OpenVINO introduction

Obolenskiy Arseniy, Nesterov Alexander

ITLab

December 4, 2025

Contents

Overview

What is OpenVINO?

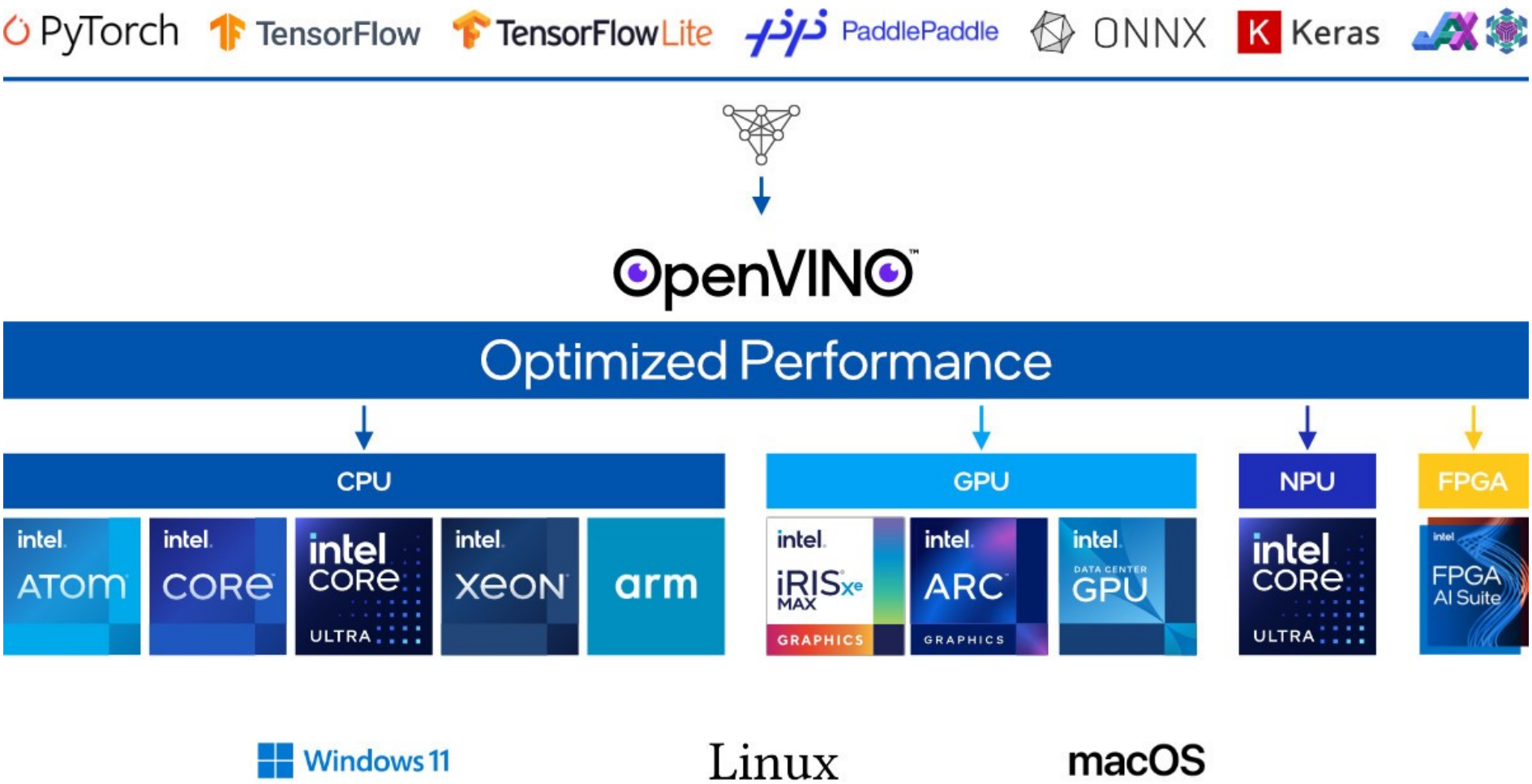
OpenVINO (Open Visual Inference and Neural Network Optimization) is a toolkit developed by Intel for optimizing and deploying deep learning models for inference on Intel and other vendors hardware. It provides a unified API and a set of tools to streamline the process of model optimization, conversion, and deployment across various Intel architectures.

