HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

NPU Utilization for Edge AI

Assoc. Prof. La The Vinh
MSc. Mai Xuan Ngoc,
School of Information Technology and Communications
Hanoi University of Science and Technology.

ONE LOVE. ONE FUTURE.

HUST

hust.edu.vn fb.com/dhbkhn

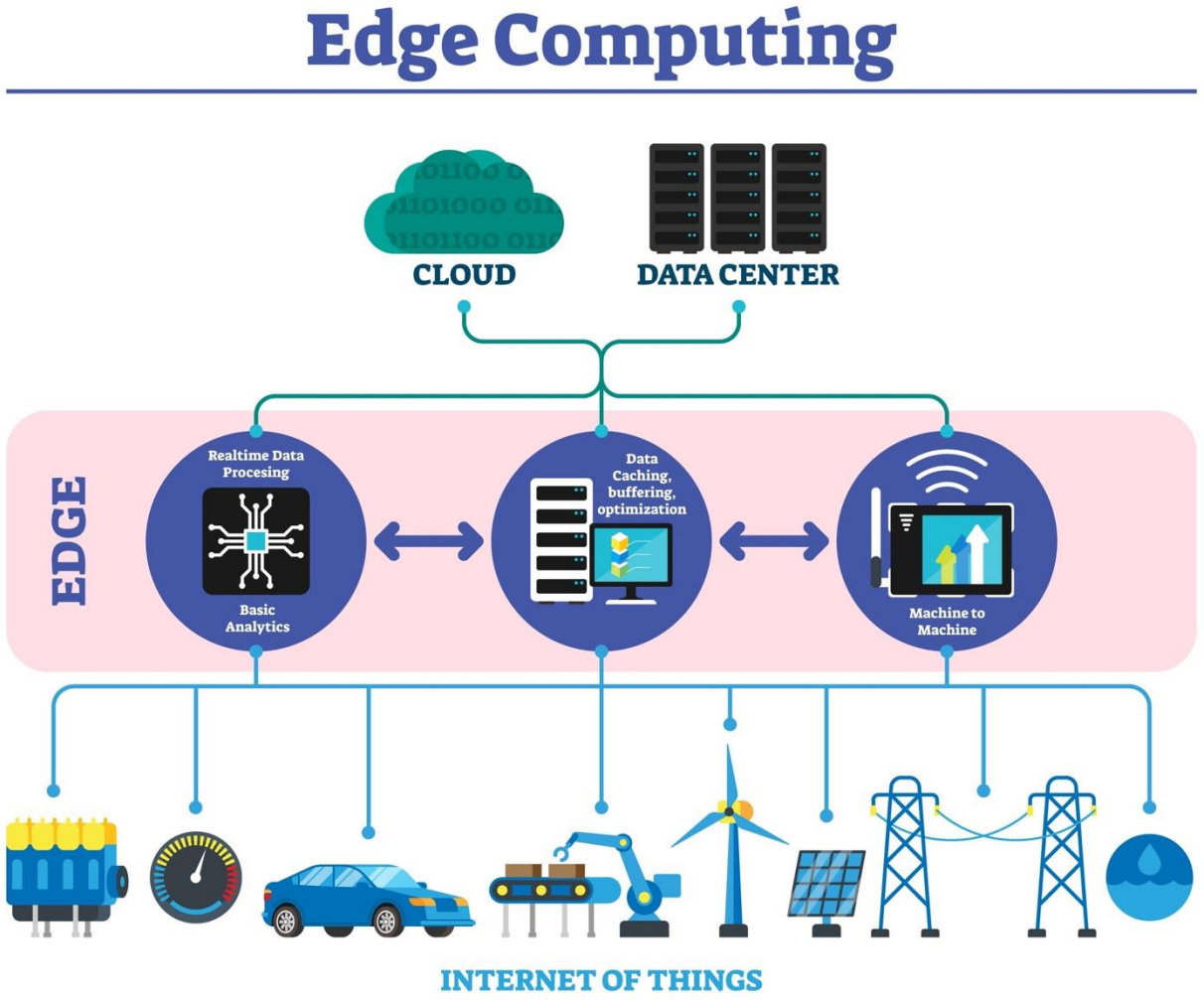
Outline

1. Edge AI and AI-accelerated hardware
2. Qualcomm Edge AI hardware
3. Qualcomm AI SDK
4. Qualcomm AI Hub
5. Application development
6. Production

1. Edge AI and AI-Accelerated Hardware

What is Edge AI?

