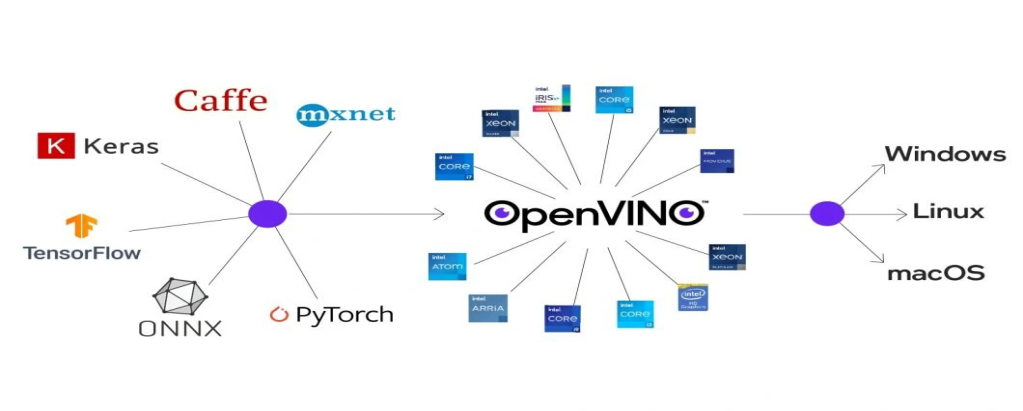
𝐖𝐡𝐚𝐭 𝐢𝐬 𝐎𝐩𝐞𝐧𝐕𝐈𝐍𝐎?

Open VINO, short for Open Visual Inference and Neural Network Optimization, is an open-source toolkit developed by Intel.

* It is designed to optimize and accelerate deep learning workloads for various edge devices, including CPUs, GPUs, FPGAs (Field-Programmable Gate Arrays), and other hardware accelerators.
* The primary goal of Open VINO is to provide a framework that enables developers to deploy deep learning models efficiently on a wide range of hardware platforms, making it well-suited for applications like computer vision, image and video analysis, and other AI-related tasks.
* Open VINO focuses on optimizing neural network inference with a write-once, deploy-anywhere approach for Intel hardware platforms.
* The toolkit is free for use under Apache License version 2.0 and has two versions:

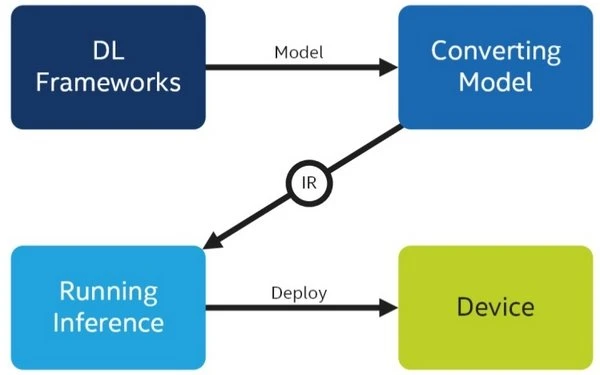
1. OpenVINO toolkit, which is supported by the open-source community. (<https://github.com/openvinotoolkit/openvino>)
2. Intel Distribution of OpenVINO toolkit, which is supported by Intel. (<https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html>)

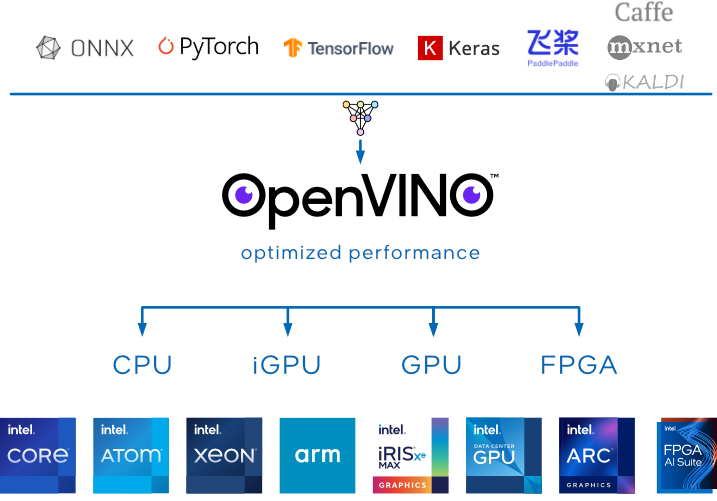
* Using the OpenVINO toolkit, software developers can select models and deploy pre-trained deep learning models (YOLO v3/v4/v5/v6/v7/v8, ResNet 50, etc.) through a high-level C++ Inference Engine API integrated with application logic.



Why Use OpenVINO?

* Deep Neural Networks (DNNs) have made considerable advances in many industrial domains in the past few years, bringing the accuracy of computer vision algorithms to a new level. However, deploying and producing such accurate and useful models requires adaptations for the hardware and computational methods.
* OpenVINO allows the optimization of DNN models for inference to be a streamlined, efficient process through the integration of various tools. The OpenVINO toolkit is based on the latest generations of Artificial Neural Networks (ANN), such as Convolutional Neural Networks (CNN) as well as recurrent and attention-based networks.
* The OpenVINO toolkit covers both computer vision and non-computer vision workloads across Intel hardware. It maximizes performance and accelerates application development.
* OpenVINO aims to accelerate AI workloads and speed up time to market using a library of predetermined functions as well as pre-optimized kernels. In addition, other computer vision tools such as OpenCV, OpenCL kernels, and more are included in toolkit.
* The 2023.0 version of [OpenVINO](https://www.linkedin.com/feed/hashtag/?keywords=openvino&highlightedUpdateUrns=urn%3Ali%3Aactivity%3A7090615584769355777) was designed to improve the developer journey by minimizing offline conversions, broadening model support, and advancing hardware optimizations.



