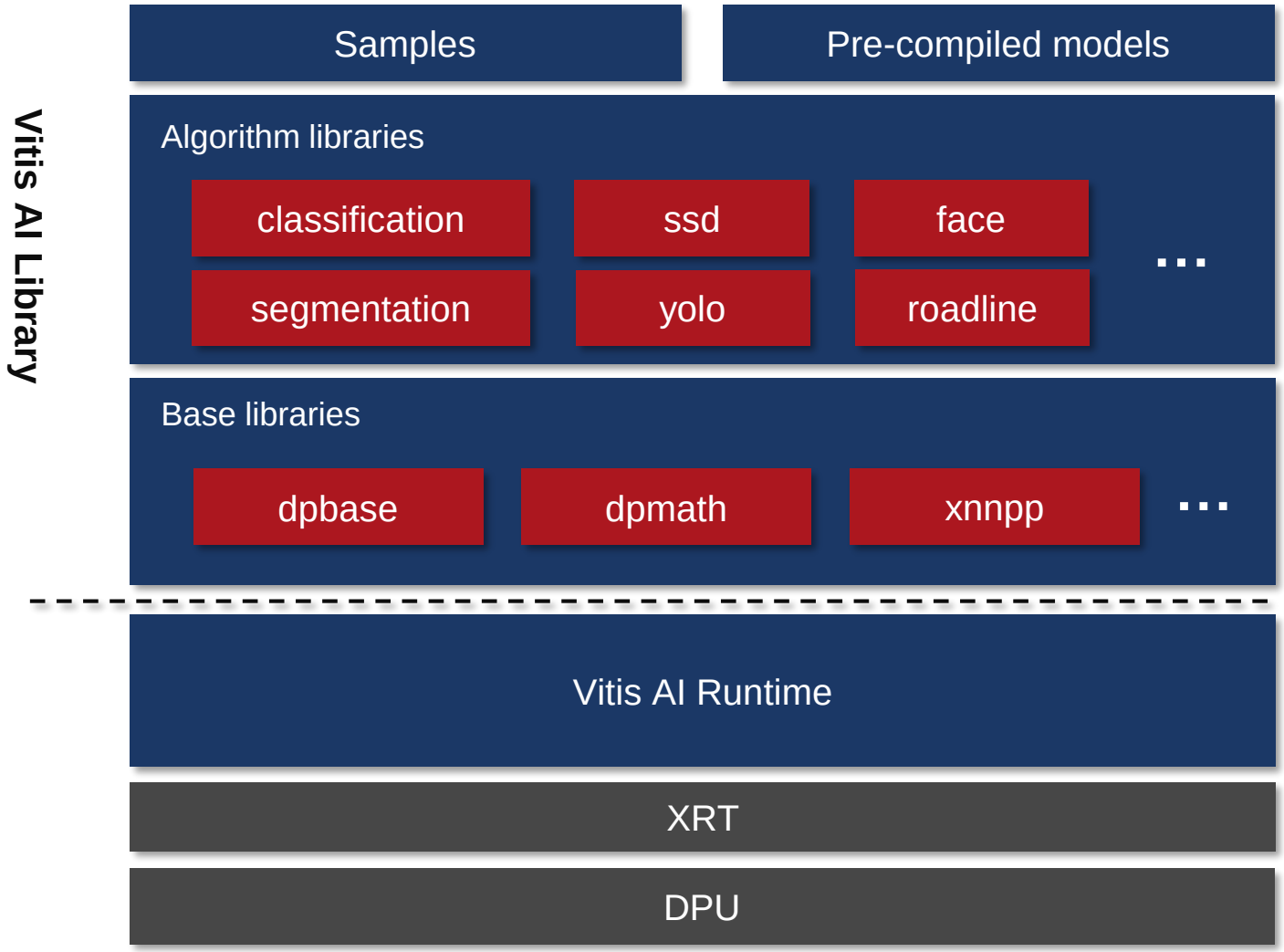
XILINX.

Multi-threading Support



- Multiple models with zero switching overhead
- Multiple threads for better throughput performance
- Multiple processes for multiple channel applications

Vitis AI Library



- Ease of Use
  - Quicker mode deployment
  - Easier AI application development
- Performance Optimized
  - Optimized pre & post processing
- Open
  - Open source
  - Fully sync with Vitis AI Model Zoo

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;

};
```

Implement YOLOv3 in AI Library

- Most important is to implement **run()** method

Pre-processing

Running DPU

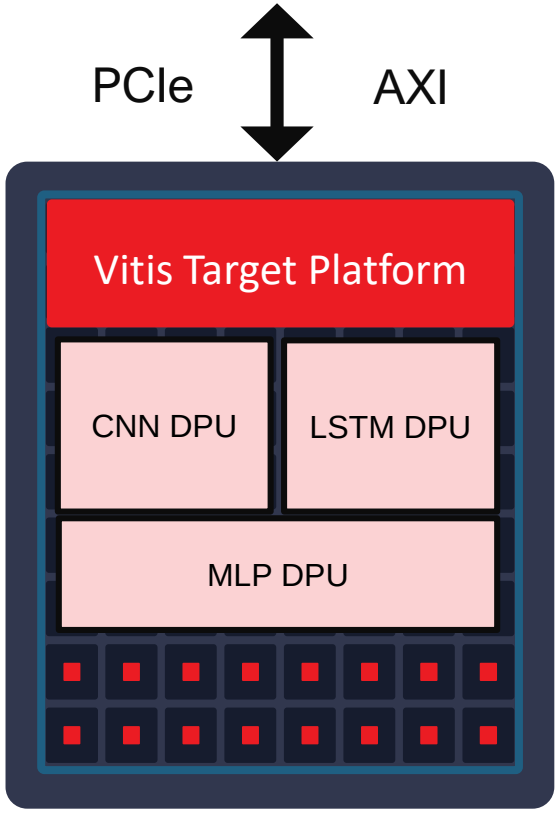
Post-processing

```
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;
}
```

Deploy Yolov3 Using AI Library

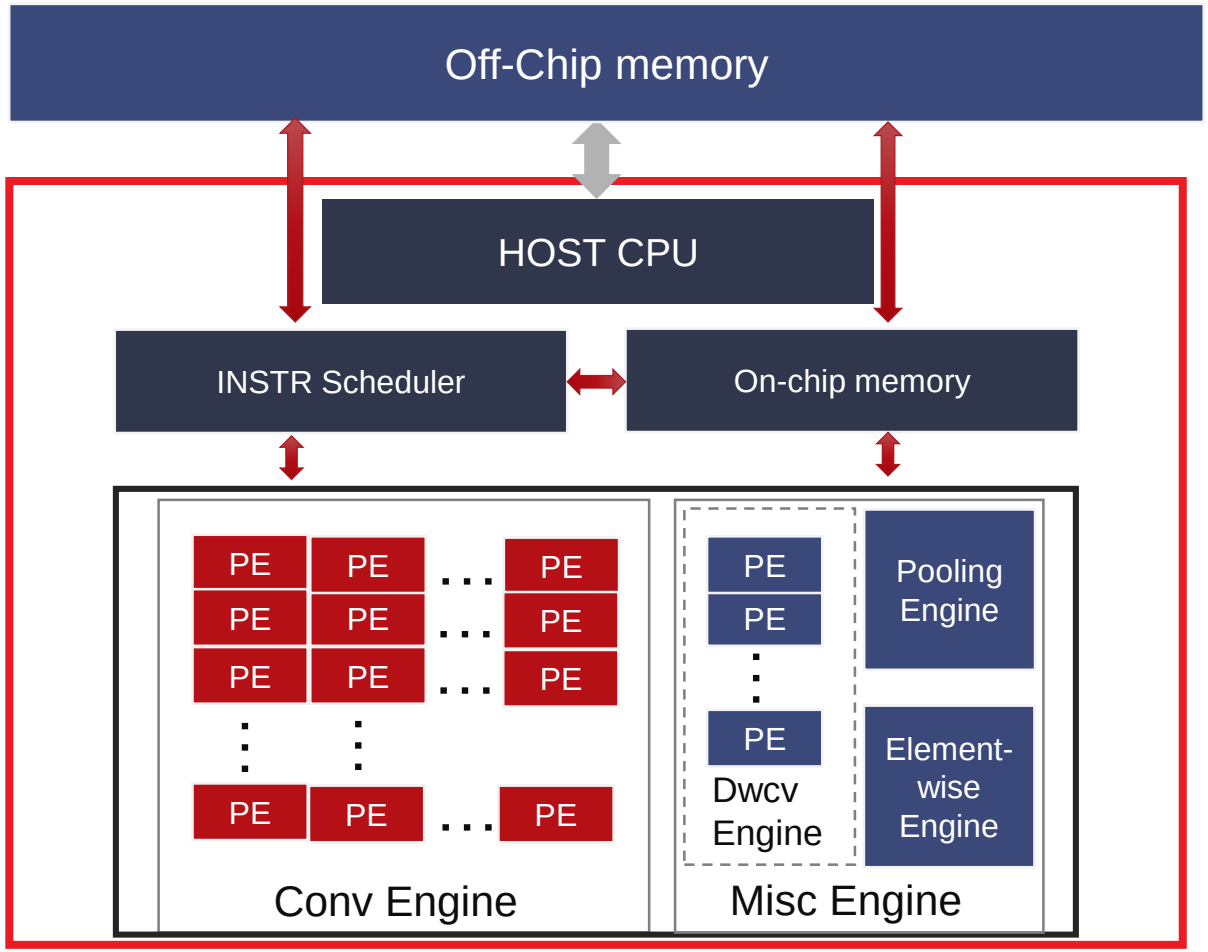
```
cv::Mat img = cv::imread(argv[2]);
auto yolo = xilinx::ai::YOLOv3::create(argv[1], true);

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);
}
```

Deep Learning Processing Unit (DPU)



- **Domain-Specific Architecture for AI Inference**
  - High-efficient Programmable Tensor Processor
  - Scalable across All Xilinx Platforms
- **Tailored for Different AI Workloads**
  - CNN, LSTM, MLP, …
  - Fully HW customizable and adaptable
  - Optimized for throughput, latency or utilization
- **Ease of Use**
  - Easy System Integration with Vitis or Vivado
  - Detailed Refence Design and Tutorials

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

CNN DPUs for Alveo

HIGH

THROUGHPUT

LOW

LATENCY

HIGH

Latency optimized

- Alveo U280 /U50
- Optimized for MobileNet-like models and HBM devices
- Extreme low latency with high-resolution images

Throughput optimized (DDR)

- Alveo U200 / U250
- Optimized for 100% INT8 models and ResNet/Inception Models
- Optimized for pruned models
- High Throughput at medium latency

Throughput optimized (HBM)

- Alveo U280 / U50
- Optimized for ResNet/Inception models and HBM devices
- Low-cost and high power efficiency solution

XILINX.

