Asia's Largest

Cloud & AI

Conference 2023

17 - 18, November 2023
IIT Madras Research Park, Chennai

Anisha Udayakumar

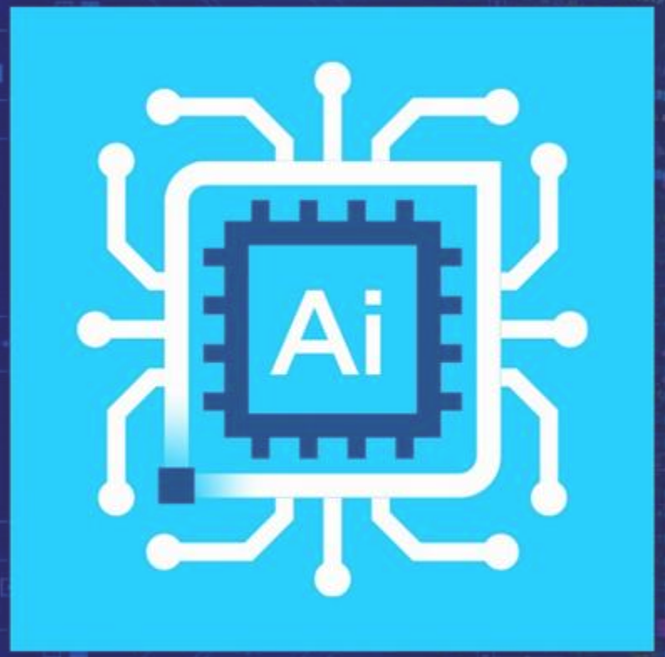
AI Software Evangelist - APJ

intel

From Idea to Performance Impact:

Supercharging Generative AI with the power of Optimization

What is Generative AI?



Generating and manipulating data at a large scale with increased flexibility

Compelling Creative Use Cases

azconf

Gaming Experiences

Global Generative AI Market Size*

$1.3T by 2032

42% CAGR over 10 years

*Source: Bloomberg Intelligence

Room Design

Novel Illustration

Fashion & E-Commerce

Web Design

The Challenge

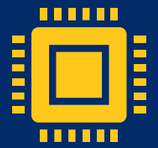
Generative AI: Pain Points

Large model size

Large memory footprint

Slow inference speed

Difficulty training + optimizing



No flexibility to run workloads on different HW

Having Your Model Trained is Only 1/2 the Challenge!

Deployment is hard...
and we know, because
we've done a lot of it!

How Do I Deploy My Neural Network?

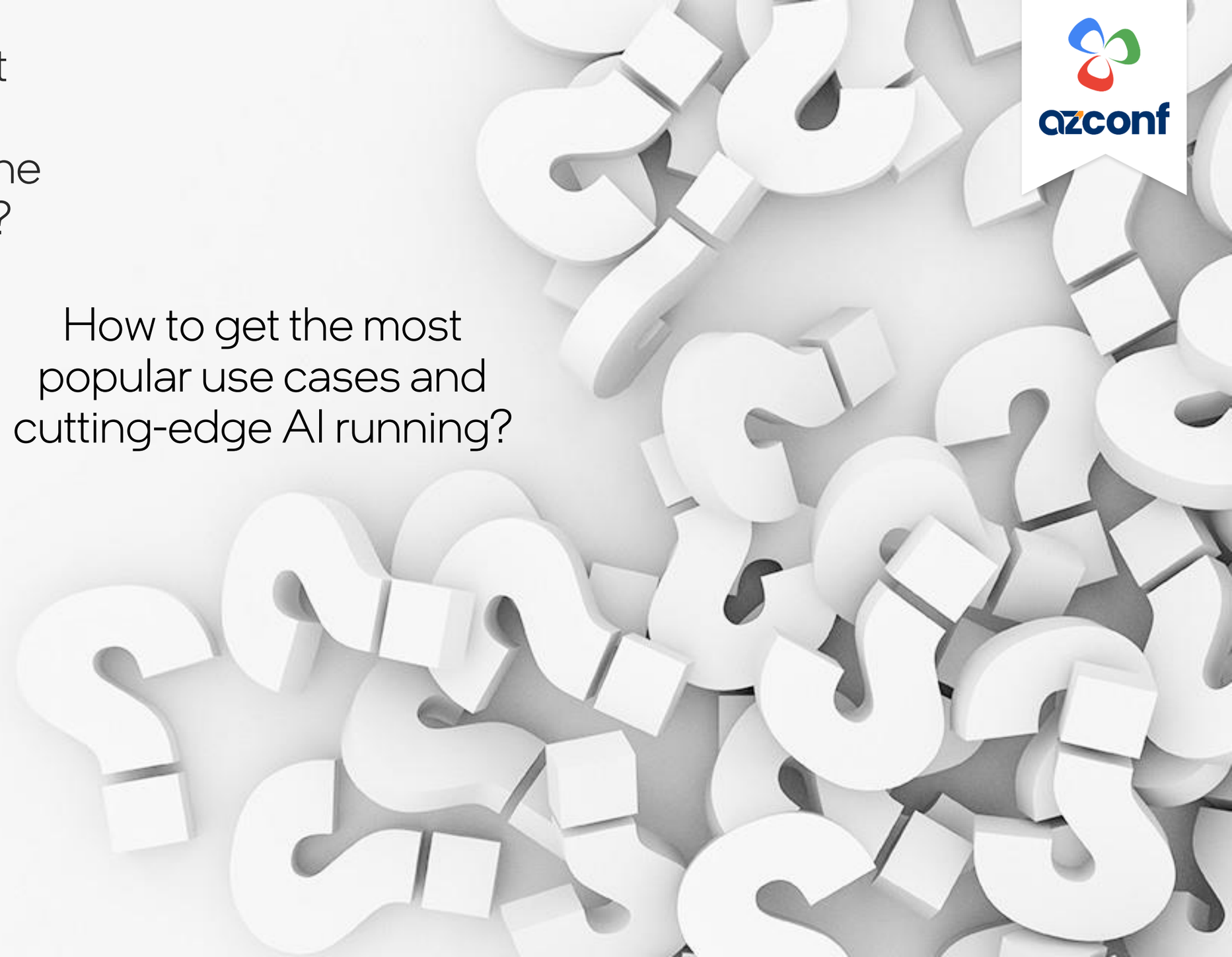
Should we start with
CPU, GPUs?


azconf
How about XPU or 'all of the above' since Intel now has CPU, GPU, and NPU all working under the same one roof.
OpenVINO

How to get the best AI application performance out of the available hardware?

How to get the most popular use cases and cutting-edge AI running?

How to maintain code across different architectures?




The Solution

Accelerate Gen AI with OpenVINO™ OpenVINO™ Toolkit

azconf

OpenVINO™

Visual + (NLP, Audio, Generative AI, LLM, ...)

Inference

Neural Network

Optimization

OpenVINO website

OpenVINO™

DEVELOPER JOURNEY

1 MODEL 2 OPTIMIZE 3 DEPLOY

PyTorch
TensorFlow
Keras
TensorFlow Lite
ONNX
Caffe
PaddlePaddle

azconf

OpenVINO™

Optimized Performance

CPU

intel ATOM
intel CORE
intel XEON
arm
NEW IN 2023

GPU

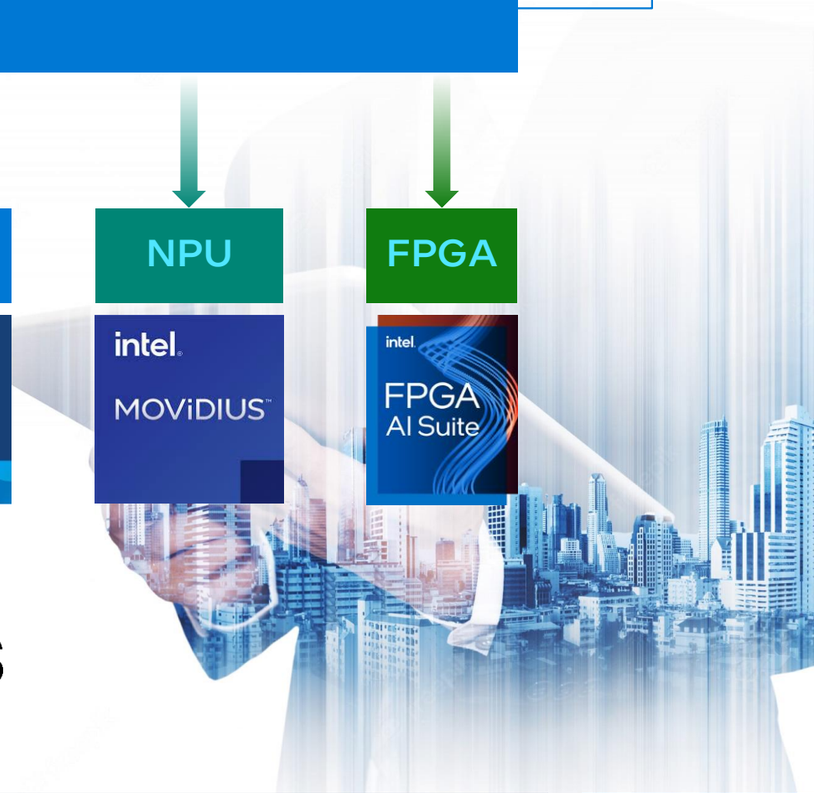
intel iRIS Xe GRAPHICS
intel ARC GRAPHICS
intel DATA CENTER GPU FLEX SERIES

