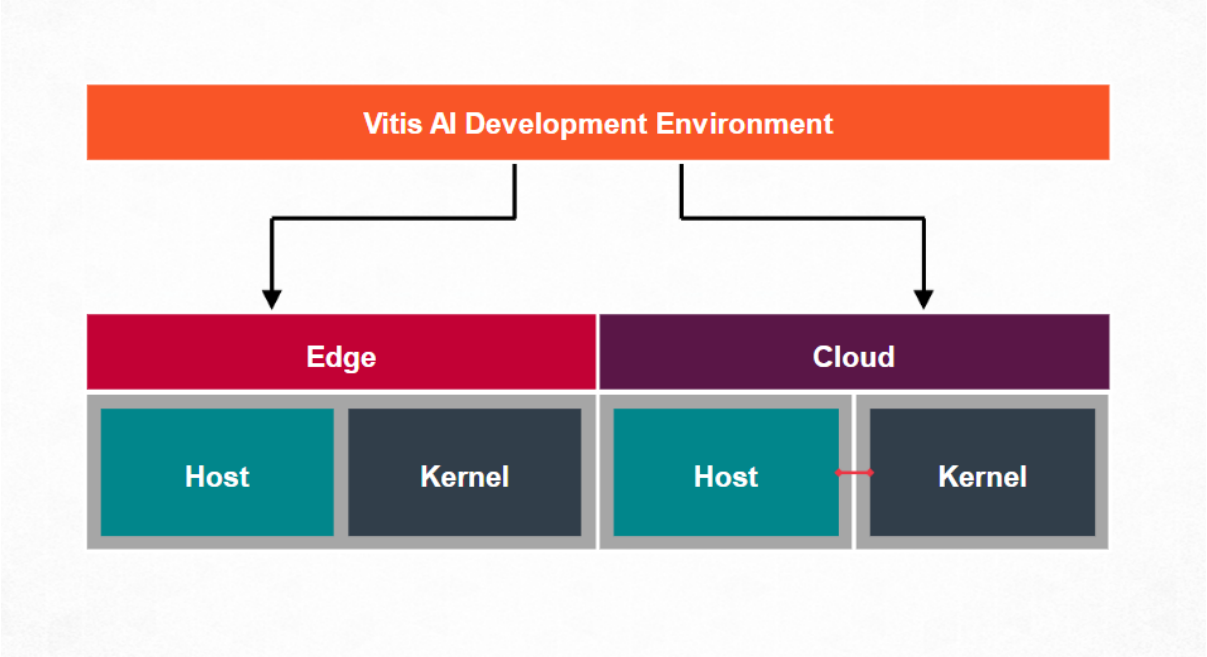
AI evolves rapidly - adapt to your workload
The speed of innovation has readily outpaced silicon cycles for ASICs, CPUs, and GPUs. FPGAs and adaptive SoCs help keep up with innovations by using flexible and reconfigurable architectures.

For eg- if your Asic is optimized for one type of DNN, by the time the ASIC development cycle is complete and the silicon is ready, it is very likely that a more efficient DNN might be available, rendering the time and expenses of developing the ASIC a waste.

Adaptive hardware enables rapid, domain-specific architecture changes on the same silicon device.