OpenVINO at a Glance

- **Purpose:** Optimize and deploy AI inference across CPUs (x86, ARM, RISC-V), GPUs, NPUs, and other accelerators
- **Core components:** Runtime (Inference Engine), Plugins (Targets), Frontends
- **Model formats (Frontends):** IR (`.xml`/`.bin`), ONNX (`.onnx`), TensorFlow (SavedModel/MetaGraph/frozen `.pb`/`.pbtxt`), TensorFlow Lite (`.tflite`), PaddlePaddle (`.pdmodel`), PyTorch (TorchScript/FX `.pt`/`.pth`)
- **Targets:** CPU, GPU (e.g., Intel Arc), NPU, and more via plugins
- **Key benefits:** Performance, portability, unified API, quantization (INT8), easy deployment

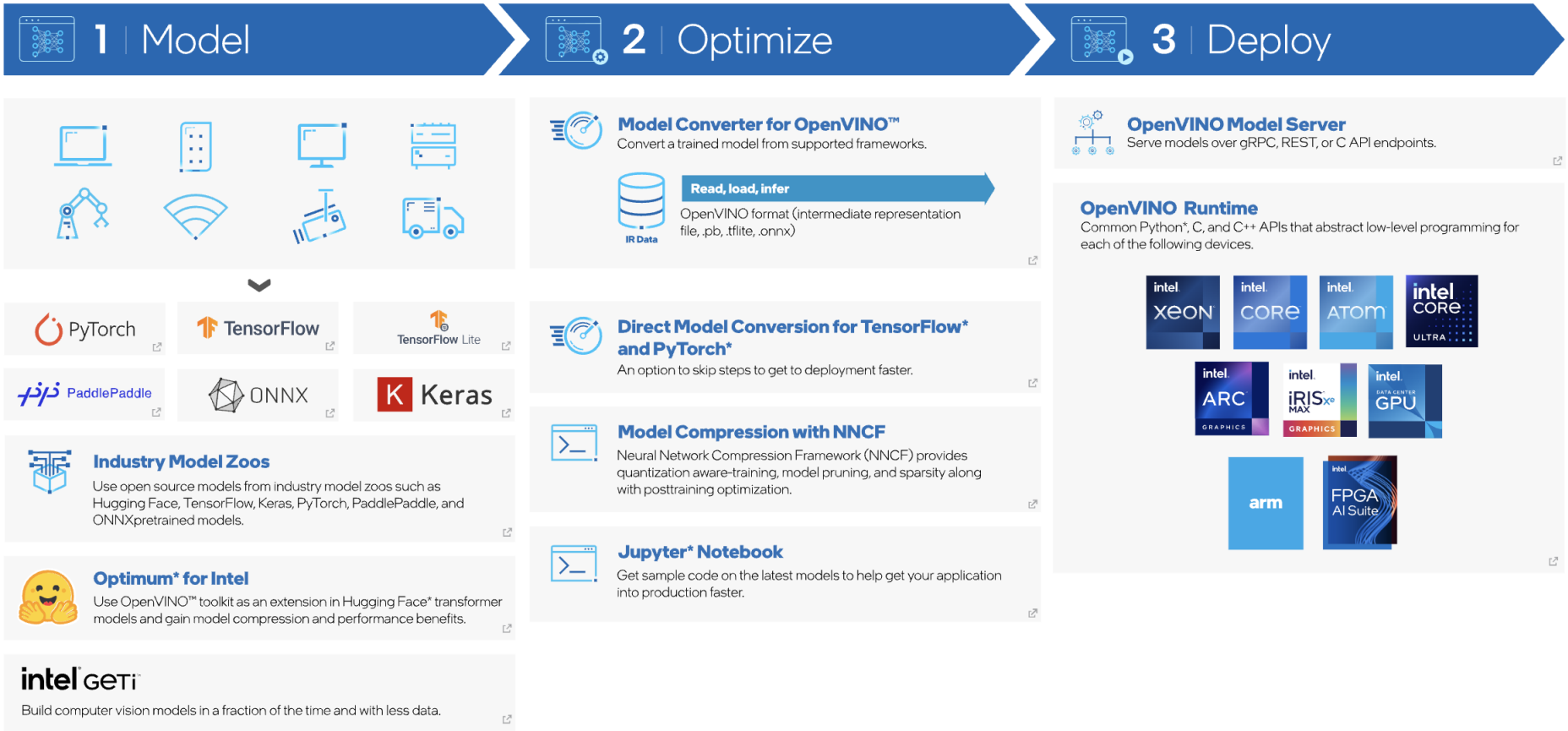
Reference: docs.openvino.ai

Overview Diagram



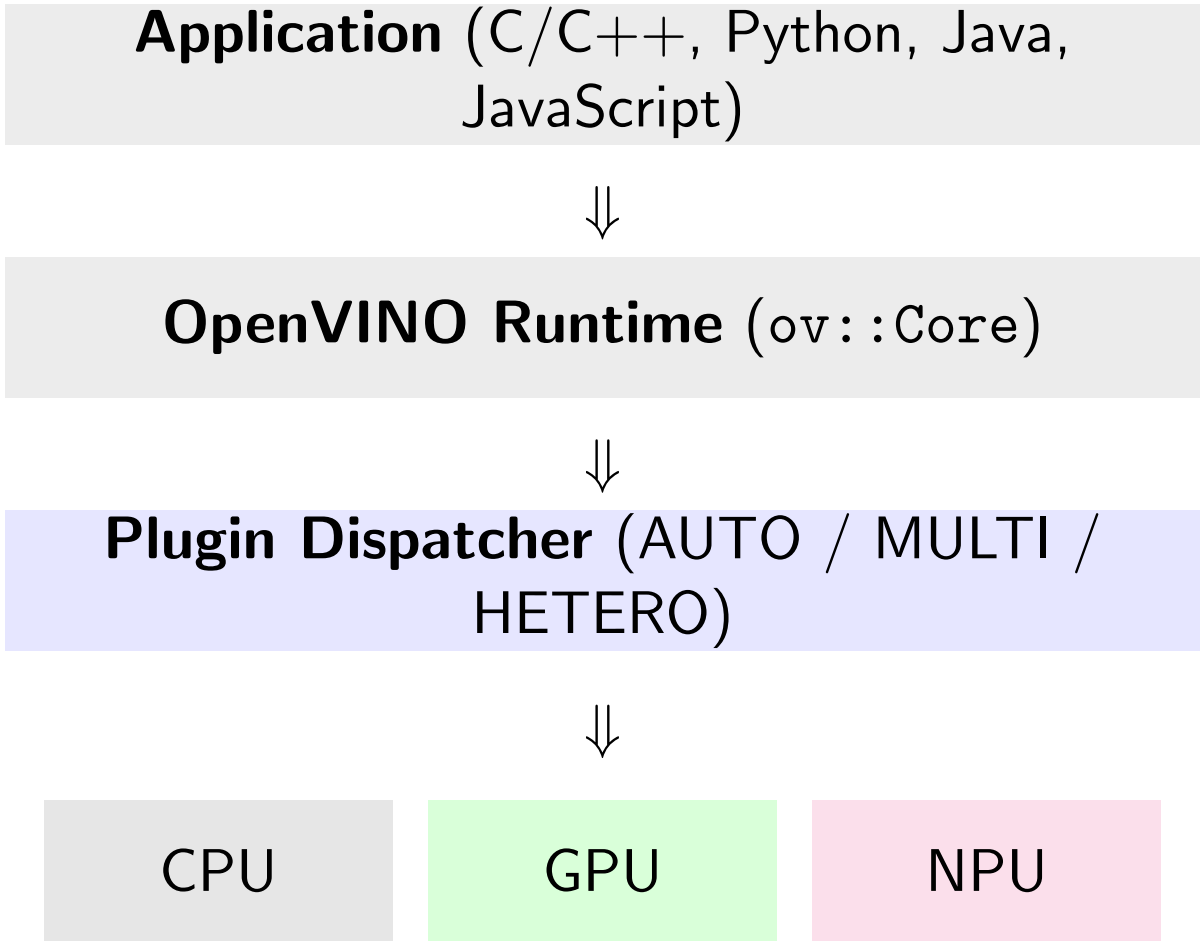
Source: https://docs.openvino.ai/2025/index.html

Workflow Overview



Source: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html

Device Plugins Architecture



Examples: `CPU`, `GPU.0`, `NPU`, `AUTO:CPU,GPU`, `MULTI:GPU,CPU`, `HETERO:GPU,CPU`

Device Plugin Details

- **CPU**: High compatibility and strong baseline performance; uses optimized kernels (e.g., oneDNN). Supports FP32/FP16/INT8 with quantized models.
- **GPU**: Integrated and discrete Intel GPUs via Level Zero/OpenCL, delivering strong FP16 and INT8 throughput and benefiting from device-specific kernels and memory bandwidth.
- **NPU**: Intel NPU (e.g., Core Ultra) for efficient, low-power inference on common vision/LLM ops; ideal for always-on and battery-sensitive workloads.
- **TEMPLATE plugin**: Reference backend for building custom device plugins; demonstrates the plugin API (compiled model, infer request, op support, memory) and is useful for prototyping.