NPU

intel MOVIDIUS

FPGA

intel FPGA AI Suite



Windows
Linux
macOS

OpenVINO™

What's New in the 2023.0 Release?

azconf

Model

New TensorFlow frontend

New PyTorch frontend (experimental)

Conda Forge installation for C++ developers

ARM processors support

Python 3.11 support

Optimize

Broader model support, e.g., StableDiffusion, SAM, Whisper

Dynamic shape on GPU

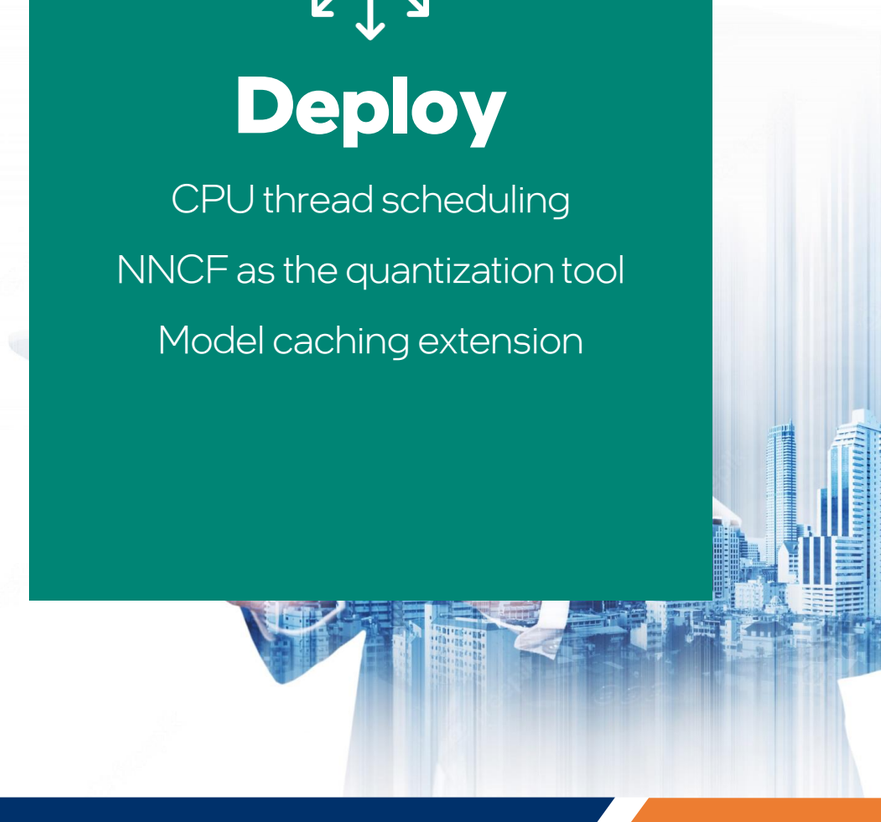
Default inference precision

Deploy

CPU thread scheduling

NNCF as the quantization tool

Model caching extension



Installation

ubuntu

Red Hat

docker

ANACONDA®

CONDA-FORGE

CONAN
C/C++ Package Manager

www.openvino.ai

Installation

```
pip install openvino
```

ubuntu

Red Hat

docker

ANACONDA

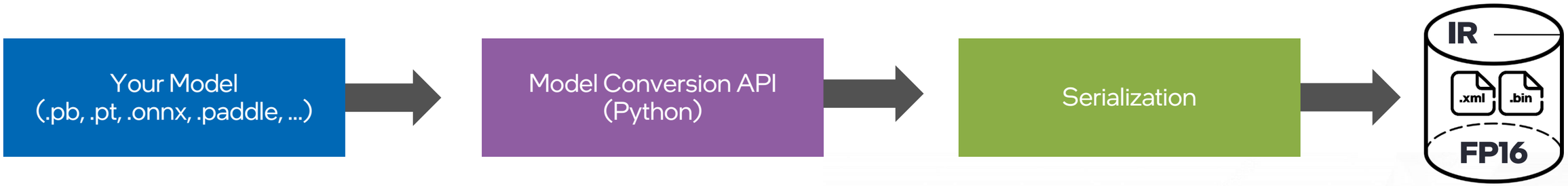
CONDA-FORGE

CONAN
C/C++ Package Manager

www.openvino.ai

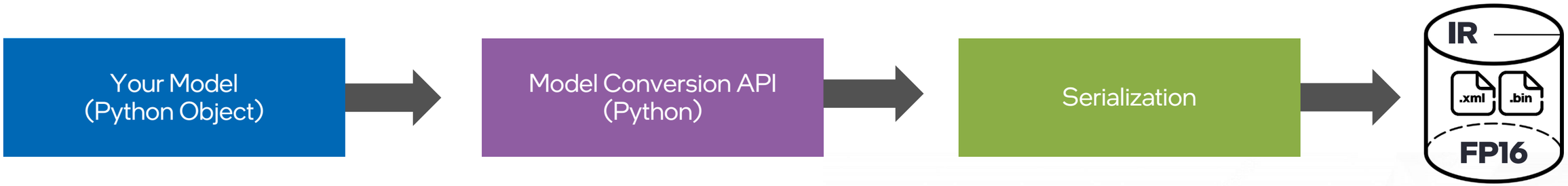
OpenVINO™ Model Converter

OpenVINO™ Model Converter



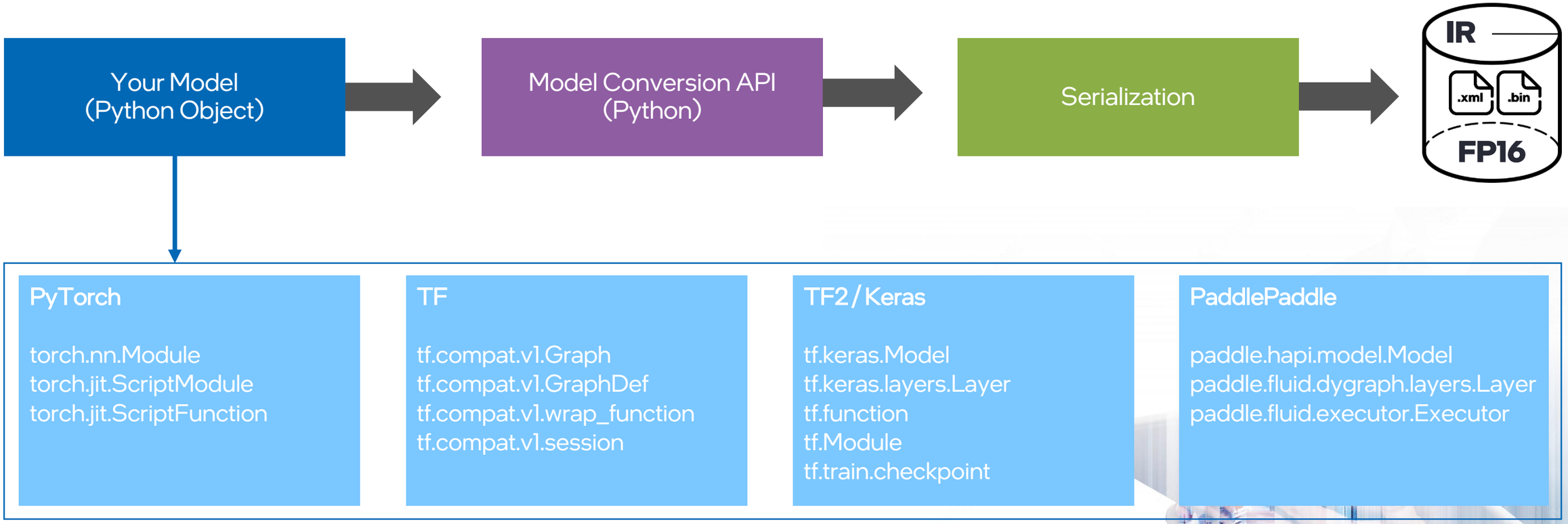

```
ov_model = ov.convert_model("model.onnx",
                            input={"input1": [1,3,224,224]})

ov.save_model(ov_model, "converted_model.xml")
```

OpenVINO™ Model Converter




```
pytorch_model = torchvision.models.resnet50(pretrained=True)

ov_model = ov.convert_model(pytorch_model,
                            input={"input1": [1,3,224,224]})

ov.save_model(ov_model, "converted_model.xml")
```