Vitis AI Environment

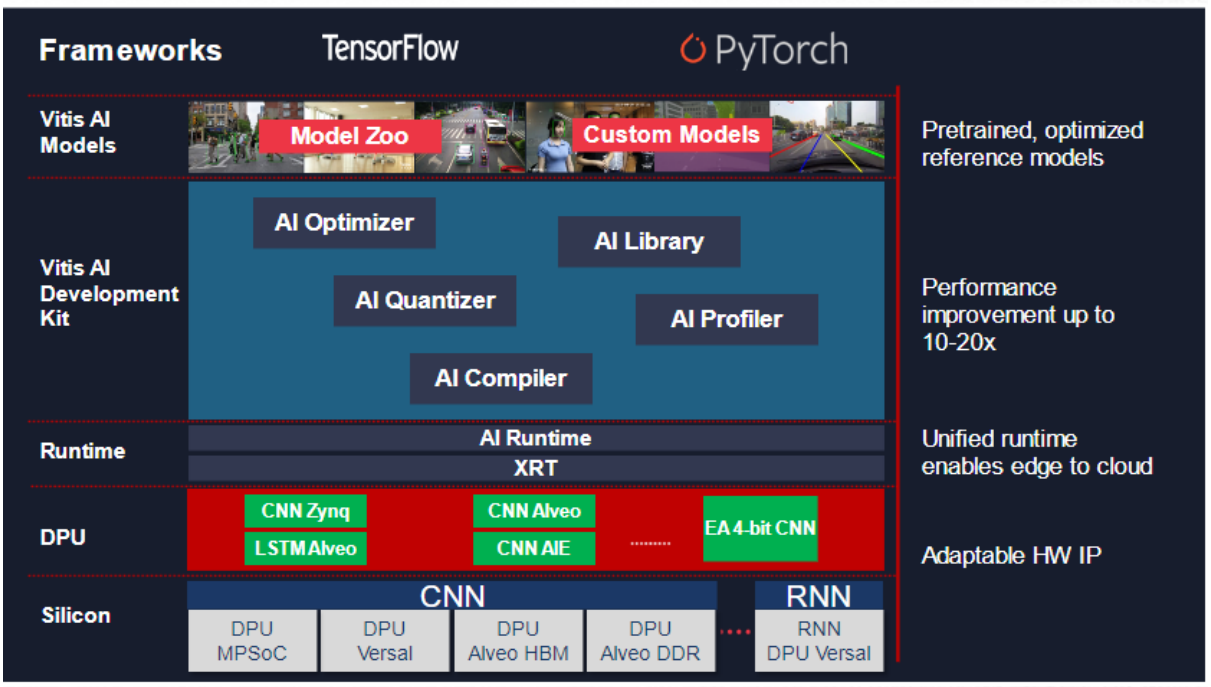


Edge devices - contain “host” processor and accelerators within the same package
Cloud devices - separate “host” processors and accelerators typically connected through PCIe interconnect.

Vitis AI development Environment consists:

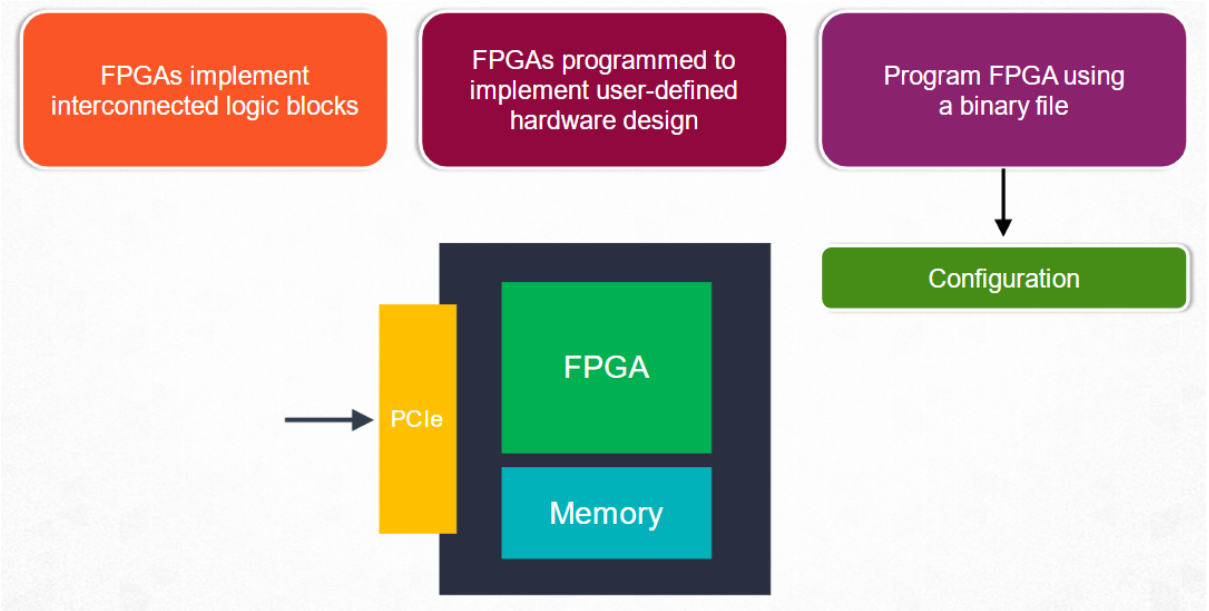
Vitis AI Development Environment Features

AI Quantizer: Bit reduction, Calibration, Fine Tuning
AI Optimizer: Prune model upto 90%
AI Compiler: Maps AI model to a highly-efficient instruction set and data flow
Sophisticated optimizations, such as layer fusion, instruction scheduling, and on-chip memory reuse
AI Library: Offers unified high-level C++ and Python APIs
AI Profiler: Powerful tool to perform a layer-by-layer analysis to locate bottlenecks