See: https://docs.openvino.ai/2025/documentation/compatibility-and-support/supported-devices.html — Supported devices

Inference Modes

- **AUTO plugin**: Chooses the "best" device available at runtime; can constrain candidates, e.g., `AUTO:GPU,CPU`.
- **MULTI plugin**: Executes across multiple devices in parallel to maximize throughput, e.g., `MULTI:GPU,CPU`.
- **HETERO plugin**: Splits a single graph by layer/op support across devices, e.g., heavy ops on GPU, fallbacks on CPU.

See: https://docs.openvino.ai/2025/openvino-workflow/running-inference/inference-devices-and-modes.html — Inference Devices and Modes

OpenVINO Network Intermediate Representation

What is OpenVINO IR?

- **IR (Intermediate Representation)** is OpenVINO's graph format used by the runtime for efficient inference.
- A model is stored as two files: `model.xml` (network topology, layers, attributes) and `model.bin` (weights).
- IR is framework-agnostic: models from PyTorch, TensorFlow, ONNX, and others are converted into a unified format.
- The same IR can be executed on different devices (CPU, GPU, NPU, etc.) via plugins without changing the model itself.

IR Structure and Benefits

- Graph of operations (nodes) and tensors (edges) with explicit input / output shapes and data types.
- Uses OpenVINO operation sets (*opsets*) that define supported ops and attributes for compatibility across versions.
- Enables offline optimizations such as constant folding, layout changes, precision conversions (FP32 $\rightarrow$ FP16/INT8).
- Portable artifact for CI/CD workflows: generate IR once, then deploy to multiple targets (cloud, edge, embedded) with the same files.

API Examples

Language Frontends

- **C++**: primary, feature-complete API for production workloads and samples; direct access to `ov::Core` and low-level controls.
- **C**: lightweight C wrapper for integrating OpenVINO into C-only or legacy codebases.
- **Python**: high-level API (`openvino`, `openvino.runtime`) for rapid prototyping, notebooks, and integration with the Python ML ecosystem.
- **Java** (contrib, optional): bindings for JVM-based services and desktop apps, suitable for server-side inference pipelines.
- **JavaScript**: Web and Node.js frontends (via WebAssembly and native addons) for running inference in browsers or JS backends.

See: https://docs.openvino.ai/2025/api/api_reference.html

Python API Example (YOLO-style Model)

- **Goal:** Run object detection with a YOLO-like model using OpenVINO Runtime
- **Code sketch**

```
import openvino as ov

core = ov.Core()
model = core.read_model("yolo.xml")
compiled = core.compile_model(model, "AUTO")
infer_request = compiled.create_infer_request()
infer_request.set_tensor(input_name, image_tensor)
infer_request.infer()
output = infer_request.get_tensor(output_name)
```

C++ API Example (YOLO-style Model)

- **Goal:** Same pipeline in C++ with `ov::Core`
- **Code sketch**

```
#include <openvino/openvino.hpp>

int main(int argc, char* argv[]) {
  ...
  ov::Core core;
  auto model = core.read_model("yolo.xml");
  auto compiled = core.compile_model(model, "AUTO");
  auto infer_request = compiled.create_infer_request();
  infer_request.set_tensor(input_port, input_tensor);
  infer_request.infer();
  auto output = infer_request.get_tensor(output_port);
  ...
}
```

Tools and Benchmarks

Brief Tools Overview

- **benchmark_app**: measures latency / throughput on target devices
- **OpenVINO Notebooks**: interactive tutorials for many models (YOLO, SSD, Segmentation, LLMs)
  https://github.com/openvinotoolkit/openvino_notebooks

benchmark_app Usage Examples

- **Basic run on CPU**
  - `benchmark_app -m yolo.xml -d CPU`
- **Run on GPU with async API**
  - `benchmark_app -m yolo.xml -d GPU -api async`
- **Use AUTO plugin and prioritize throughput**
  - `benchmark_app -m yolo.xml -d AUTO -hint throughput`
- **What to look at**
  - Latency, FPS, device utilization, batch size, number of streams

