

Bring AI everywhere with OpenVINO™: enabling developers to quickly optimize, deploy, and scale AI applications across hardware device types with cutting-edge compression features and advanced performance capabilities.

What is OpenVINO™ ?

OpenVINO is an open-source toolkit for optimizing and deploying deep learning models. Deploy AI across devices (from PC to cloud) with automatic acceleration!

[Documentation](#)

[Get started](#)

[Blog](#)

[Examples](#)

Use OpenVINO with...

 PyTorch

 TensorFlow

 Hugging Face

 ONNX

and more

Build, Optimize, Deploy

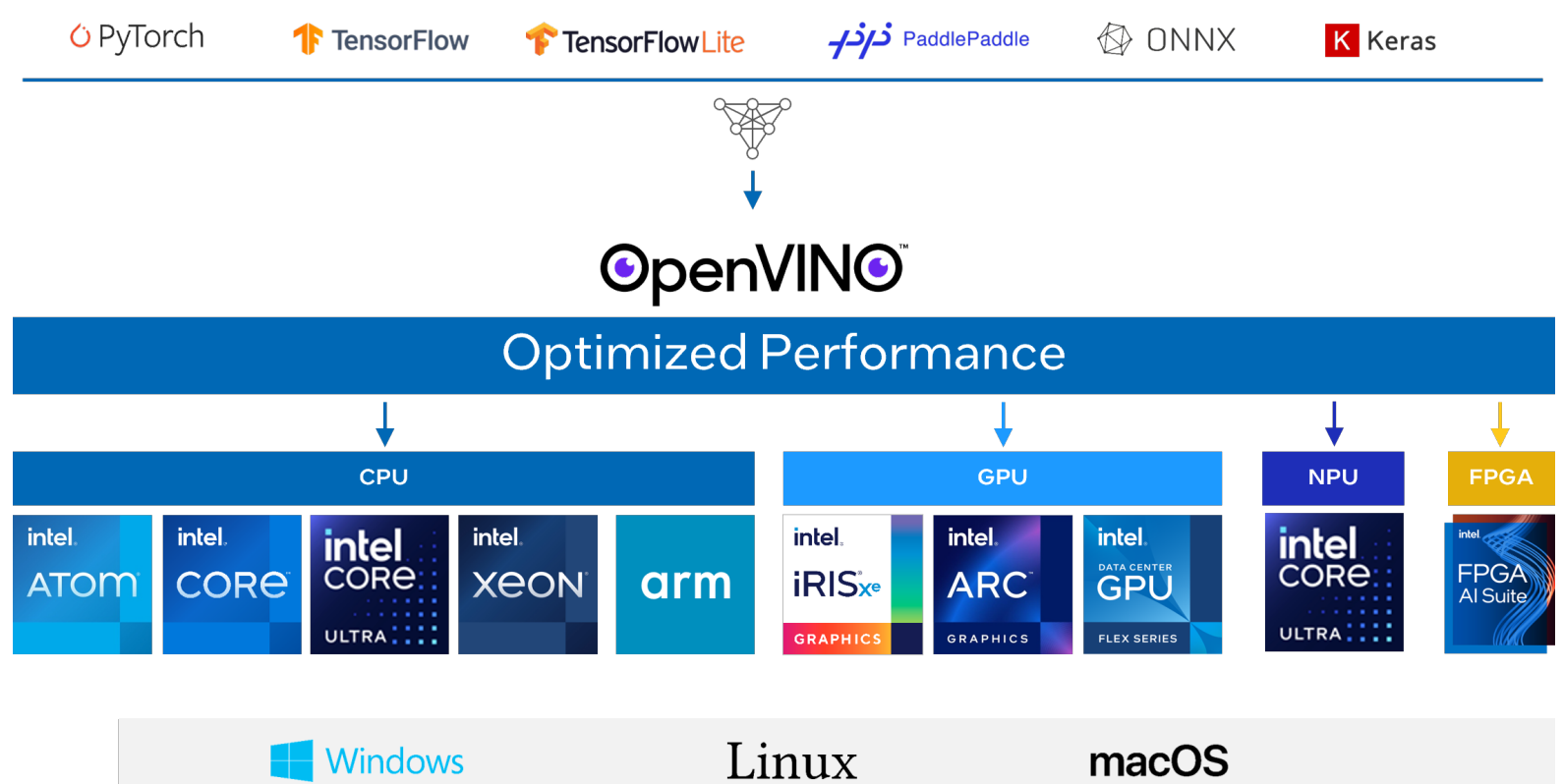
OpenVINO [accelerates inference](#) and [simplifies deployment across hardware](#), with a “build once, deploy everywhere” philosophy. To accomplish this, OpenVINO [supports + integrates with frameworks](#) (like PyTorch) and offers [advanced compression capabilities](#).