OpenVINO™ Model Converter



Before Model Conversion

After Model Conversion

OpenVINO™ Runtime

```python
from openvino import runtime as ov

img = load_img()

core = ov.Core()

model = core.read_model(model="model.xml")
compiled_model = core.compile_model(model=model, device_name="CPU")

output_layer = compiled_model.outputs[0]

result = compiled_model(img)[output_layer]
```

OpenVINO™ Runtime

PyTorch Automatic Conversion

```
from openvino import runtime as ov

img = load_img()

core = ov.Core()
# PyTorch frontend
model = core.read_model(model="model.pt")
compiled_model = core.compile_model(model=model, device_name="CPU")

output_layer = compiled_model.outputs[0]

result = compiled_model(img)[output_layer]
```

OpenVINO™ Runtime

TensorFlow Automatic Conversion

```
from openvino import runtime as ov

img = load_img()

core = ov.Core()
# TensorFlow frontend
model = core.read_model(model="model.pb")
compiled_model = core.compile_model(model=model, device_name="CPU")

output_layer = compiled_model.outputs[0]

result = compiled_model(img)[output_layer]
```

OpenVINO™ Runtime

TensorFlow Lite Automatic Conversion

```
from openvino import runtime as ov

img = load_img()

core = ov.Core()
# TensorFlow Lite frontend
model = core.read_model(model="model.tflite")
compiled_model = core.compile_model(model=model, device_name="CPU")

output_layer = compiled_model.outputs[0]

result = compiled_model(img)[output_layer]
```

OpenVINO™ Runtime

ONNX Automatic Conversion

```
from openvino import runtime as ov

img = load_img()

core = ov.Core()
# ONNX frontend
model = core.read_model(model="model.onnx")
compiled_model = core.compile_model(model=model, device_name="CPU")

output_layer = compiled_model.outputs[0]

result = compiled_model(img)[output_layer]
```

OpenVINO™ Runtime

PaddlePaddle Automatic Conversion

```python
from openvino import runtime as ov

img = load_img()

core = ov.Core()
# Paddlepaddle frontend
model = core.read_model(model="model.pdmodel")
compiled_model = core.compile_model(model=model, device_name="CPU")

output_layer = compiled_model.outputs[0]

result = compiled_model(img)[output_layer]
```

Supported Devices

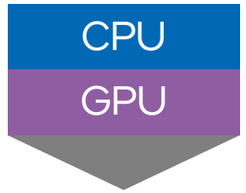
```
compiled_model = core.compile_model(model=model, device_name="CPU")
```

Supported Devices

CPU

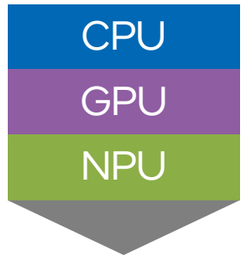
```
compiled_model = core.compile_model(model=model, device_name="CPU")
```

Supported Devices



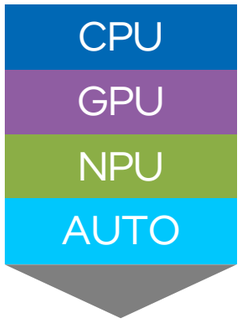

```
compiled_model = core.compile_model(model=model, device_name="GPU")
```

Supported Devices

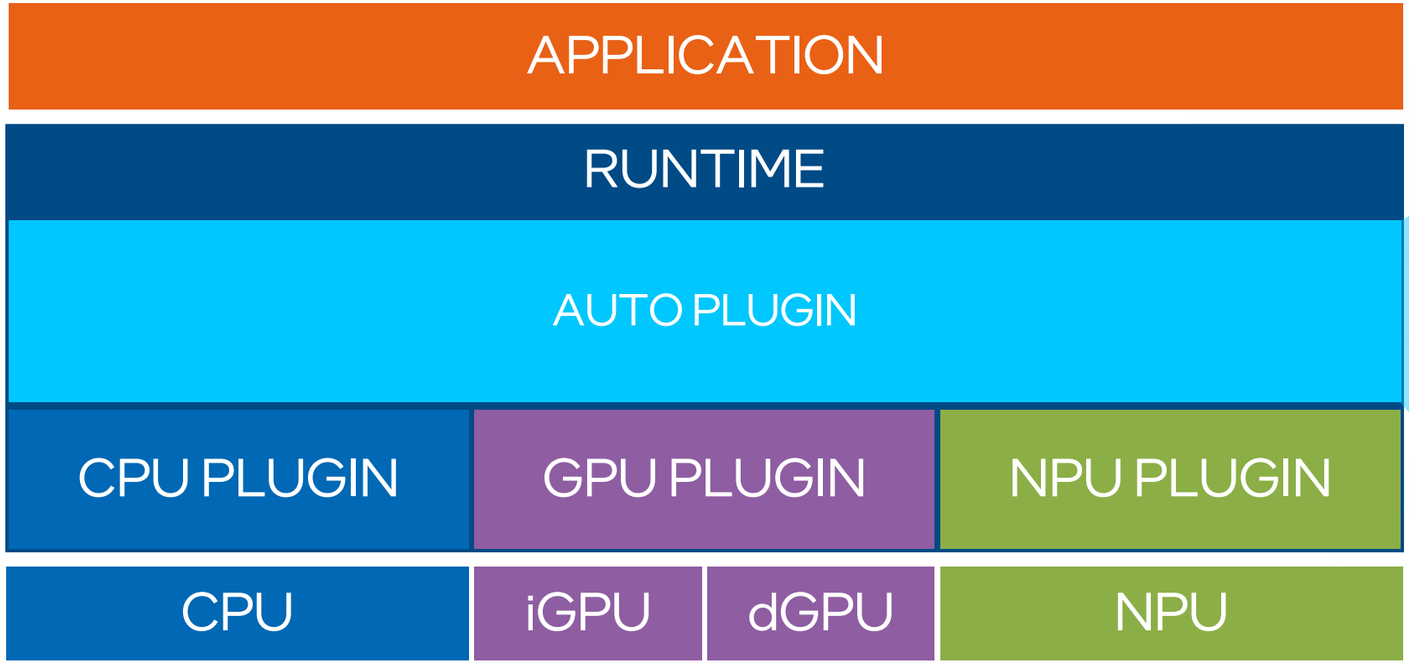



```
compiled_model = core.compile_model(model=model, device_name="NPU")
```

AUTO Device




```
compiled_model = core.compile_model(model=model, device_name="AUTO")
```