****

**What Are the Benefits of OpenVINO?**

1. **Streamline Deep Learning Deployment**: Utilize Convolutional Neural Network (CNN)-based deep learning functions using one common API in addition to more than 30 pre-trained models and documented code samples. With more than 100 public and custom models, the OpenVINO toolkit streamlines deep learning innovation by providing one centralized method for implementing dozens of deep learning models.
2. **Accelerate Performance**: Expedite computer vision workloads by enabling simple execution methods across different Intel processors and accelerators such as CPU, GPU/Intel Processor Graphics, VPU (Intel AI Stick NCS2 with Myriad X), and FPGA.
3. **Extend and Customize**: OpenCL (Open Computing Language) Kernels and other tools offer an open, royalty-free standard way to add custom code pieces straight into the workload pipeline, customize deep learning model layers without the burden of framework overheads, and implement parallel programming of various accelerators.
4. **Innovate Artificial Intelligence**: The complete Deep Learning Deployment Toolkit within OpenVINO allows users to extend artificial intelligence within private applications and optimize artificial intelligence “all the way to the cloud” with processes such as the Model Optimizer, Intermediate Representation, nGraph Integration, and more.
5. **Official Model Zoo:** This provides many state-of-the-art, pre-trained models. All these models are already optimized for the OpenVINO toolkit, and provide fast inference straight out of the box. <https://docs.openvino.ai/archive/2019_R1/_docs_Pre_Trained_Models.html>
6. **ONNX Exportation:** The ability to export to an ONNX format, the Open standard for machine learning interoperability, is included in this add-on. This is an informal standard for NNs representation. Such optimized models can be converted to OpenVINO Intermediate Representation.

**What Can The OpenVINO Toolkit Be Used For?**

* The toolkit can Deploy computer vision inference on various hardware (more below)
* Import and optimize models from various frameworks such as PyTorch, TensorFlow, etc. (Post-training to accelerate inference)
* Run deep learning models outside of computer vision
* Perform “traditional” computer vision tasks (such as background subtraction)

The toolkit can not

* Training a machine learning model (although there are Training Extensions https://github.com/openvinotoolkit/training\_extensions)
* Run “traditional” machine learning outside of computer vision (such as Support Vector Machine), check out OpenCV
* Interpret the output of the model

**How OpenVINO Works**

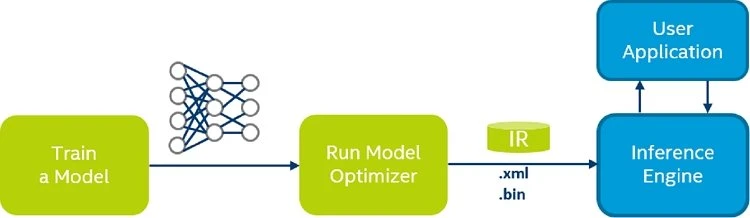
The OpenVINO workflow primarily consists of four main steps:

1. **Train:** A model is trained with code.

2. **Model Optimizer:** The model is fed to the Model Optimizer, whose objective is to optimize the model and generate an Intermediate Representation (.xml + .bin files) of the model. The models are optimized with techniques such as quantization, freezing, fusion, and more. In this step, pre-trained models are configured according to the framework chosen and then converted with a simple, single-line command. Users can choose from an array of pre-trained models in the OpenVINO Model Zoo, which contains models for every purpose, from object detection to text recognition to human pose estimation.

3. **Inference Engine:** The Intermediate Representation is fed to the Inference Engine. The inference engine’s job is to check for model compatibility based on the framework used to train the model as well as the hardware used (otherwise known as the environment). Frameworks supported by OpenVINO include TensorFlow, TensorFlow Lite, Caffe, MXNet, ONNX (PyTorch, Apple ML), and Kaldi.

4. **Deployment:** The application is deployed to devices.

****

The OpenVINO Toolkit represents neural network models with the help of two files:

* **An XML (.xml) file**—this file contains the neural network topology, more commonly known as the architecture.
* **A binary (.bin) file**—it contains the weights of the neural network model.

This representation is called the OpenVINO Intermediate Representation (IR).

Okay, so what’s there in an XML file?

* The XML file has different tags to represent the neural network operations and the data flow between them. For example, the <layer> tag is meant for operations like convolution or max-pooling.

Here’s a short snippet of one such XML file.

<net name="yolo-v2-tiny-ava-0001" version="10">

<layers>

<layer id="0" name="data" type="Parameter" version="opset1">

<data element\_type="f32" shape="1,3,416,416"/>

<output>

<port id="0" names="data:0" precision="FP32">

<dim>1</dim>

<dim>3</dim>

<dim>416</dim>

<dim>416</dim>

</port>

</output>

</layer>

...

<layer id="5" name="yolov2/darknet\_model/conv1/Conv2D/Transpose2101\_const" type="Const" version="opset1">

    <data element\_type="f32" offset="24" shape="16,3,3,3" size="1728"/>

    <output>

    <port id="0" names="yolov2/darknet\_model/conv1/W/read:0" precision="FP32">

    <dim>16</dim>

    <dim>3</dim>

    <dim>3</dim>

    <dim>3</dim>

   </port>

   </output>

</layer>

…

</meta\_data>

</net>

The above block shows a small part of the Tiny YOLOv2 XML file. One of the <layer> tags, as you can see, contains the convolution operation. Similarly, in the rest of the topology, the other <layer> tags may contain pooling or activation operations. The different sub-tags represent the data type, as well as the input and output dimensions.

* The XML file does not contain any model weights. It only has the topology for the corresponding binary (.bin) file that contains the model weights.