Build your model in the training framework or grab a pre-trained model from Hugging Face

Optimize your model for faster responses & smaller memory

Deploy the same model across hardware, leveraging automatic performance enhancements

Leverage the hardware's AI acceleration by default



OpenVINO Installation

[Linux install](#)

[Windows install](#)

[macOS install](#)

[PyPI example](#) for Linux, macOS & Windows:

```
#set up python venv  
python -m pip install openvino
```

The [install table](#) also has: APT, YUM, Conda, vcpkg, Homebrew, Docker, Conan, & npm

Interactive Notebook Examples

Test out [150+ interactive Jupyter notebooks](#) with [cutting-edge open-source models](#).

Includes model compression, pipeline details, interactive GUIs, and more.

Try out top models for a [range of use cases](#), including:

[LLMs](#)

[YOLO-v9](#)

[Stable Diffusion](#)

[CLIP](#)

[Segment Anything](#)

[Whisper](#)

[Setup:](#)

[Windows](#)

[Ubuntu](#)

[macOS](#)

[RedHat](#)

[CentOS](#)

[AzureML](#)

[Docker](#)

[SageMaker](#)

Model Compression with NNCF

NNCF is OpenVINO's [deep learning model compression tool](#), offering cutting-edge AI [compression capabilities](#), including:

1. **Quantization**: reducing the bit-size of the weights, while preserving accuracy
2. **Weight Compression**: easy post-training optimization for LLMs+
3. **Pruning for Sparsity**: drop connections in the model that don't add value
4. **Model Distillation**: a larger 'teacher' model trains a smaller 'student' model

Compression results in smaller and faster models that can be deployed across devices.

Easy install: `pip install nncf`

[Documentation](#)[GitHub](#)[NNCF Notebooks](#)[NNCF + Hugging Face](#)

PyTorch + OpenVINO Options

PyTorch models can be [directly converted](#) within OpenVINO™:

```
import openvino as ov
import torch
model = torch.load("model.pt") # Convert model loaded from PyTorch file
model.eval()
ov_model = ov.convert_model(model)
core = ov.Core()
compiled_model = core.compile_model(ov_model) # Compile model from memory
```

Or, you can use the [OpenVINO backend](#) for torch.compile:

```
import openvino.torch
import torch
# Compile PyTorch model #
opts = {"device" : "CPU", "config" : {"PERFORMANCE_HINT" : "LATENCY"}}
compiled_model = torch.compile(model, backend="openvino", options=opts)
```

[Direct conversion](#)[PyTorch Backend](#)[Examples](#)[Blog](#)

Performance Features

OpenVINO can do [automatic performance enhancements](#) at runtime customized to your hardware (preserving model accuracy), including:

Asynchronous execution, batch processing, tensor fusion, load balancing, dynamic inference parallelism, automatic BF16 conversion, and more.

Creates a smaller memory footprint of framework + model improving edge deployments.

There are also optional security features: [the ability to compute on an encrypted model](#).

Additional [advanced performance features](#):

- [Automatic Device Selection \(AUTO\)](#) selects the best available device
- [Multi-Device Execution \(MULTI\)](#) parallelizes inference across devices
- [Heterogeneous Execution \(HETERO\)](#) efficiently splits inference between cores
- [Automatic Batching](#) ad-hoc groups inference requests for max memory/core utilization
- [Performance Hints](#) auto-adjusts runtime parameters to prioritize latency or throughput
- [Dynamic Shapes](#) reshapes models to accept arbitrarily-sized inputs, for data flexibility
- [Benchmark Tool](#) characterizes model performance in various hardware and pipelines

Supported Hardware

OpenVINO supports [CPU](#), [GPU](#), and [NPU](#). ([Specifications](#))



The plugin architecture of OpenVINO enables development and plug-independent inference solutions dedicated to different devices. Learn more about the [Plugin](#), [OpenVINO Plugin Library](#), and [how to build one with CMake](#).

Additional community-supported plugins for Nvidia, Java and Rust can be found [here](#).

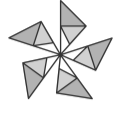
OpenVINO can Accelerate as a Backend

If you want to stay in another framework API, OpenVINO provides accelerating backends:



PyTorch

```
import openvino.torch
#compile PyTorch model as usual with PyTorch
compiled_model = torch.compile(model, backend="openvino", options =
 {"device" : "CPU"})
```



ONNX Runtime

```
onnx_model = onnx.load("model.onnx")
onnx.save_model(onnx_model, 'saved_model.onnx')
sess.set_providers(['OpenVINOExecutionProvider'])
```



Hugging Face

```
from optimum.intel import OVModelForCausalLM
#define model_id, use transformers tokenizer & pipeline
model = OVModelForCausalLM.from_pretrained(model_id)
pipe = pipeline("text-generation", model=model, tokenizer=tokenizer)
```



Nvidia Triton

```
$ docker run --rm -p 8000:8000 -p 8001:8001 -p 8002:8002 -v /path/to/
model_repository:/models nvr.io/nvidia/tritonserver:<xx.yy>-
py3 tritonserver --model-repository=/models
```

Config File:

```
name: "model_a"
backend: "openvino"
```



LangChain

```
ov_llm=HuggingFacePipeline.from_model_id(...backend="openvino",
      model_kwargs={"device":"CPU","ov_config": ov_config})
ov_chain = prompt | ov_llm
print(ov_chain.invoke({"question":"what is neurobiology?"}))
```

Hugging Face Integration

[Hugging Face + Intel Optimum](#) offers [OpenVINO integration](#) with Hugging Face models and pipelines. You can grab pre-optimized models and use OpenVINO compression features & Runtime capabilities within the Hugging Face API.