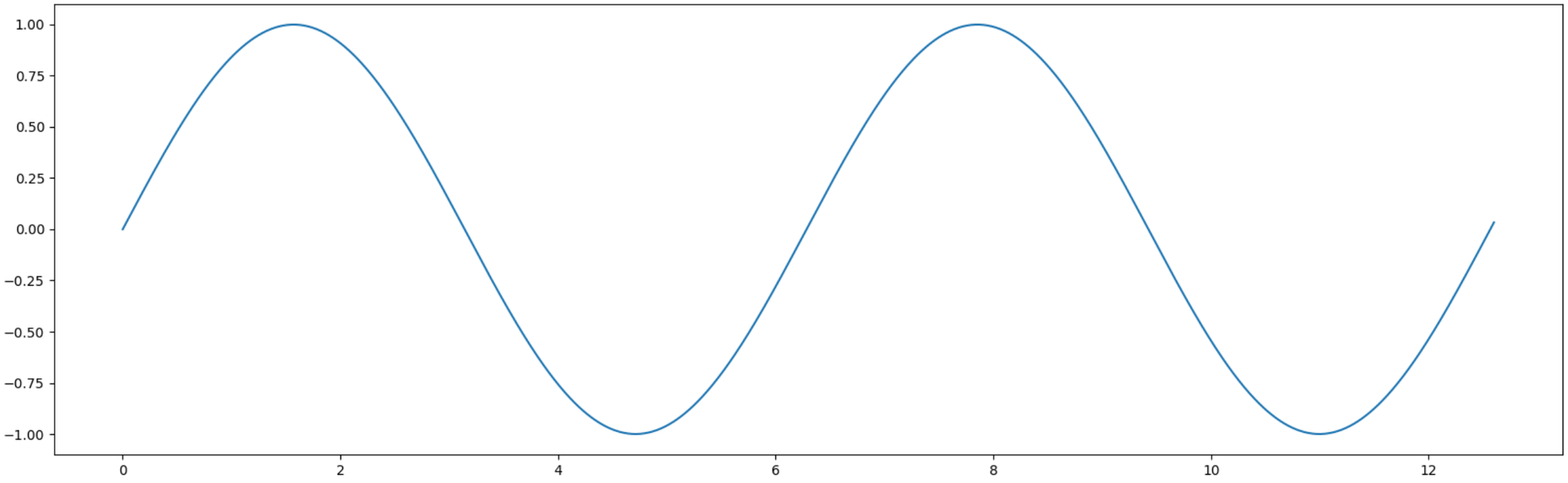


AUTO chooses a device best-suited for the task

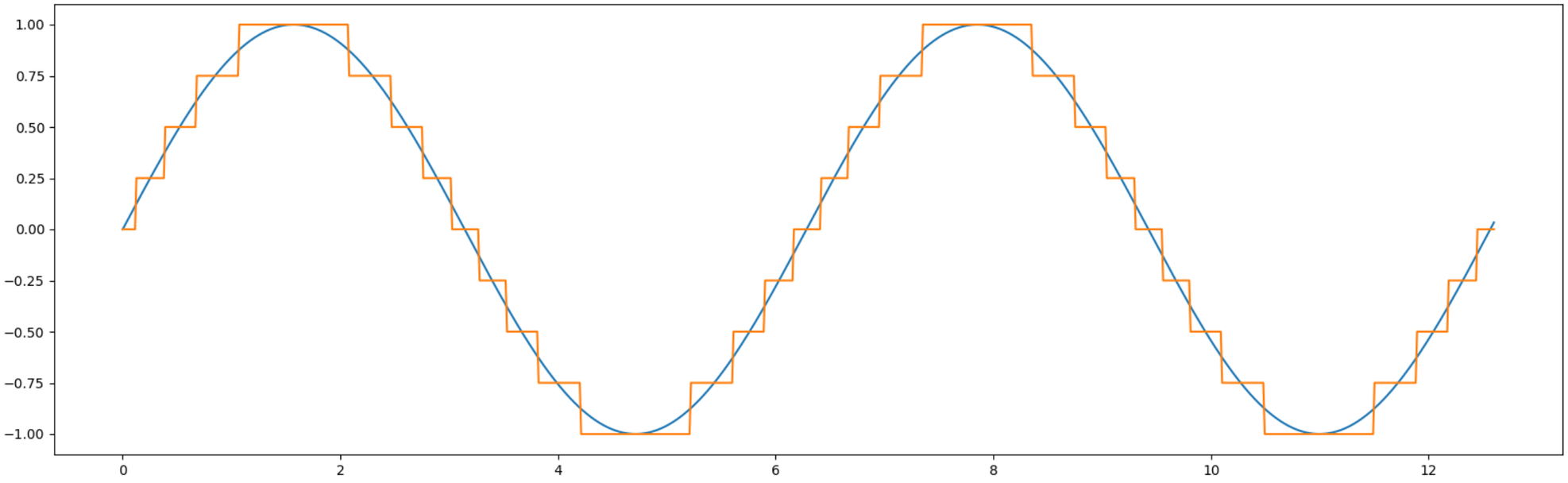
AUTO configures the device based on hints

AUTO handles the exec logic on multiple devices

Quantization

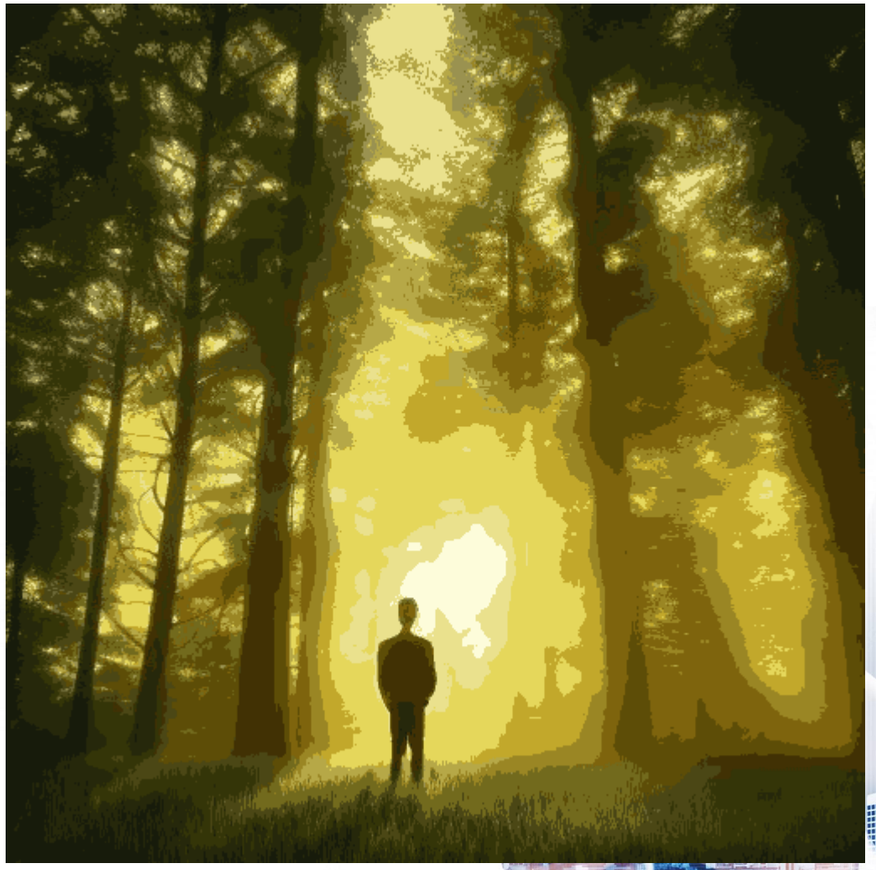


Quantization



Color Quantization

16,777,216 colors = “FP32”

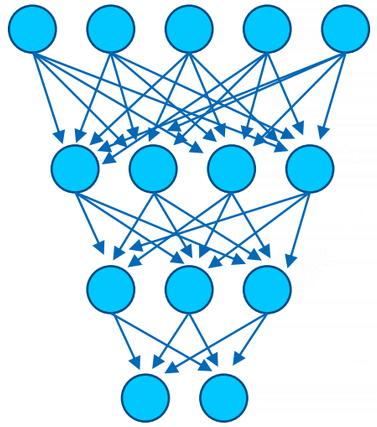


16 colors = “INT8”

Neural Network Compression Framework (NNCF)

TensorFlow PyTorch

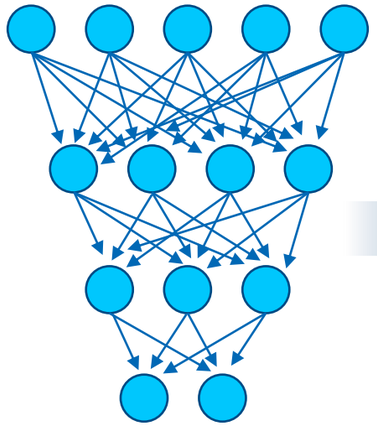
OpenVINO ONNX



Neural Network Compression Framework (NNCF)

TensorFlow PyTorch

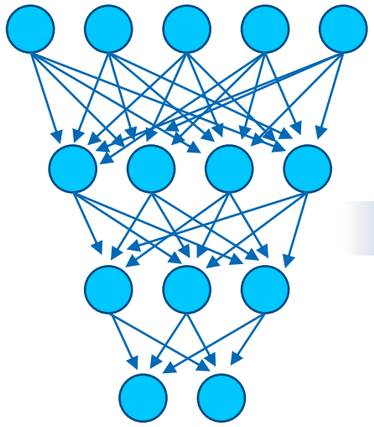
OpenVINO ONNX



- Post-Training Quantization
- Accuracy-Control Quantization
- Quantization-Aware Training
- Weight Compression
- Filter pruning, Binarization, Sparsity, ...

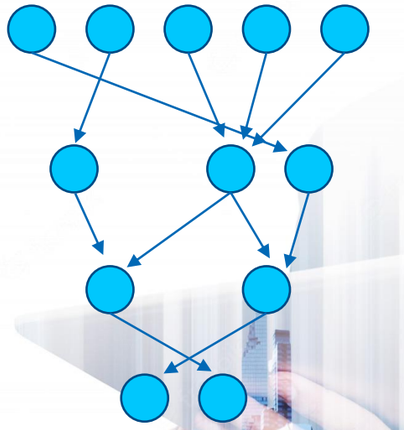
Neural Network Compression Framework (NNCF)

TensorFlow PyTorch
OpenVINO ONNX



- Post-Training Quantization
- Accuracy-Control Quantization
- Quantization-Aware Training
- Weight Compression
- Filter pruning, Binarization, Sparsity, ...