Build and Deploy AI to All Platforms

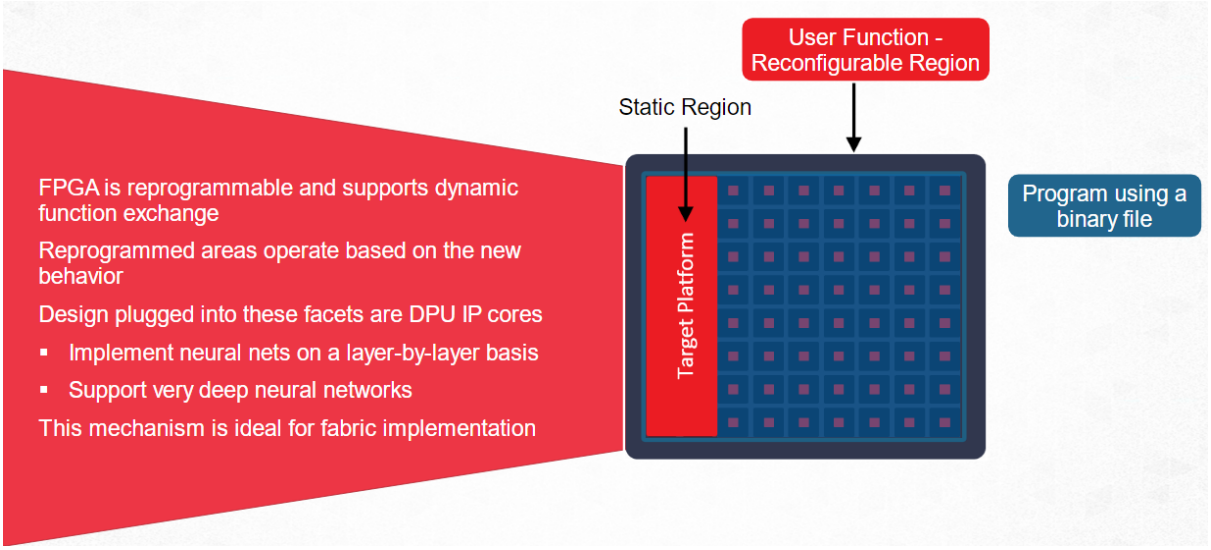
Overlay Bins



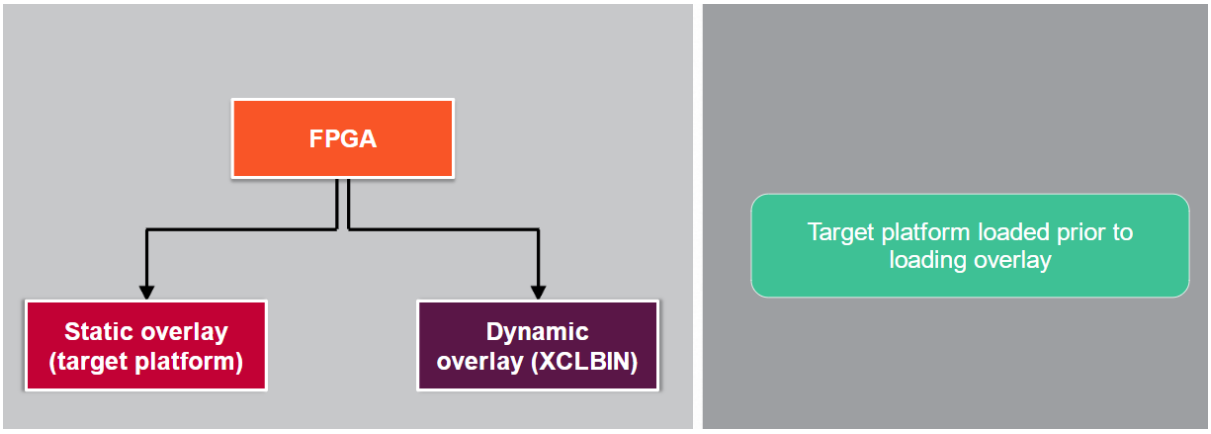
An accelerator kernel refers to a portion of code that is offloaded to an accelerator device, such as a GPU (Graphics Processing Unit) or an FPGA (Field-Programmable Gate Array), for parallel execution. The accelerator device is designed to perform computations in parallel, which can significantly speed up certain types of computations compared to running them on the CPU (Central Processing Unit).

- FPGA are reconfigured to change the behaviour of the logic and modify the way these blocks are connected to suit the designer's needs.
- Once fully configured, it begins behaving as the user intended.
- For FPGA to compute, a supportive framework (accelerator kernel) is required that contains a "pluggable" interface for the kernels to connect to the supporting logic.
- Data and commands are exchanged through a high-performance connection with the host processor cluster.
- Cloud based models use the PCIe interface, and edge models use AXI connections.
- Since data often arrives in frames, off device memory must be available for short-term data storage.