Here is an example with an LLM (from [this notebook](#)) on how to swap default Hugging Face code for optimized OpenVINO-Hugging Face code:

```
-from transformers import AutoModelForCausalLM
+from optimum.intel.openvino import OVModelForCausalLM
from transformers import AutoTokenizer, pipeline
model_id = "togethercomputer/RedPajama-INCITE-Chat-3B-v1"
-model = AutoModelForCausalLM.from_pretrained(model_id)
+model = OVModelForCausalLM.from_pretrained(model_id, export=True)
```

[Inference Documentation](#)

[Compression Documentation](#)

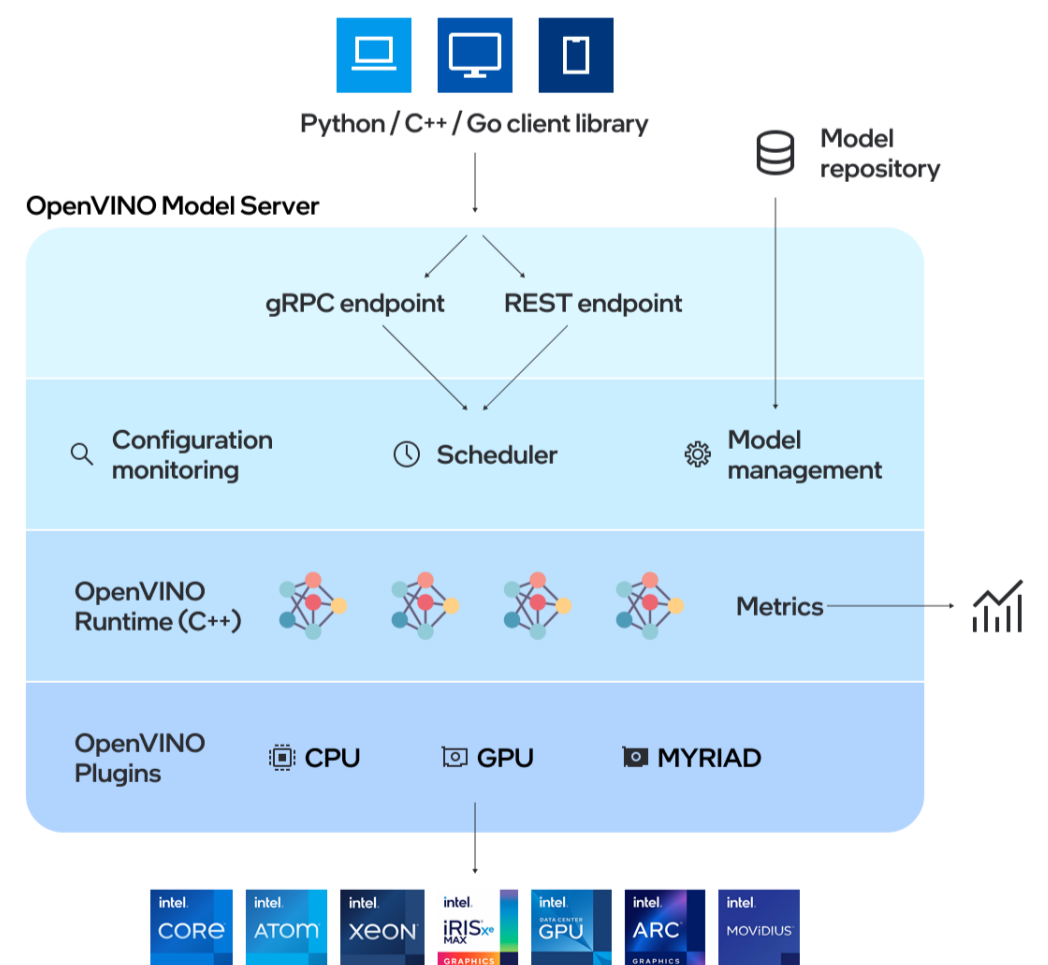
[Reference Documentation](#)

[Examples](#)

OpenVINO™ Model Server (OVMS)

OVMS hosts models and makes them accessible to software components over standard network protocols: a client sends a request to the model server, which performs model inference and sends a response back to the client.

OVMS is a high-performance system for serving models. Implemented in C++ for scalability and optimized for deployment on Intel architectures, the model server uses a [KServe](#) standard, while applying OpenVINO for inference execution. Inference service is provided via gRPC or REST API, making deploying new models/experiments easy.



[Documentation](#)

[QuickStart Guide](#)

[Features](#)

[Demos](#)

Join the OpenVINO Community

We welcome [code contributions](#) and [feedback!](#) [Submit on GitHub](#) and engage on [GitHub discussions](#) or [our forum](#). Share your examples ([via PR](#)) to be featured [here](#).