TensorFlow PyTorch
OpenVINO ONNX




```
pip install nncf
```

Weight Compression (NNCF)

```
from nncf import compress_weights
compressed_model = compress_weights(model)
```

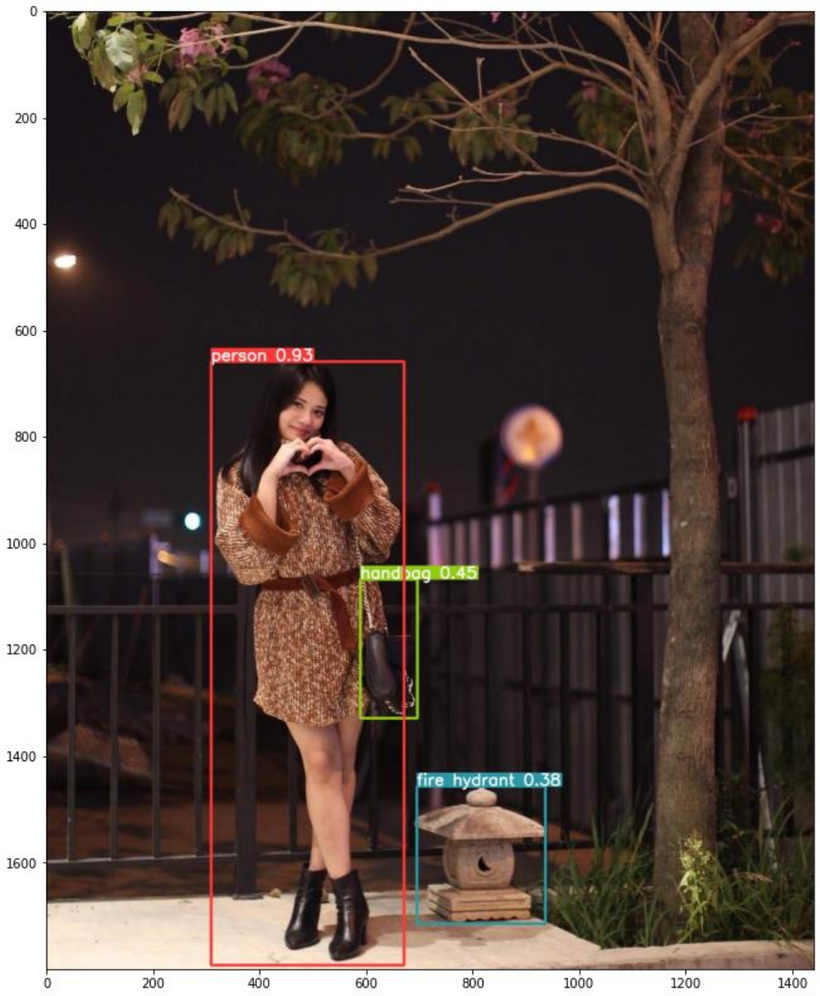
Model	Size Reduction (GB) FP32 → INT8
llama-2-7b-chat	25 → 6
open-llama-3b	13 → 3
dolly-v2-12b	44 → 11
gpt-neox-20b	77 → 19
llama-7b	25 → 6
gpt-j-6b	23 → 6

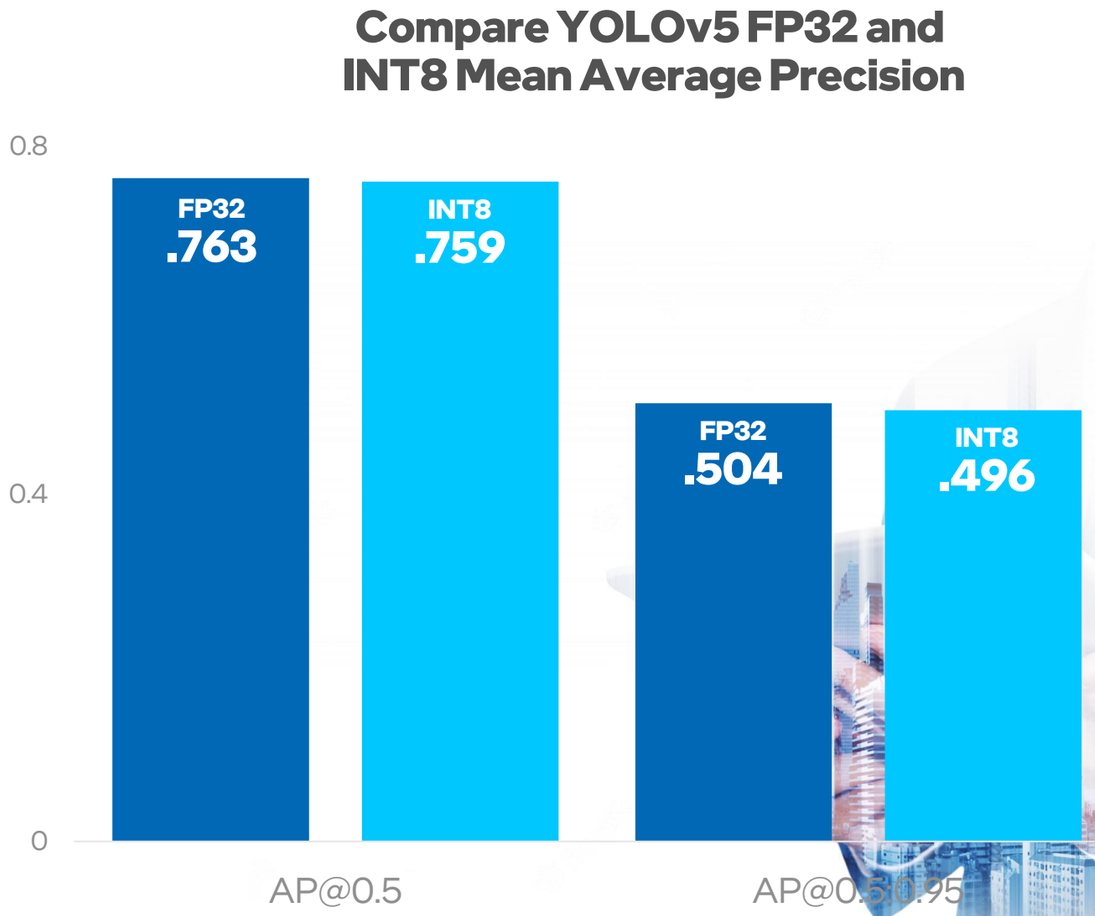
Model size comparison before and after weight compression

Post-Training Quantization (NNCF)

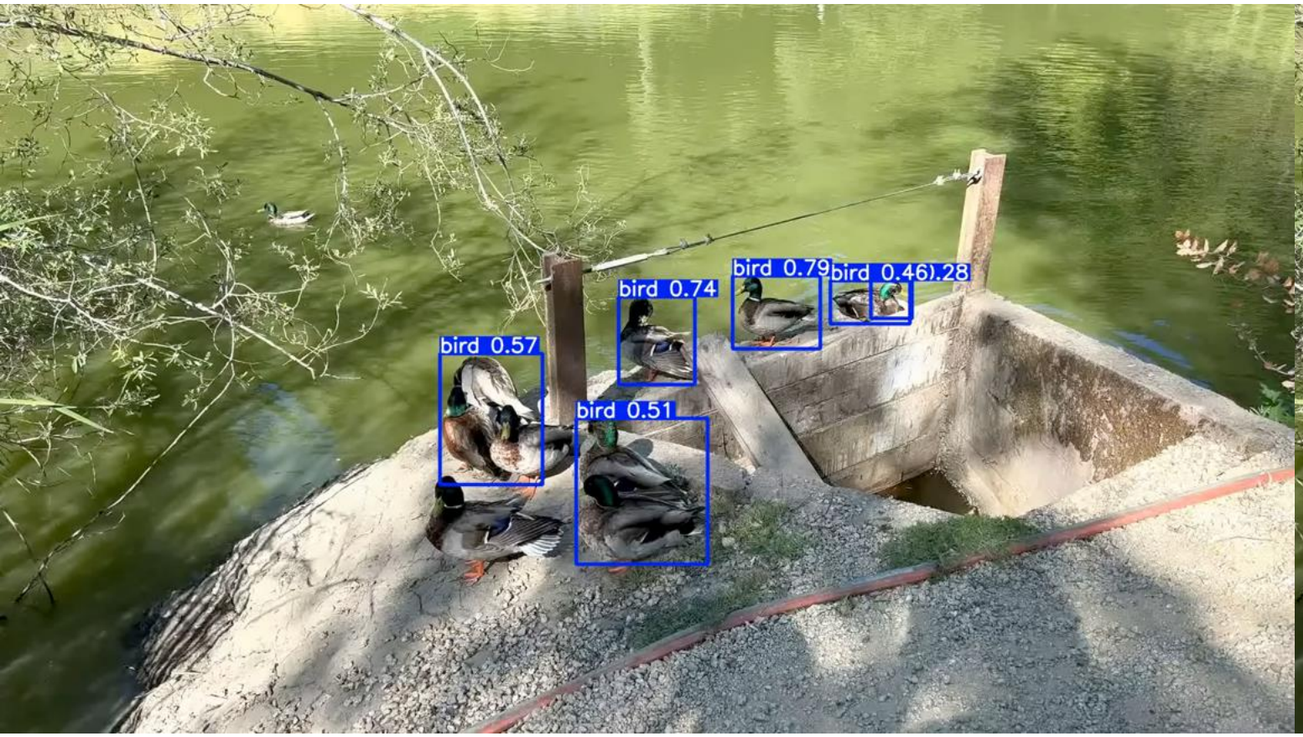
```
from openvino import runtime as ov
import nncf

core = ov.Core()
model = core.read_model("model.xml")
data_loader = get_data_loader()

# Remove labels from data item and prepare input data (~100-500 samples)
def transform_fn(data_item):
    input_tensor = preprocess(data_item)['img'].numpy()
    return input_tensor

calibration_dataset = nncf.Dataset(data_loader, transform_fn)
quantized_model = nncf.quantize(model, calibration_dataset,
                                preset=nncf.QuantizationPreset.PERFORMANCE)
```