CNN DPU for Alveo: Applications

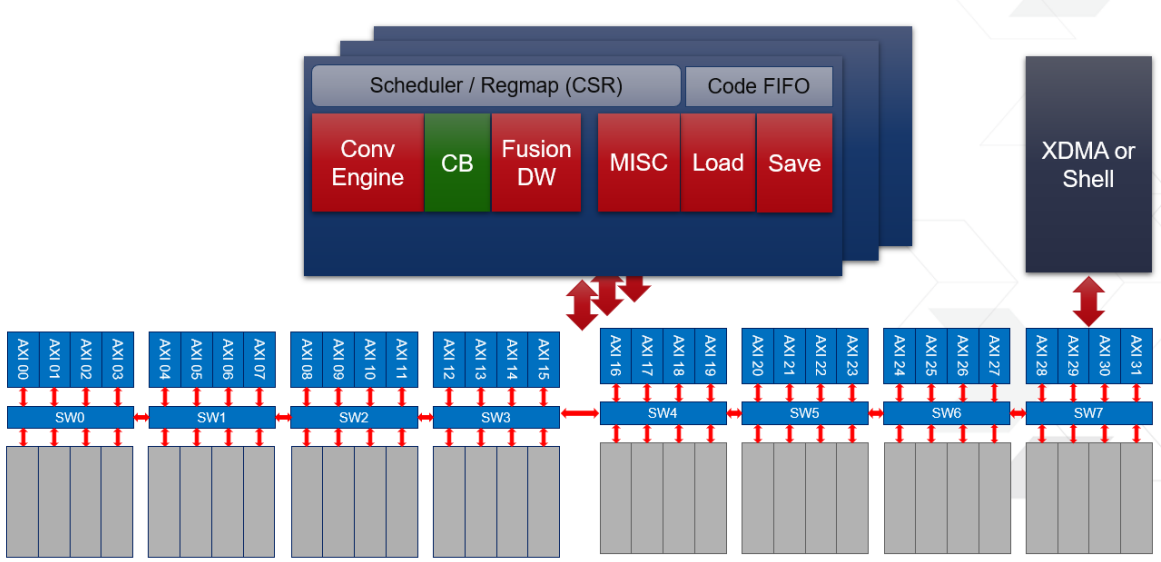
Automated Driving

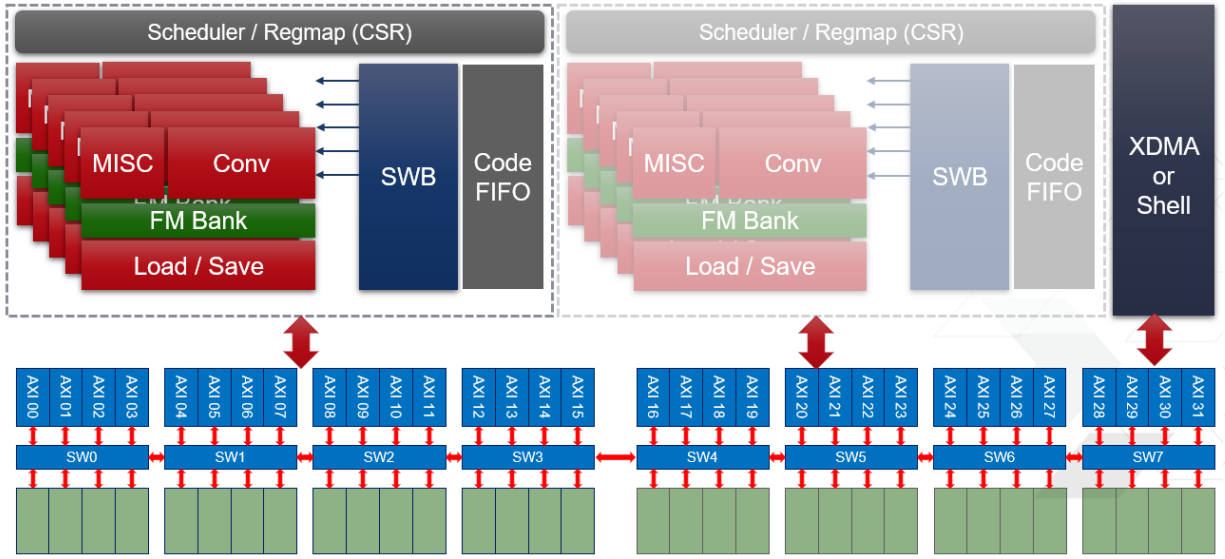
- Ultra low latency DPU
- High HW parallelism for conv & depthwise conv

Data Center

- Flexible pipeline with optimized architecture
- Multi-engine organization for SLR implementation

CNN DPU for Alveo

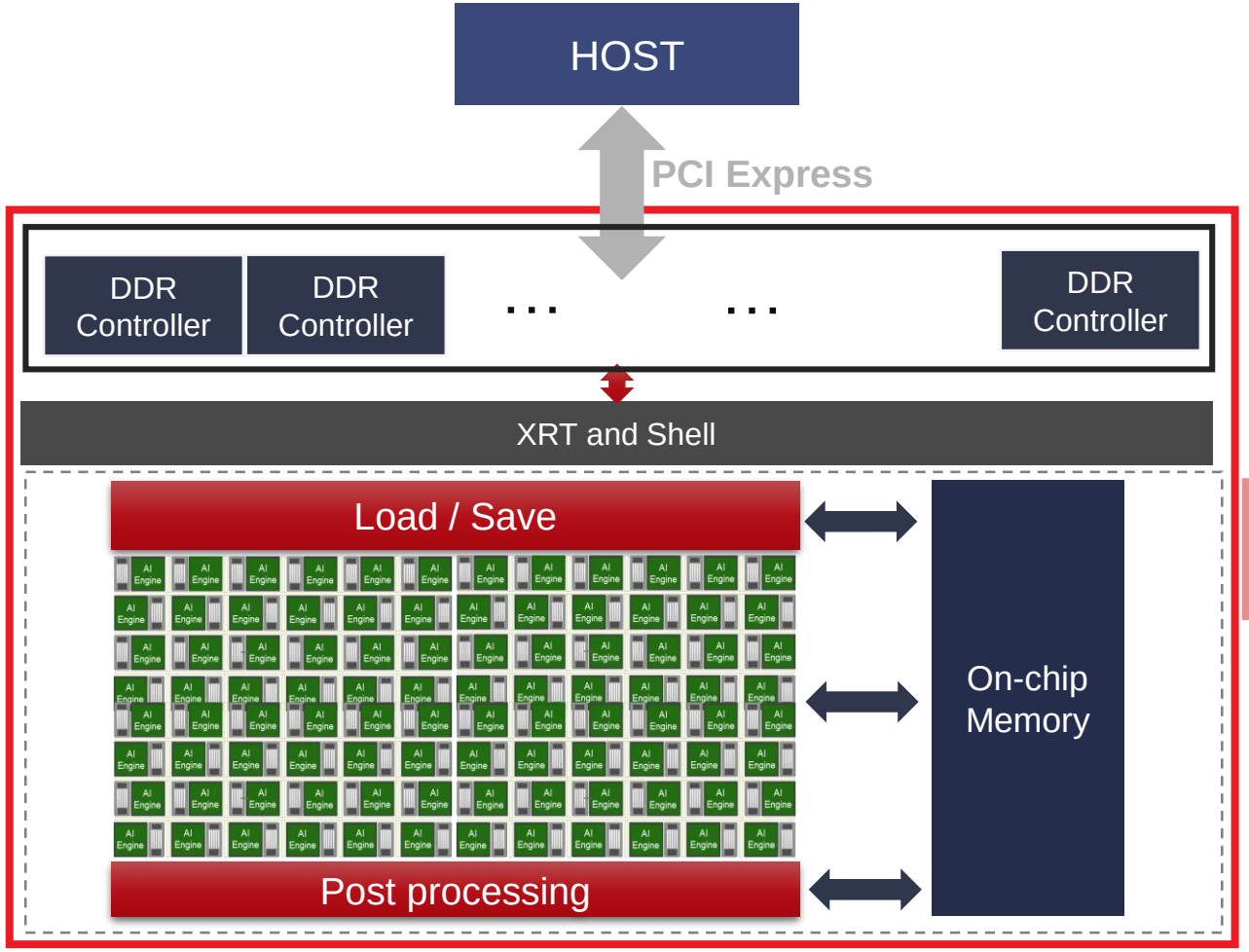




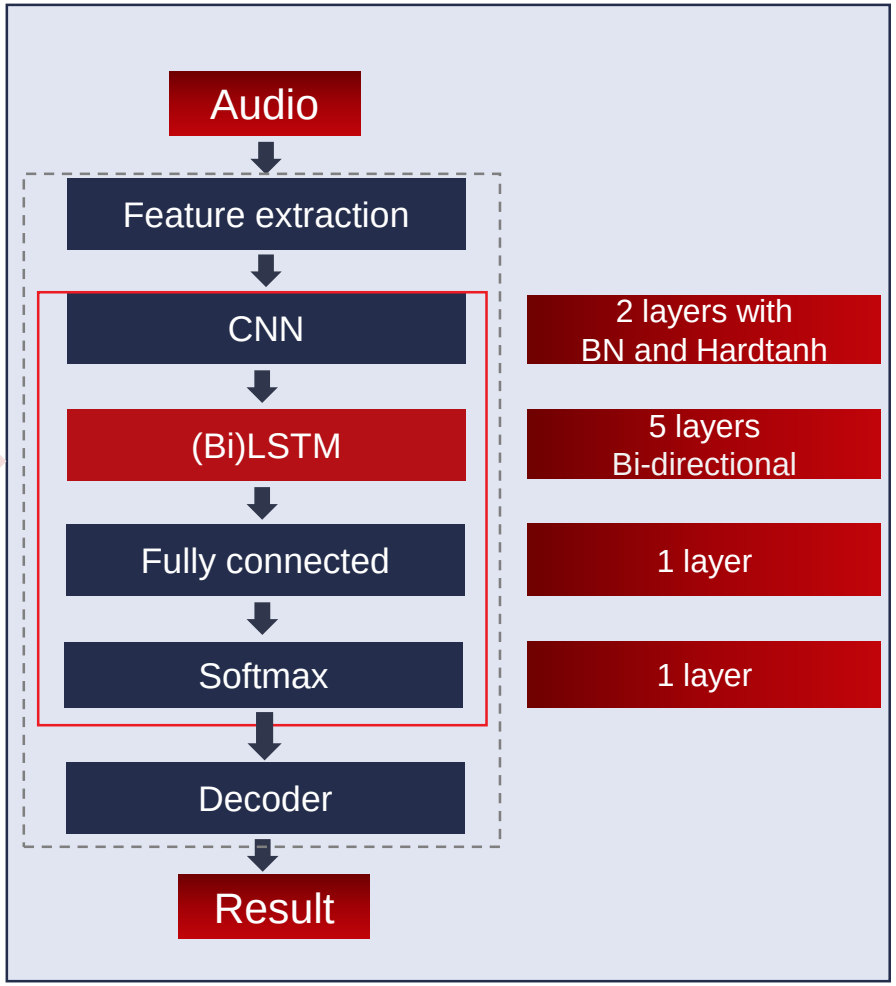
- **Latency Optimized**
  - Mobilenetv2 1920x1080, <13ms @ U50
- Throughput Optimized
  - Resnet50, 1100fps @ U50

XILINX

LSTM DPU for Versal ACAPs