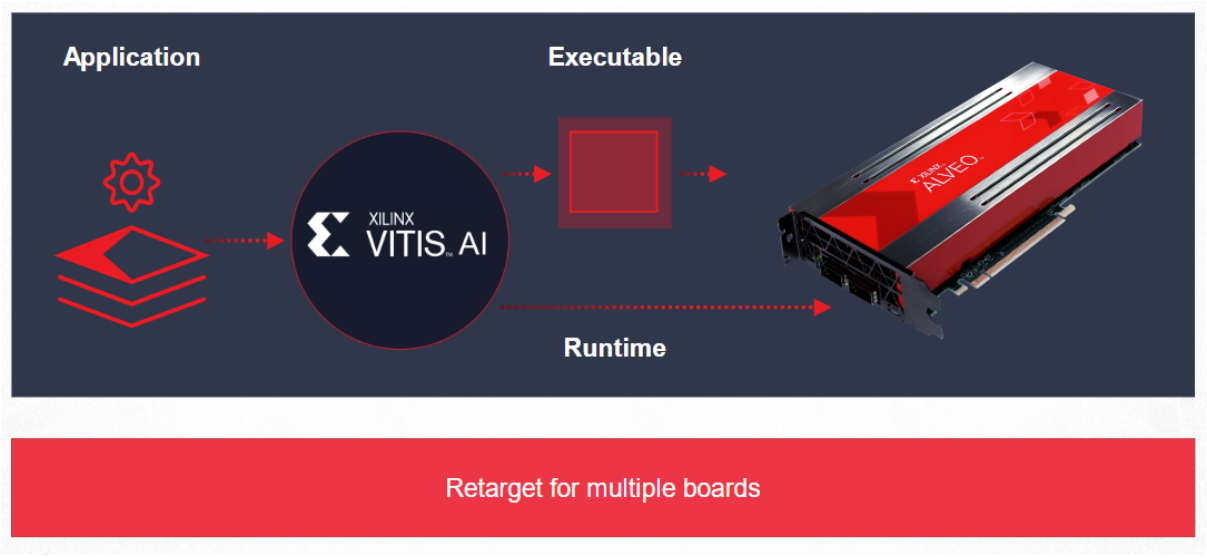
Whole deployment process



FPGA Partition

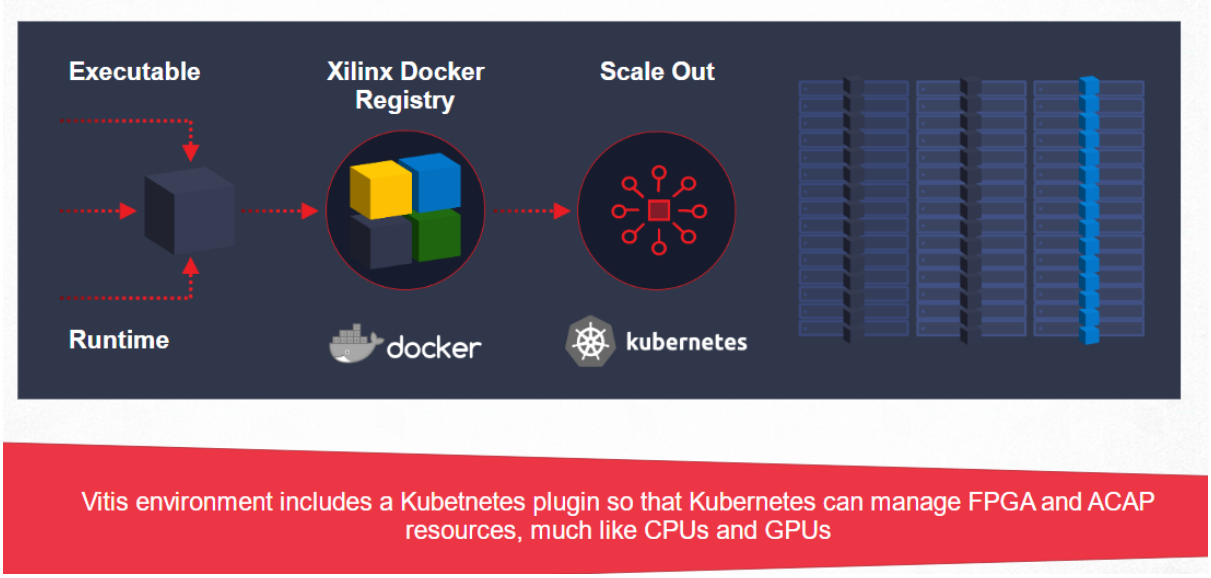


Deploy: Embedded Deployment/ Single Server Deployment

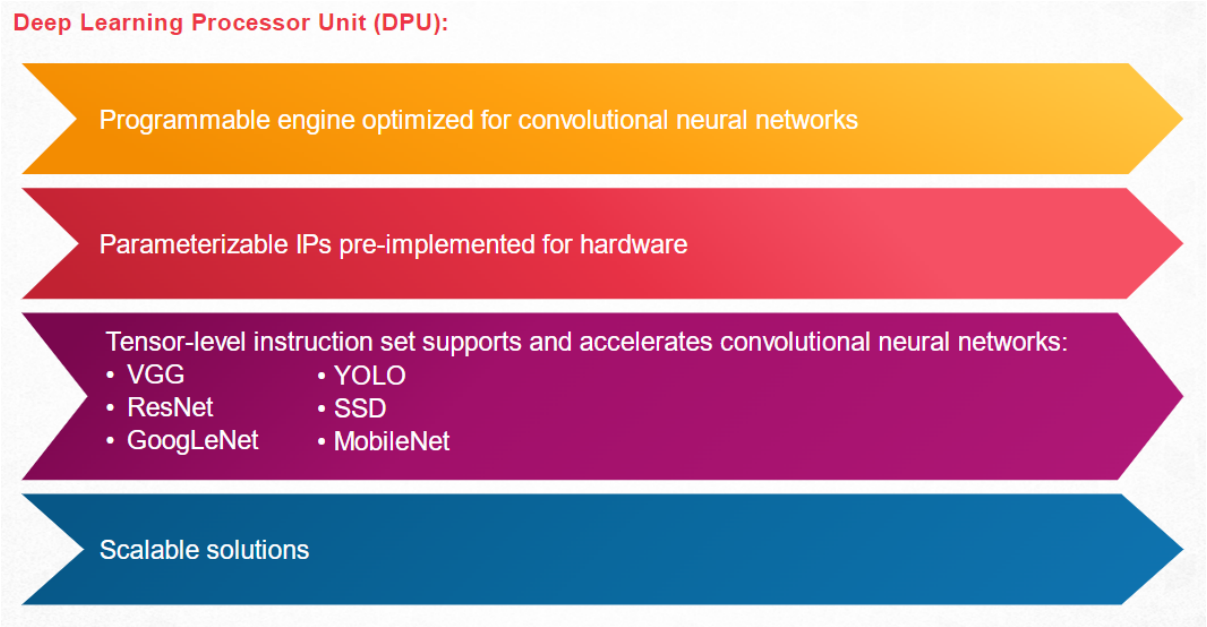


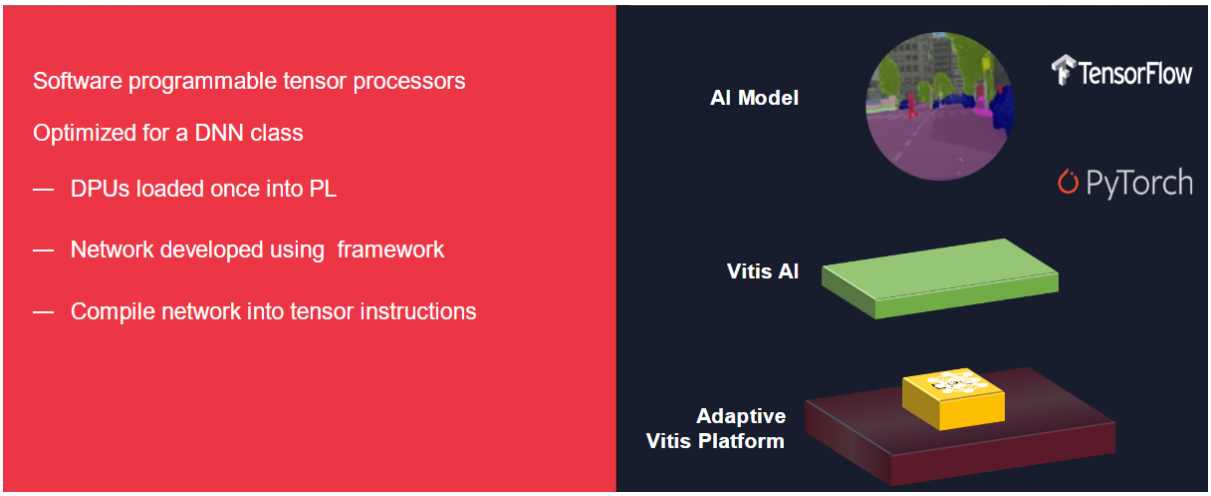
Deploy: Scale-Out Deployment

For scale-out deployment across many servers in a data center, your application and the XRT can be packaged into a Docker container as a service and then deployed using Kubernetes.



From Model to Implementation in Minutes
Vitis AI development environment offers a powerful way to implement AI inferencing with a series of Deep Learning Processor Units (DPUs).





Deep Learning development stages:

1. **Training**
   Training a neural network involves feeding a very large quantity of known good labeled data into the input layer of the network. This label data is selected based on the goal of the network.