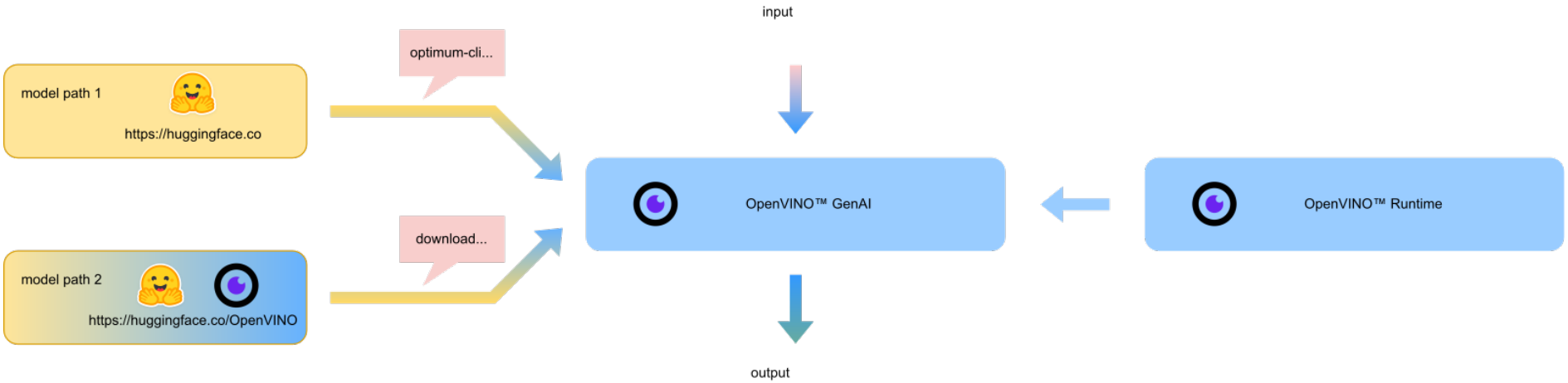
See: https://docs.openvino.ai/nightly/get-started/learn-openvino/openvino-samples/benchmark-tool.html

GenAI with OpenVINO

OpenVINO GenAI

- **Target use cases:** chatbots, code assistants, summarization, RAG pipelines, text-to-image / image editing with diffusion models
- **Model types:** LLMs (decoder-only, encoder-decoder), vision-language models, diffusion models; integration with Hugging Face and ONNX model zoo
- **Optimizations:** 8-bit / 4-bit quantization, weight compression, low-rank adapters (LoRA), CPU/GPU-specific graph optimizations for lower latency
- **Deployment:** Python/C++ APIs, OpenVINO GenAI APIs, notebooks and samples for serving models locally or in containers
- **Resources:** https://github.com/openvinotoolkit/openvino.genai, https://docs.openvino.ai/2025/openvino-workflow-generative/inference-with-genai.html

GenAI Workflow Diagram



Source:
https://docs.openvino.ai/2025/openvino-workflow-generative/inference-with-genai.html

Getting OpenVINO

Installing OpenVINO (User)

- Use prebuilt packages from Intel:
  - Linux: `pip install openvino` (Python API + tools)
  - Windows: `pip install openvino` or installer from Intel website
- (Optional) Create isolated environment:
  - `python -m venv venv`
    `source venv/bin/activate` (Linux/macOS)
    `venv\Scripts\activate` (Windows)
- Verify installation in Python:

```
import openvino as ov
print(ov.__version__)
```

- Check available devices:

```
core = ov.Core()
print(core.get_available_devices())
```

Building OpenVINO (Developer)

Build from source (advanced):
https://github.com/openvinotoolkit/openvino/blob/master/docs/dev/build.m

- Clone sources:

```
git clone https://github.com/openvinotoolkit/openvino.git --recurse
    -submodules
cd openvino
```

- Install build dependencies (compiler, CMake, Python, git)
- Configure build directory:

```
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release -DENABLE_PYTHON=ON -
    DENABLE_TESTS=ON
```

  Note: the full cmake flags reference can be found in the documentation
- Build and run tests:

```
cmake --build build --parallel
```

References

References

- OpenVINO Official documentation: https://docs.openvino.ai/
- OpenVINO repository: https://github.com/openvinotoolkit/openvino
- OpenVINO Contrib: https://github.com/openvinotoolkit/openvino_contrib
- OpenVINO Notebooks: https://github.com/openvinotoolkit/openvino_notebooks
- OpenVINO GenAI project: https://github.com/openvinotoolkit/openvino.genai