Post-Training Quantization (NNCF)

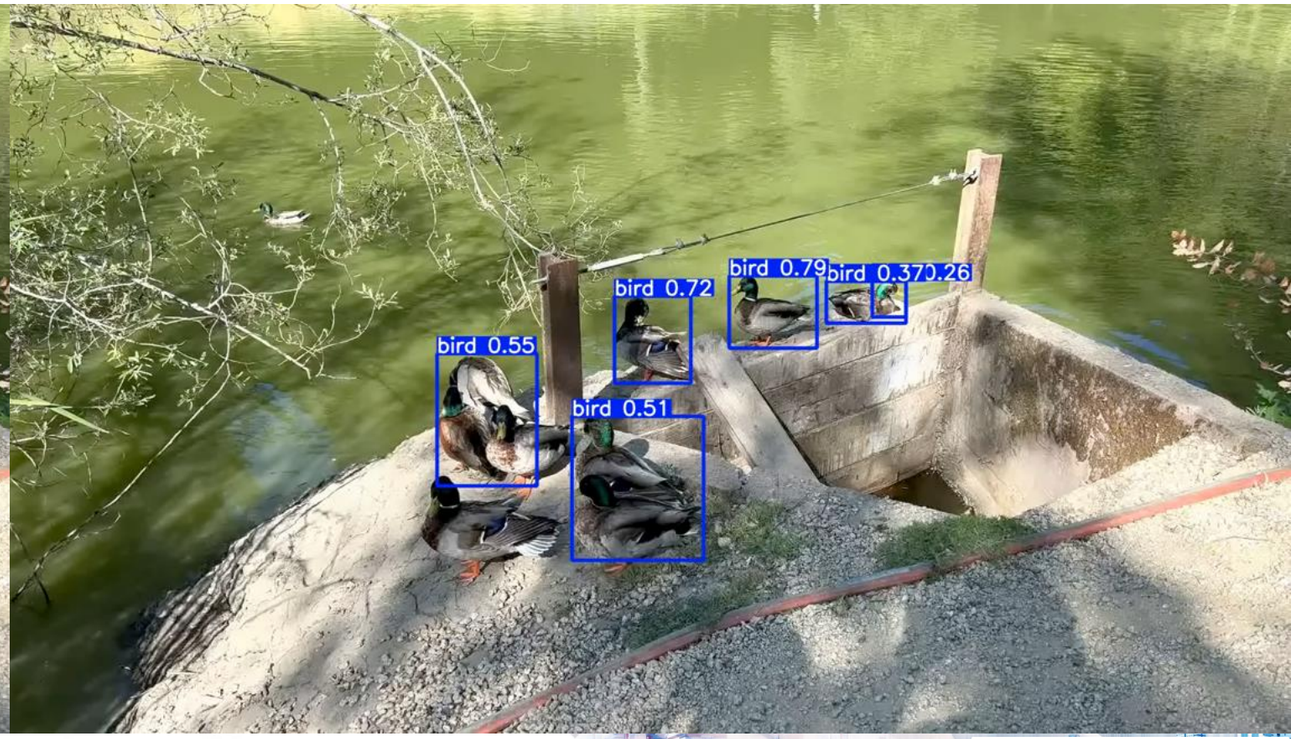




Quantization Results Comparison (YOLOv5)



FP32



INT8

Quantization Performance Comparison (OpenPose)

16.6 MB

4.7 MB

Optimization Journey

Low Latency

AUTO Device
+
“Latency”
+
MO, NNCF

Optimization Journey

azconf

Low Latency	High Throughput
AUTO Device + **"Latency"** + **MO, NNCF**	**AUTO Device** + **Async Inference** + **"Throughput"** + **MO, NNCF**

Optimization Journey

azconf

Low Latency	High Throughput	High Throughput
AUTO Device + **"Latency"** + **MO, NNCF**	**AUTO Device** + **Async Inference** + **"Throughput"** + **MO, NNCF**	**AUTO Device** + **"Cumulative Throughput"** + **Auto Batching** + **MO, NNCF**

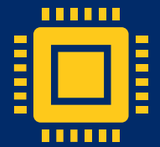
Generative AI: Pain Points

Large model size

Large memory footprint

Slow inference speed

Difficulty training + optimizing



No flexibility to run workloads on different HW

GenAI takes time and resources

Solution

Use powerful and expensive machines?

Use an edge device or my laptop?

Use both?

Accelerate Generative AI with OpenVINO™

azconf

Strategy optimizing

Reduce model size

Reduce memory footprint

Faster inference speed

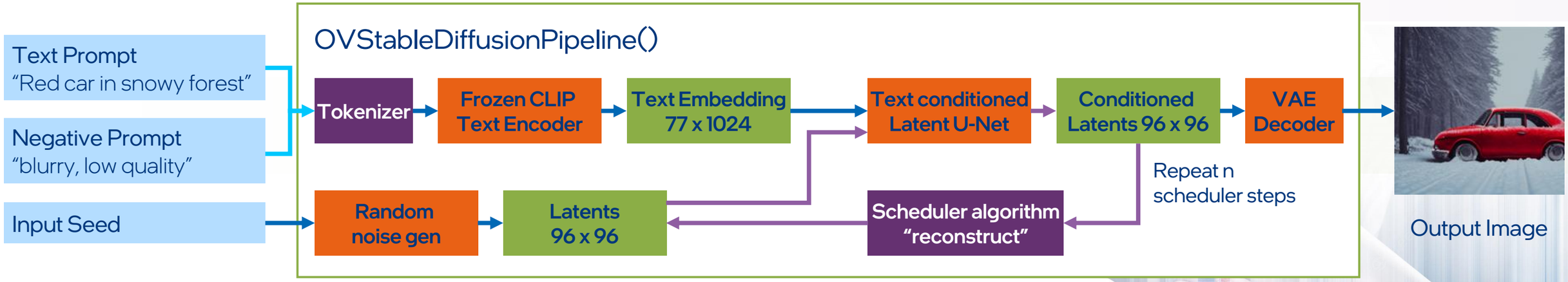
Flexibility to run workloads on CPUs and Intel GPUs

Strategy
optimizing

azconf

SD Pipeline

Inference Pipeline



Reduce model size

8.0K	stabilityai_cpu/feature_extractor
1.3G	stabilityai_cpu/text_encoder
8.0K	stabilityai_cpu/scheduler
1.6M	stabilityai_cpu/tokenizer
3.3G	stabilityai_cpu/unet
320	stabilityai_cpu/vae
M	stabilityai_cpu/
4.9G	

FP32 Native Model

8.0K	openvino_ir/feature_extractor
1.3G	openvino_ir /text_encoder
131M	openvino_ir /vae_encoder
8.0K	openvino_ir /scheduler
1.6M	openvino_ir /tokenizer
3.3G	openvino_ir /unet
190M	openvino_ir /vae_decoder
4.9G	openvino_ir /