   Training typically uses floating point numbers to represent the weights and is very computationally intensive. Fielded networks are optimized to use integer numbers as weights, so they are less accurate but can still provide very good results.
2. **Inference**
   Inference stage is the process of optimizing and implementing the network in the field. Because the training stage typically uses floating point numbers and these require more computational resources, optimization is performed to reduce the

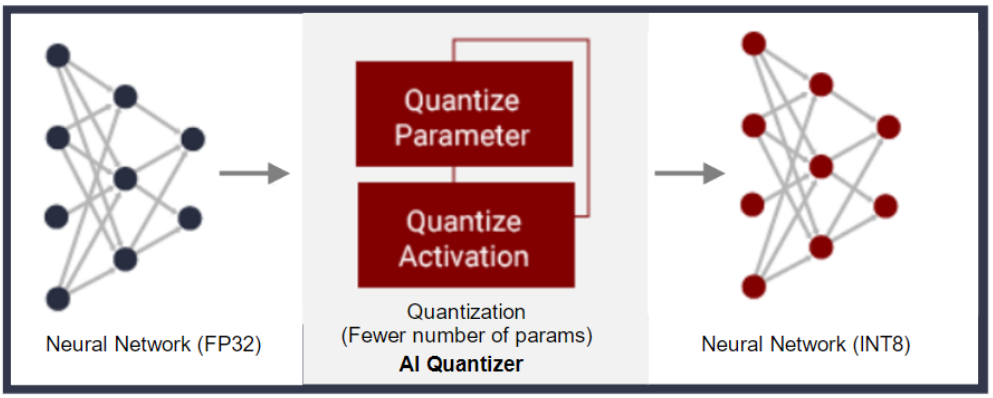
amount of these resources used. This has the additional benefit of letting the network run faster and with less power. Typical optimizations include conversion from floating point numbers to fixed point numbers and trimming. Inference uses this train network to predict or estimate outcomes from new observations.