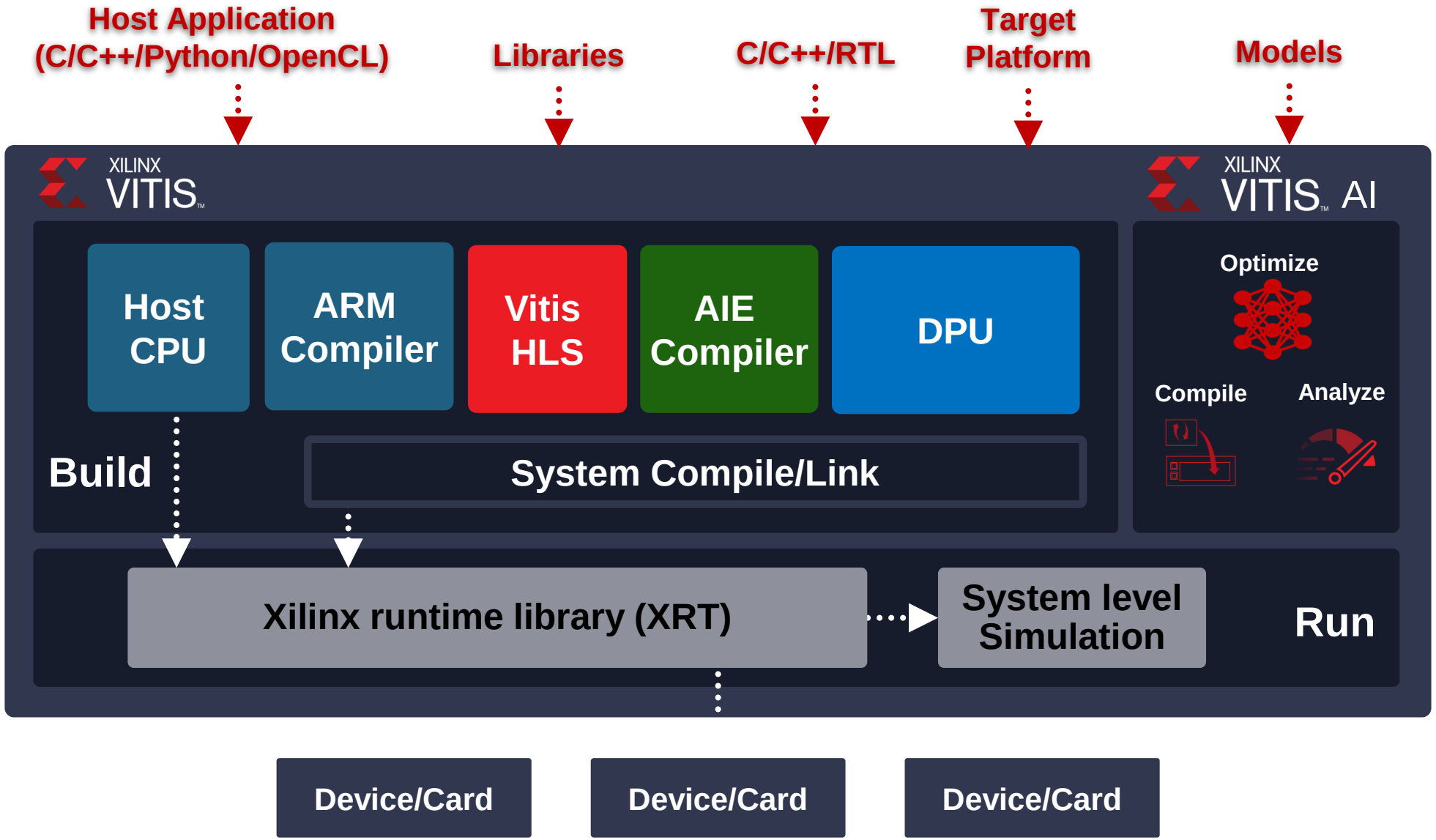
Flow Chart



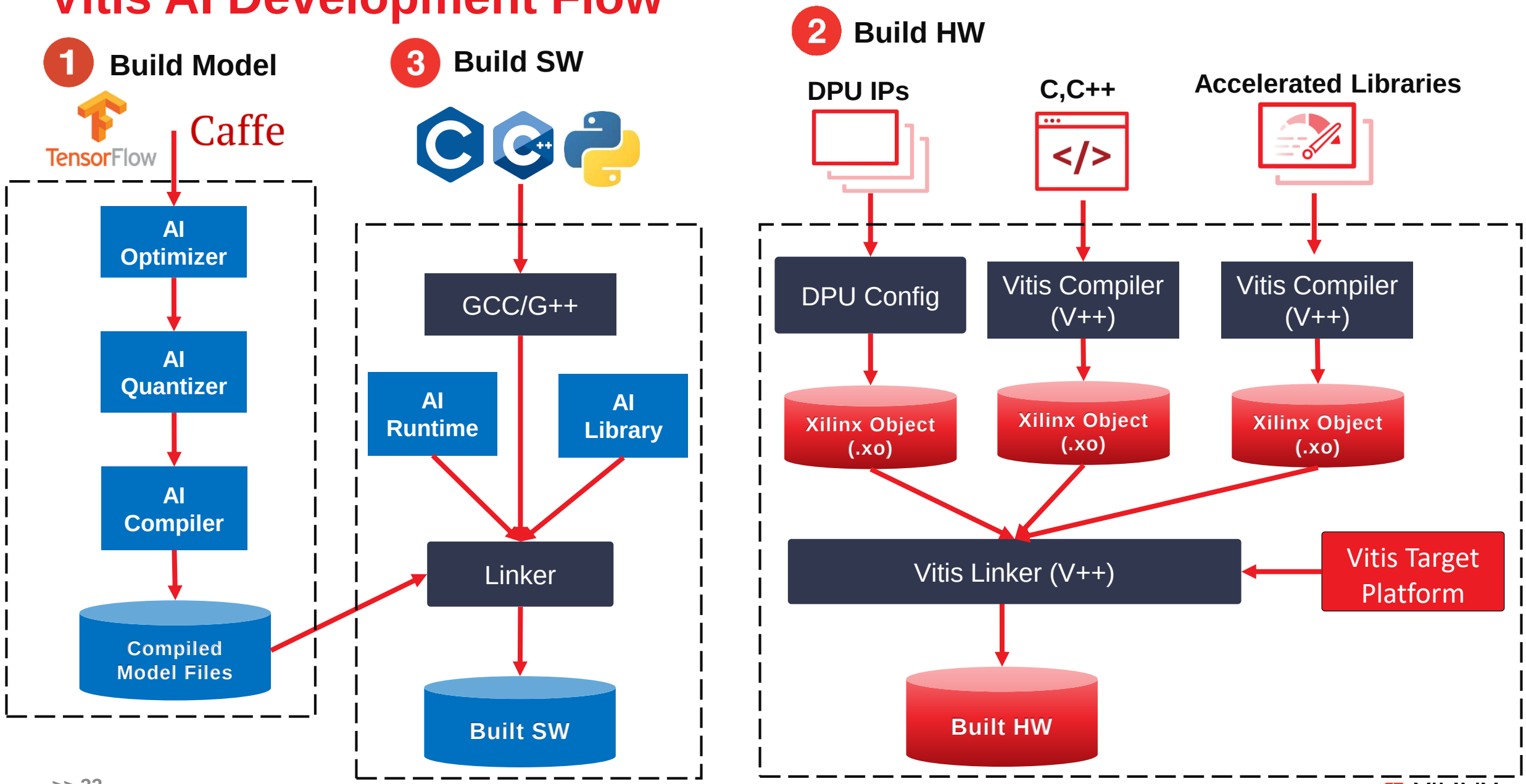
- **Low Latency LSTM Acceleration**
  - Ultra low latency
  - Matrix multiplication in AI Engine

XILINX

Build: Comprehensive Development Tool Suite

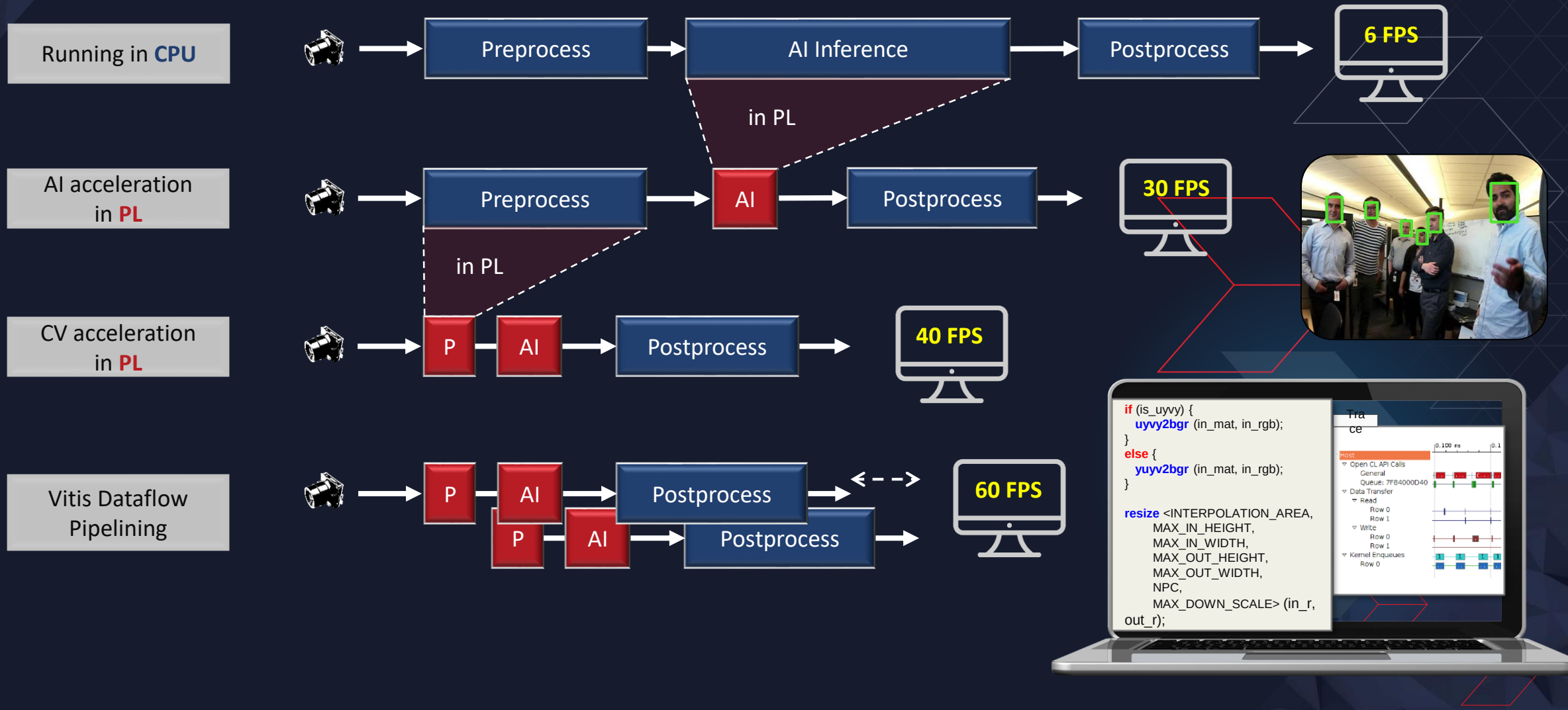


Vitis AI Development Flow



XILINX

An Example of Vitis AI Development



Vitis AI Programming Interface

- Unified high-level C++/Python programming APIs
  - For all DPU
  - Will keep forward compatibility
  - Add new APIs if necessary

- Low-level C++/Python programming APIs