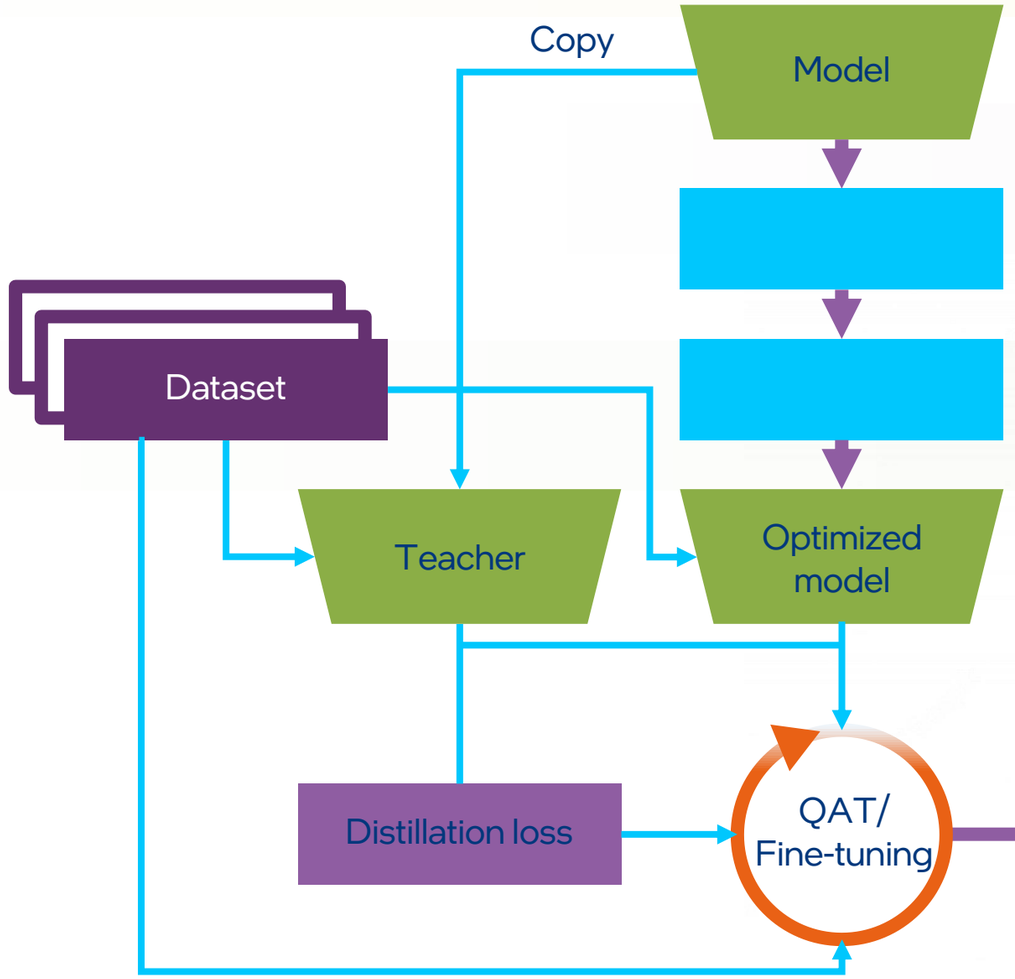
FP32 OpenVINO™ Model

8.0K	modelSD21_dGPU_0V/feature_extractor
652	modelSD21_dGPU_0V/text_encoder
M	modelSD21_dGPU_0V/scheduler
8.0K	modelSD21_dGPU_0V/tokenizer
1.6M	modelSD21_dGPU_0V/unet
1.7G	modelSD21_dGPU_0V/vae_decoder
96M	modelSD21_dGPU_0V/
2.4G	

FP16 OpenVINO™ model

Quantization-Aware Training (NNCF) for Stable Diffusion

Reduce memory footprint



Heavy Memory Load

Reduced memory footprint with OpenVINO™ optimizations

Read the blog

Ref: https://huggingface.co/blog/train-optimize-sd-intel

Fast inference speed

```
from openvino.tools.mo import convert_model
from openvino.runtime import serialize

ov_model = convert_model (“model.onnx”)
serialize(model=ov_model, xml_path=“model.xml”)
```

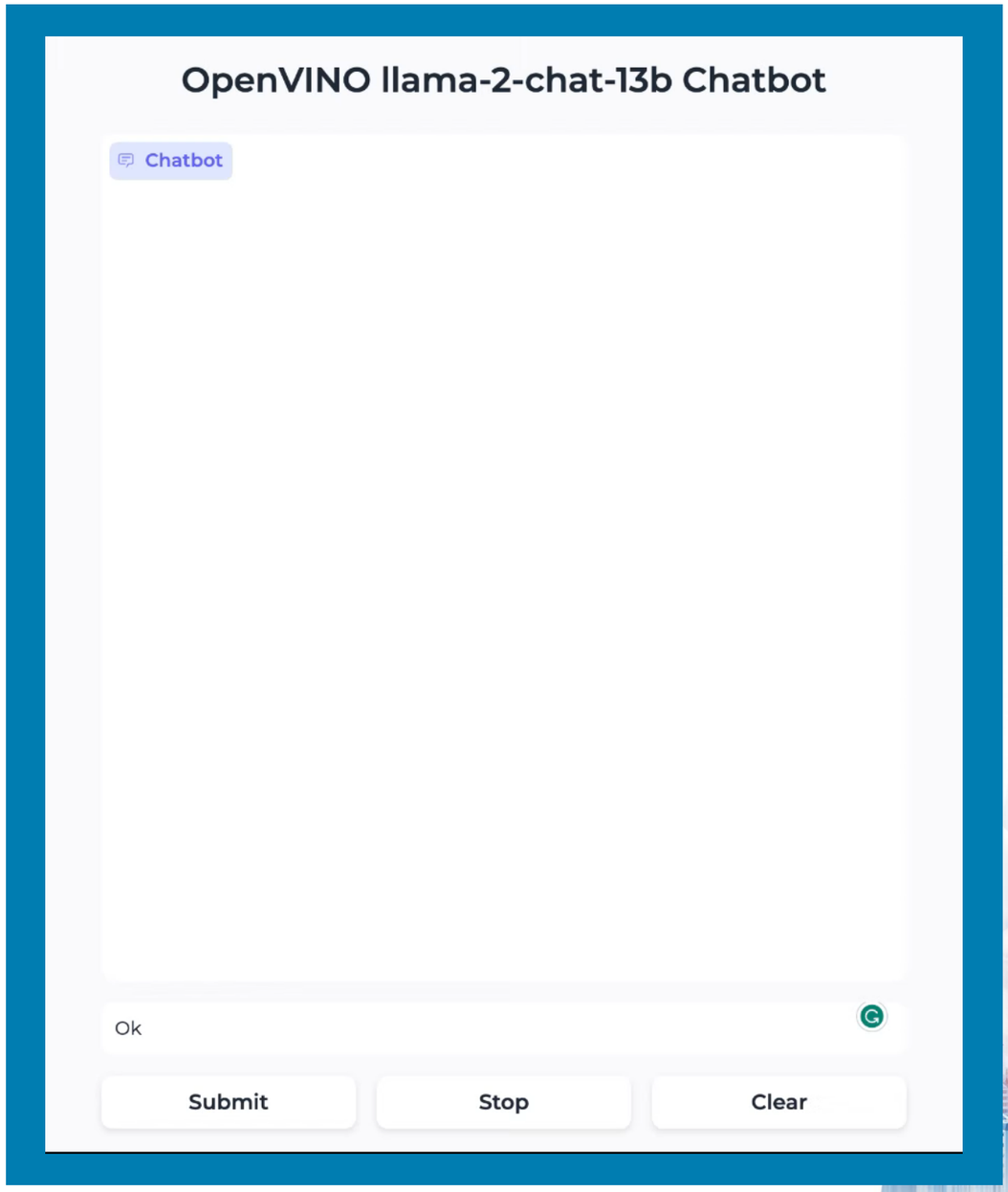
FP16 conversion on the fly for GPU devices

Flexibility to run workloads on CPUs and Intel GPUs

```
from openvino.runtime import core
core = Core()
text_enc  = core.compile_model (TEXT_ENCODER_OV_PATH, ‘GPU’)
unet_model = core.compile_model (UNET_OV_PATH, ‘GPU’)
vae_decoder = core.compile_model (VAE_DECODER_OV_PATH, ‘GPU’)
```

Let’s Run the Demo!