Vitis AI Development Kit - Tool Flow

Innovative workflow for deep learning interference DPU applications

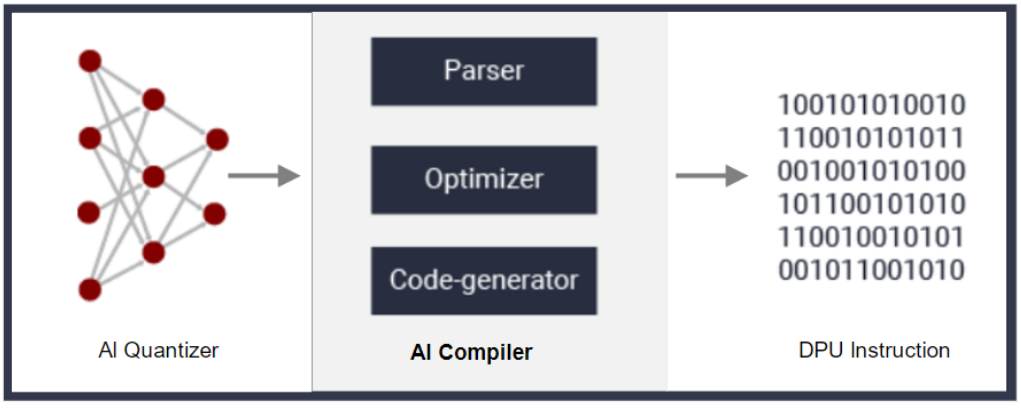
1. Quantize the Neural Network Model
2. Compile the Neural Network Model
3. Program with the Vitis AI Programming Interface
4. Run and Evaluate the Deployed DPU application

1. Quantize the Neural Network Model



- Converts 32-bit floating-point weights and activations to fixed-point representation
- Balances performance and accuracy
- Less bandwidth
- Faster speed/lower power

2. Compile the Neural Network Model



- Maps model to instruction set and data flow
- Optimizations:
  - Layer fusion, instruction scheduling
- Reuses on-chip memory

3. Program with the Vitis AI Programming Interface

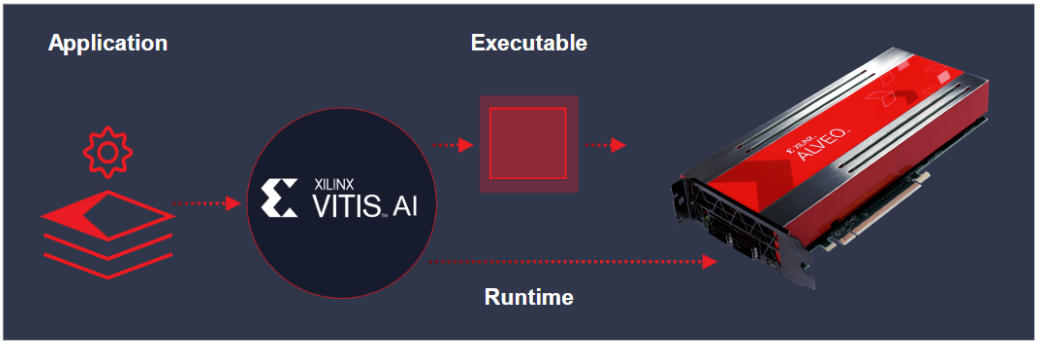
```
class YOLOv3 {
public:
  static std::unique_ptr<YOLOv3> create(const std::string &model_name,
                                        bool need_preprocess = true);
protected:
  explicit YOLOv3();
  YOLOv3(const YOLOv3 &) = delete;
public:
  virtual ~YOLOv3();
public:
  virtual int getInputWidth()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

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

};
```

Unified set of high-level C++/Python programming APIs to run AI applications across edge-to-cloud platforms

- DPUCAHX8H for Alveo U50/U55C cards
- DPUCZDX8G for Zynq Ultrascale+ MPSoCs and Zynq-7000 SoCs