  - Come from legacy DNNDK and keep backward compatibility
  - For Edge DPU (DPUv2) only

Vitis AI Programming with Unified APIs

- **C++ APIs**
  - DpuRunner::create_dpu_runner()
  - DpuRunner::execute_async()
  - DpuRunner::wait()
  - DpuRunner::get_tensor_format()
  - DpuRunner::get_input_tensors()
  - DpuRunner::get_output_tensors()
- **Python APIs**
  - runner.Runner()
  - runner.execute_async()
  - runner.wait()
  - runner.get_tensor_format()
  - runner.get_input_tensors()
  - runner.get_output_tensors()

Vitis AI Samples with Unified APIs

ID	Example Name	Models	Framework	Notes
1	resnet50	ResNet50	Caffe	Image classification with Vitis AI unified C++ APIs.
2	resnet50_mt_py	ResNet50	TensorFlow	Multi-threading image classification with Vitis AI unified Python APIs.
3	inception_v1_mt_py	Inception-v1	TensorFlow	Multi-threading image classification with Vitis AI unified Python APIs.
4	pose_detection	SSD, Pose detection	Caffe	Pose detection with Vitis AI unified C++ APIs.
5	video_analysis	SSD	Caffe	Traffic detection with Vitis AI unified C++ APIs.
6	adas_detection	YOLO-v3	Caffe	ADAS detection with Vitis AI unified C++ APIs.