Create LLM-powered Chatbot using OpenVINO

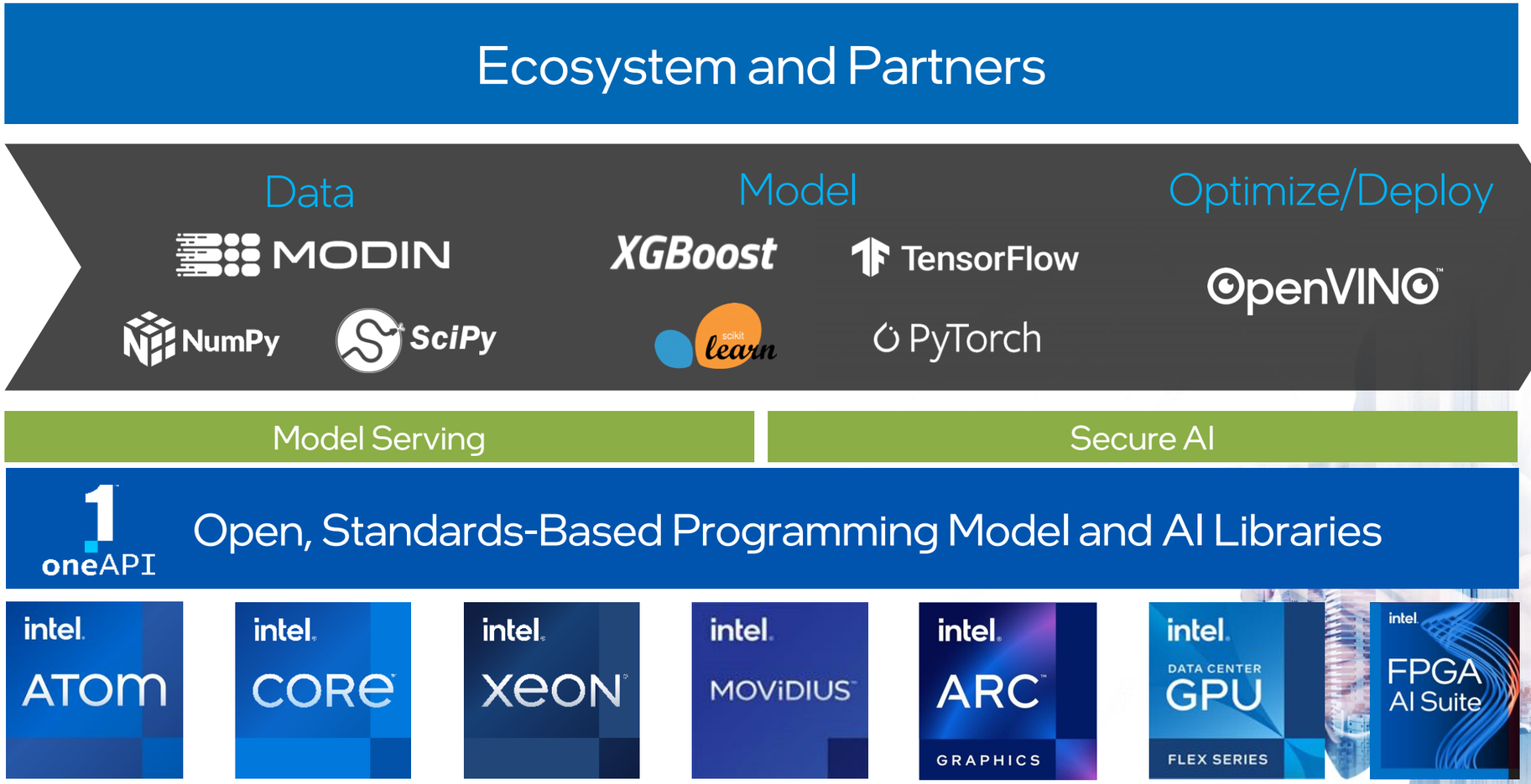


Software + Hardware

Training
50%

Software + Hardware
Optimization Makes Up The Other Half!

Intel AI Software Portfolio



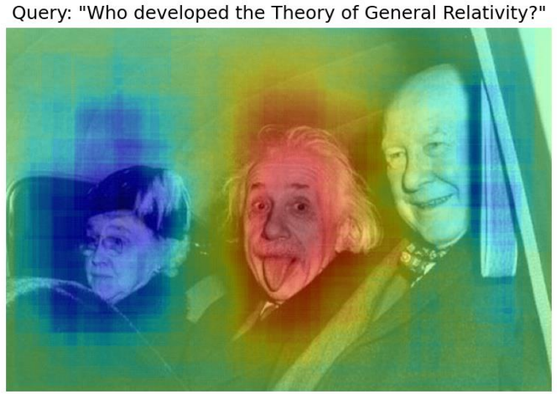
OpenVINO™ Ecosystem Adoption

azconf

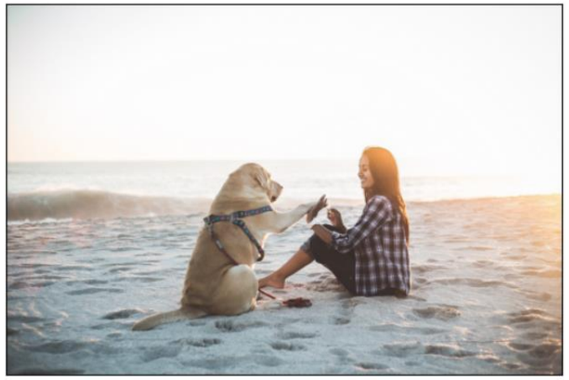
OpenVINO

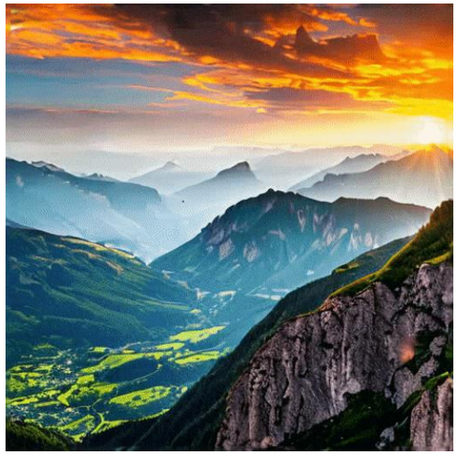
intel

Language-Visual Saliency with CLIP



Visual QA with BLIP





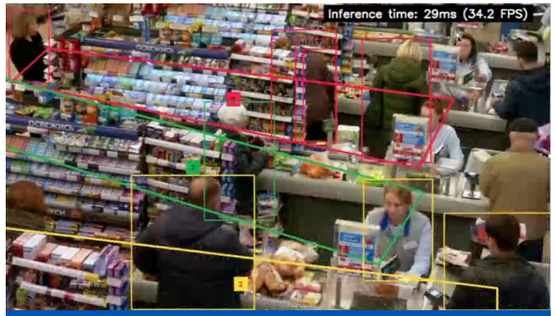
Infinite Zoom Stable Diffusion

'a photo of a blue T-shirt with yellow text "OPENVINO"'

AI Trends with OpenVINO

Build real-world AI solutions with Edge AI Reference Kits

Smart Meter Scanning



Intelligent Queue Management

Defect Detection with Anomalib

Explore more Edge AI Reference Kits


azconf
Try It Yourself
openvino.ai
OpenVINO
intel

THANK YOU!

azconf

Connect With Us

Anisha Udayakumar

Anisha.Udayakumar@intel.com

intel


intel
azconf

Q & A

Notices and Disclaimers

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details.

Intel technologies may require enabled hardware, software or service activation.

Your costs and results may vary.

Intel is committed to respecting human rights and avoiding complicity in human rights abuses. See Intel's Global Human Rights Principles. Intel's products and software are intended only to be used in applications that do not cause or contribute to a violation of an internationally recognized human right.

Platform Configurations for Performance Benchmarks

Configuration	Intel® Xeon® Gold 5218T	Intel® Core™ i9-12900HK	Intel® Core™ i5-10500TE	Intel® Core™ i3-8100
Motherboard	Intel® Server Board S2600STB	Intel Corporation internal/ Reference Validation Platform	GIGABYTE* Z490 AORUS PRO AX	GIGABYTE* Z390 UD
CPU	Intel® Xeon® Gold 5218T CPU @ 2.10GHz	Intel® Core™ i9-12900HK	Intel® Core™ i5-10500TE CPU @ 2.30GHz	Intel® Core™ i3-8100 CPU @ 3.60GHz
Hyper Threading	ON	ON	ON	OFF
Turbo Setting	ON	ON	ON	OFF
Memory	12 x 32 GB DDR4 2666MHz	4 x 8 GB DDR4 4800MHz	2 x 16 GB DDR4 2666MHz	4 x 8 GB DDR4 2400MHz
Operating System	Ubuntu* 20.04.3 LTS	Ubuntu* 20.04.3 LTS	Ubuntu* 20.04.3 LTS	Ubuntu* 20.04.3 LTS
Kernel Version	5.3.0-24-generic	5.15.0.-1003-intel-iotg	5.3.0-24-generic	5.3.0-24-generic
BIOS Vendor	Intel Corporation	Intel Corporation	American Megatrends Inc.*	American Megatrends Inc.*
BIOS Version	SE5C620.86B.02.01.0013.121520200651	ADLPFWI1.R00.2411.A02.2110081023	F21	F8
BIOS Release	15-Dec-20	8-Oct-21	23-Novl-21	24-May-19
BIOS Settings	Select optimized default settings, change power policy to "performance", save & exit	Default Settings	Select optimized default settings, set OS type to “other”, save and exit	Select optimized default settings, set OS type to “other”, save and exit
Batch size	Auto-batching	Auto-batching	Auto-batching	Auto-batching
Precision	FP32	FP32	FP32	FP32
Number of concurrent inference requests	Automatic	Automatic	Automatic	Automatic
Test Date	13-Sep-22	13-Sep-22	13-Sep-22	13-Sep-22
Rated maximum TDP/socket in Watt	28	45	35	65
GCC	gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0	gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0	gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0	gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
CMake	cmake version 3.14.1	cmake version 3.14.1	cmake version 3.14.1	cmake version 3.14.1
OpenVINO™ Version	2022.2	2022.2	2022.2	2022.2
Model Optimizer	2022.2.0-000-af16ea1d79a-HEAD	2022.2.0-000-af16ea1d79a-HEAD	2022.2.0-000-af16ea1d79a-HEAD	2022.2.0-000-af16ea1d79a-HEAD
Inference Engine	2022.2.0-000-af16ea1d79a-HEAD	2022.2.0-000-af16ea1d79a-HEAD	2022.2.0-000-af16ea1d79a-HEAD	2022.2.0-000-af16ea1d79a-HEAD