- Allows easy porting of AI applications from cloud to edge and vice versa

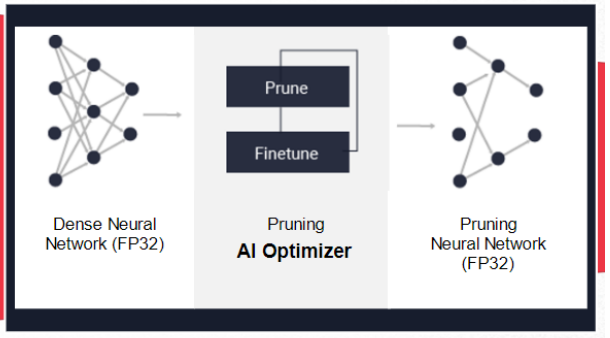
4. Run and Evaluate the Deployed DPU Application



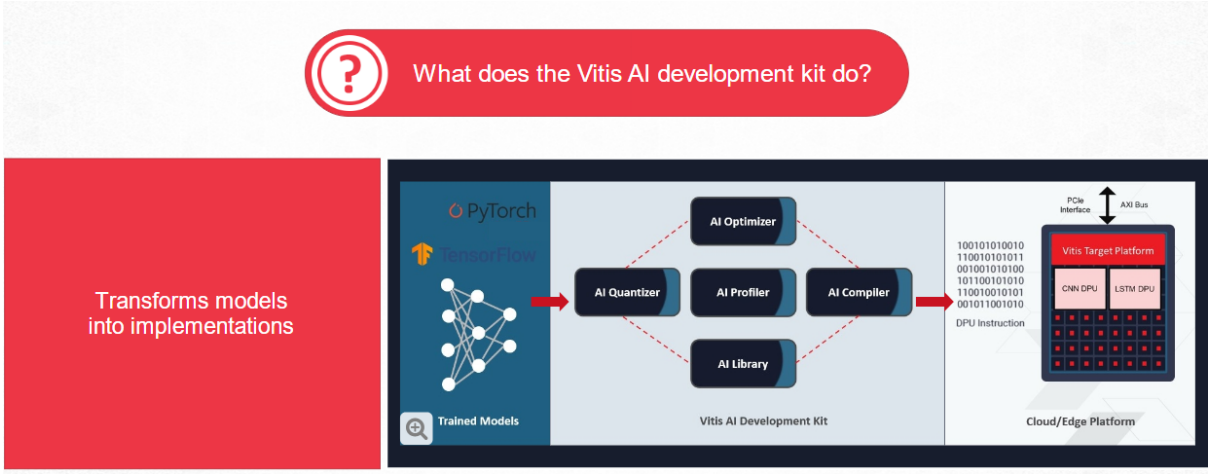
Retarget your application for other boards

Vitis AI Development Kit - Tool Flow - Optional

- Optional pruning optimizer
- Requires a commercial license
- Performed before quantization