7	segmentation	FPN	Caffe	Semantic segmentation with Vitis AI unified C++ APIs.

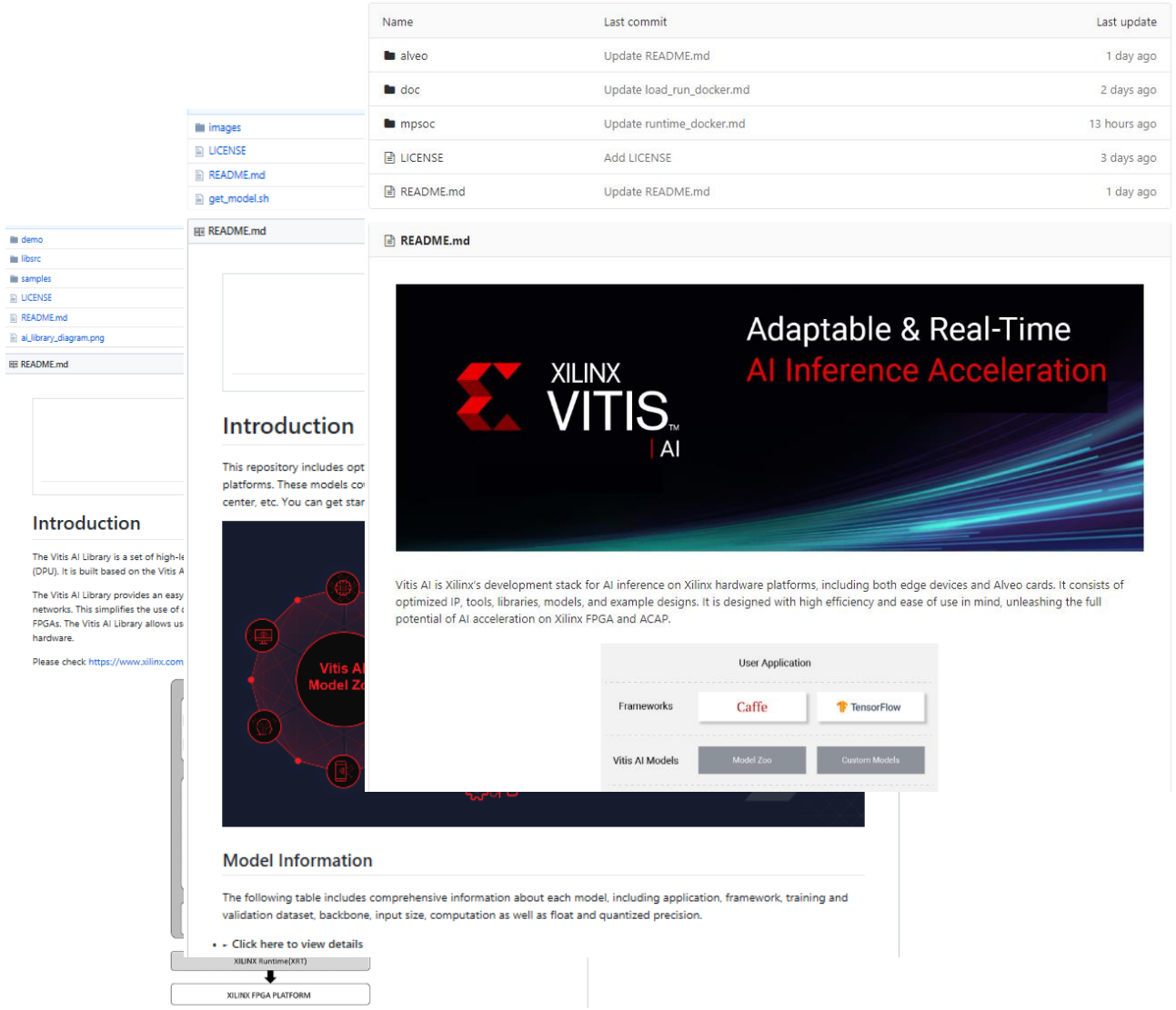
Vitis AI Programming with Unified APIs

- **Sample1 - ResNet50**
  - Caffe
  - C++ APIs
  - https://github.com/Xilinx/Vitis-AI/tree/master/mpsoc/vitis_ai_samples_zcu102/resnet50
  - https://github.com/Xilinx/Vitis-AI/tree/master/alveo/examples/vitis_ai_alveo_samples/resnet50

- **Sample2 - Inception-v1**
  - TensorFlow
  - Python APIs
  - https://github.com/Xilinx/Vitis-AI/tree/master/mpsoc/vitis_ai_samples_zcu102/inception_v1_mt_py
  - https://github.com/Xilinx/Vitis-AI/tree/master/alveo/examples/vitis_ai_alveo_samples/inception_v1_mt_py

Quick Links to Start Developing Today !