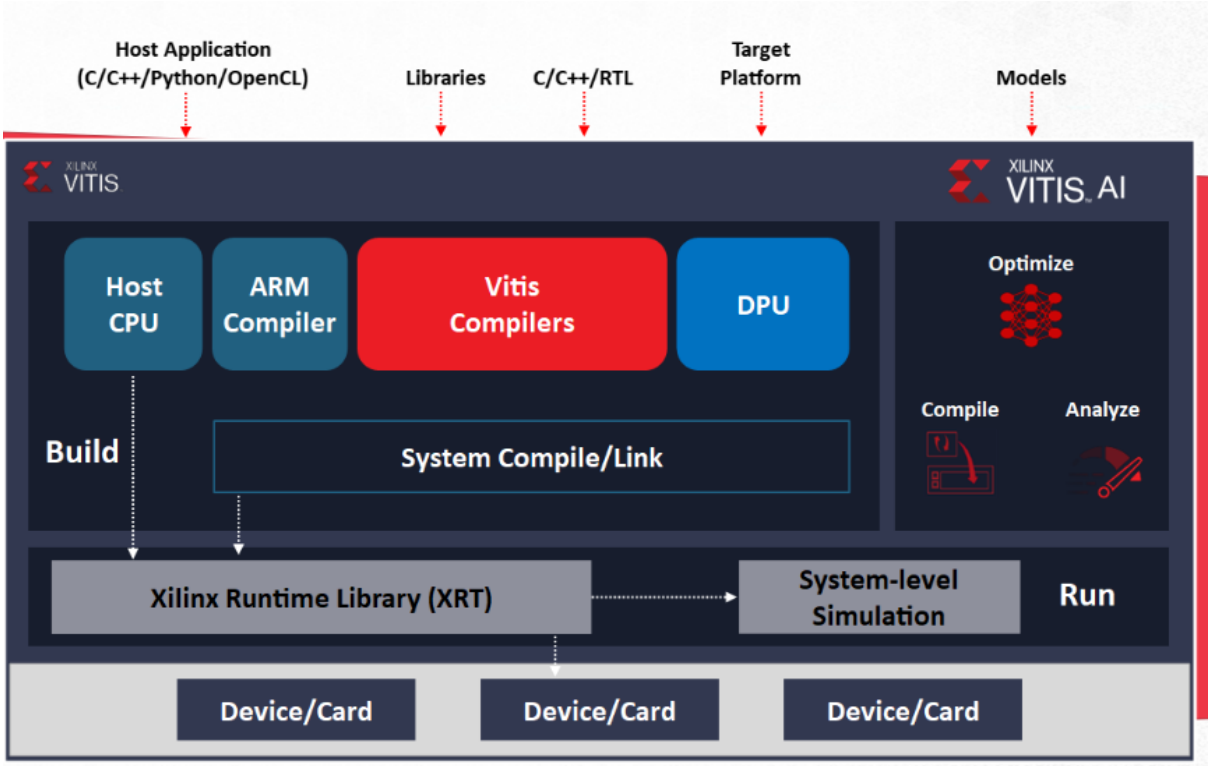


Direct Framework Compilation in Minutes



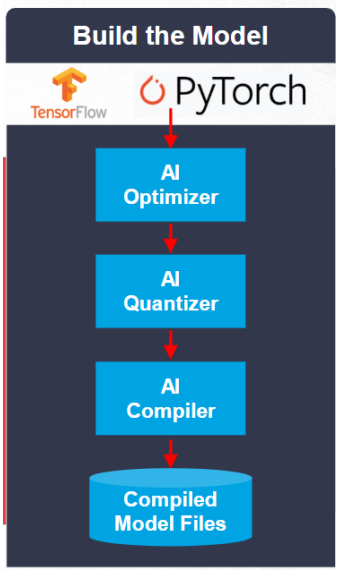
Build: Comprehensive Development Tool Suite

1. Dedicated Compilers: High Level Synthesis converts C/C++ code into a netlist and abstracts many of the underlying hardware architecture optimizations.
2. System Compile/Link: Links blocks of code and creates interconnections for data movement between memory hierarchies
3. Xilinx Runtime software: Platform/OS independent APIs managing:
    - Device Configuration
    - Memory and host-to-device data transfers
    - Accelerator execution
4. System-level simulation: Simulate with fast transaction-level or cycle-accuarte simulator
5. Performance analyzer: Optimize for partioning/performance

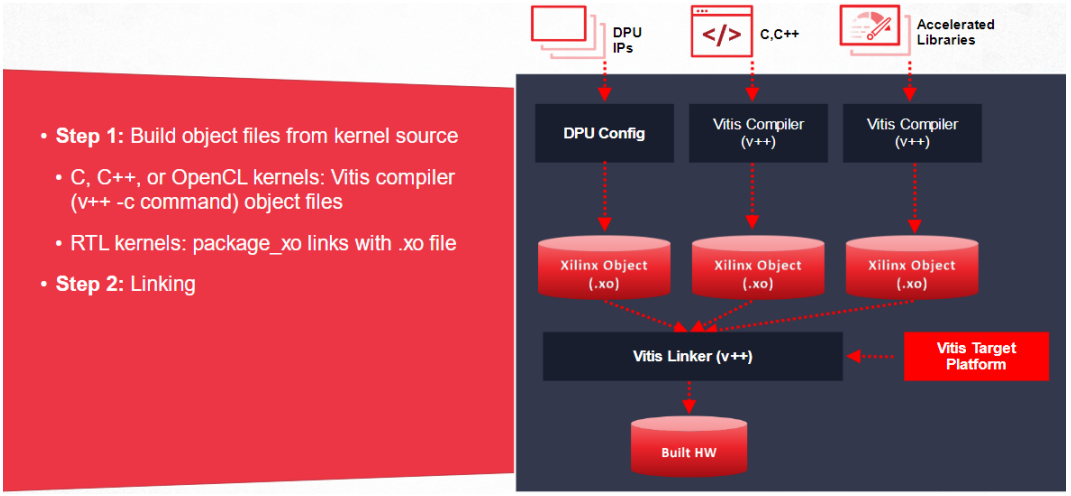


Vitis AI development Environment Flow

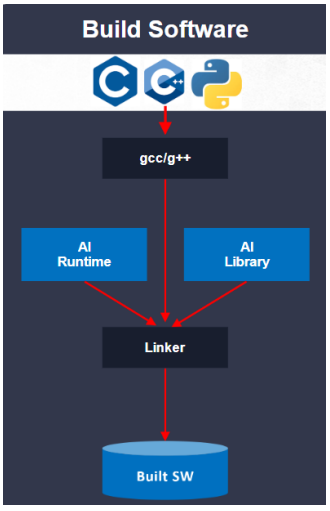
Step 1: Build the model



Step 2: Build the Hardware



Step 3: Build the software



Getting Started

1. Install the docker
2. Ensure Linux user is in the group docker
3. Clone the Vitis-AI repository:
   ```
   git clone --recurse-submodules
   https://github.com/Xilinx/Vitis-AI
   cd Vitis-AI
   ```
4. Run the docker container (running on CPU):
   ```
   ./docker_run.sh Xilinx/vitis-ai-
   cpu:latest
   ```
   Run the docker container (running on GPU):
   ```
   ./docker_run.sh Xilinx/vitis-ai-
   gpu:latest
   ```
5. Get started with examples