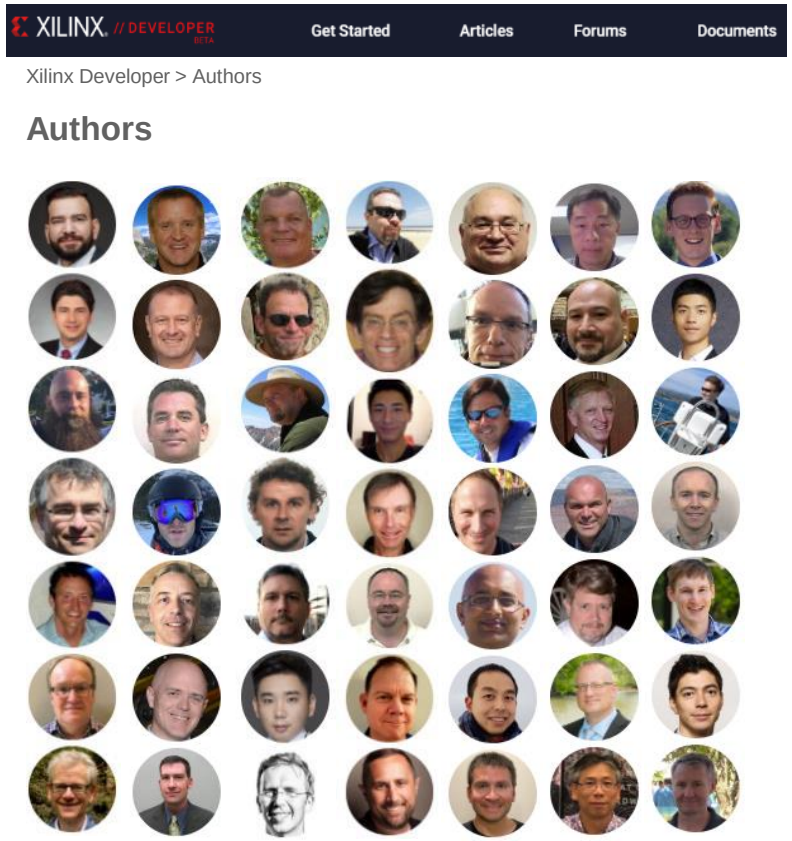
- [Getting Started from Vitis AI Github]
- [Vitis AI Model Zoo]
- [Vitis AI Optimizer Lounge*]
- [Vitis AI DPU TRD]
- [Vitis AI Library]
- [Vitis AI Tutorial]
- [Vitis AI Toolbox on SourCe]
- [Vitis AI Forum]

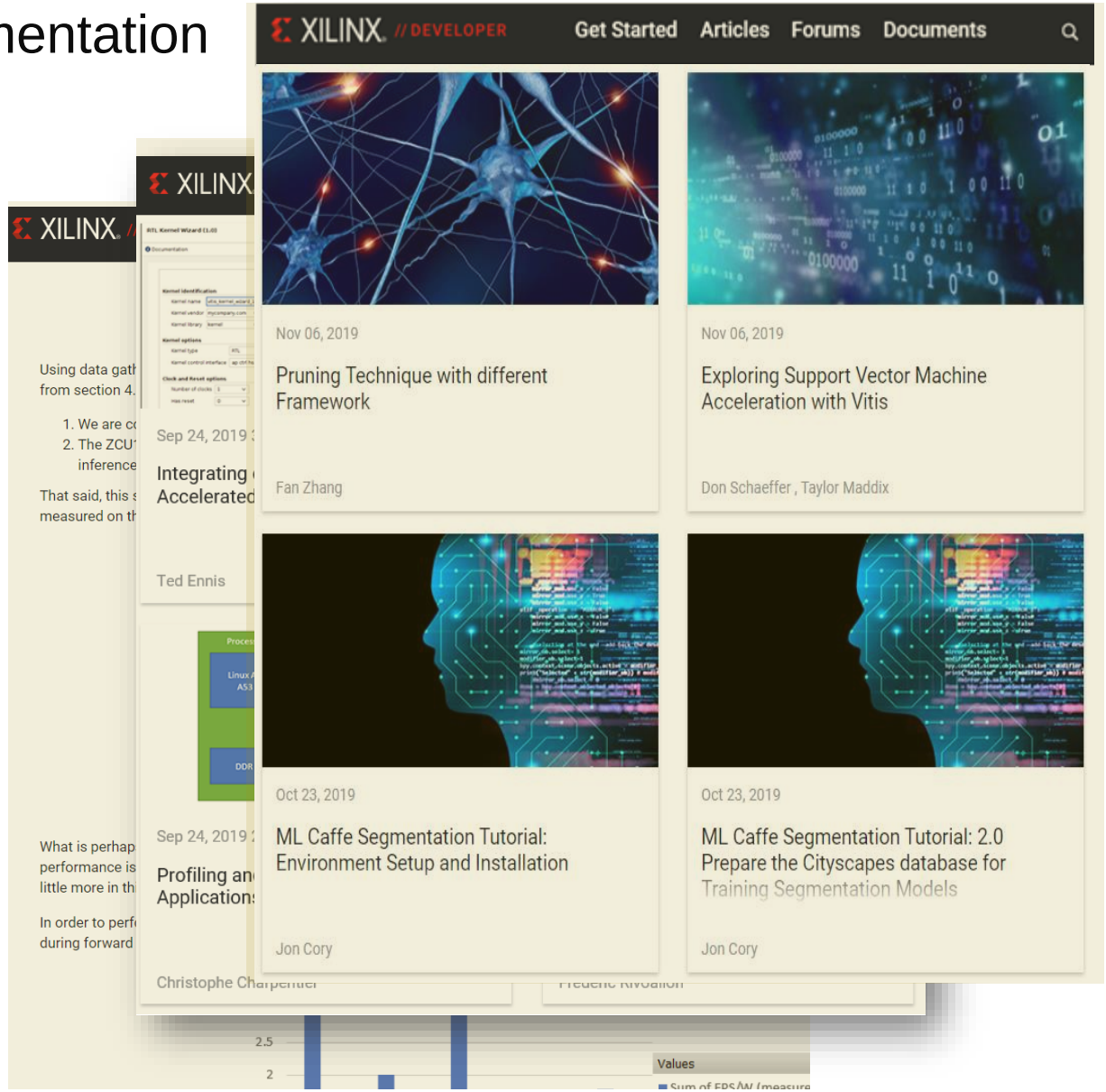


*** Approval required**

Visit *developer.xilinx.com & Get Started Now !*

Easy Access to Examples, Tutorials, Documentation





Connecting developers to experts

30+ expert articles & projects and growing

XILINX

XILINX VITIS™ | AI

Adaptable & Real-Time AI Inference Acceleration

- **Unified AI Platform**
  - Cloud to edge with unified environment and APIs
  - Comprehensive models and libraries
- **Work at Speed of Innovation**
  - Hardware adaptable to models & workloads
  - From model to implementation in minutes
- **Whole Application Acceleration**
  - End-to-end X + AI acceleration
  - Heterogenous integration & acceleration through Vitis

Adaptable.
Intelligent.

XILINX