- **On-site processing:** Runs AI algorithms directly on the end device.
- **Low Latency:** Eliminates transmission delays.
- **Maximum security:** Personal data is stored and processed locally.
- **Resource optimization:** Saves internet bandwidth and reduces cloud infrastructure operating costs.



How We Got to the NPU

1999
GeForce 256
NVIDIA coins the term “GPU”
graphics goes parallel

2006
CUDA
The GPU becomes a
general-purpose machine

2012
AlexNet
Wins ImageNet on 2× GTX 580
deep learning takes off

2015
TPU v1
Google builds its own ASIC
for inference in the data centre

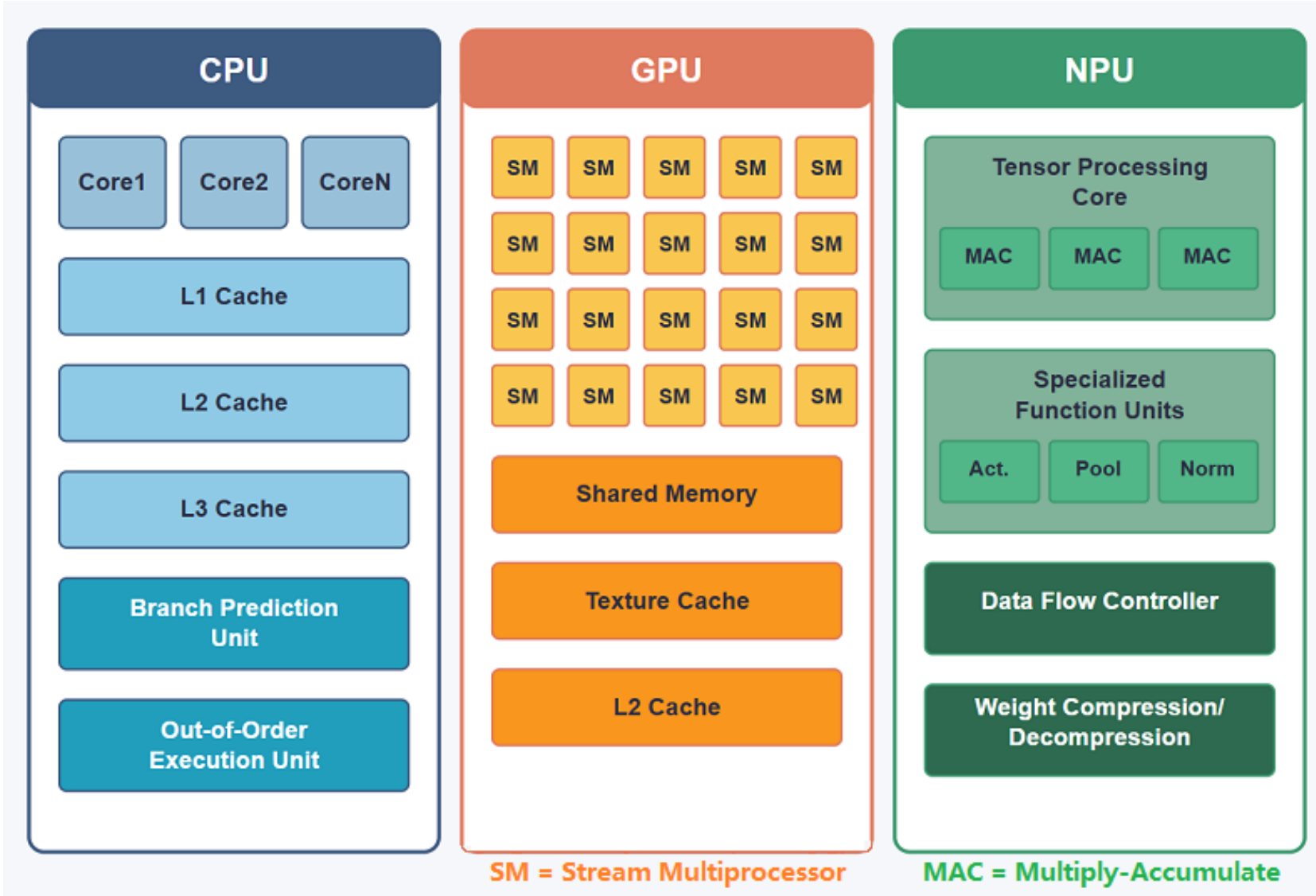
2017
NPUs reach phones
Kirin 970 · Apple A11
Neural Engine

2021
QCS6490 · Hexagon HMX
12 TOPS for IoT/robotics
new matrix engine

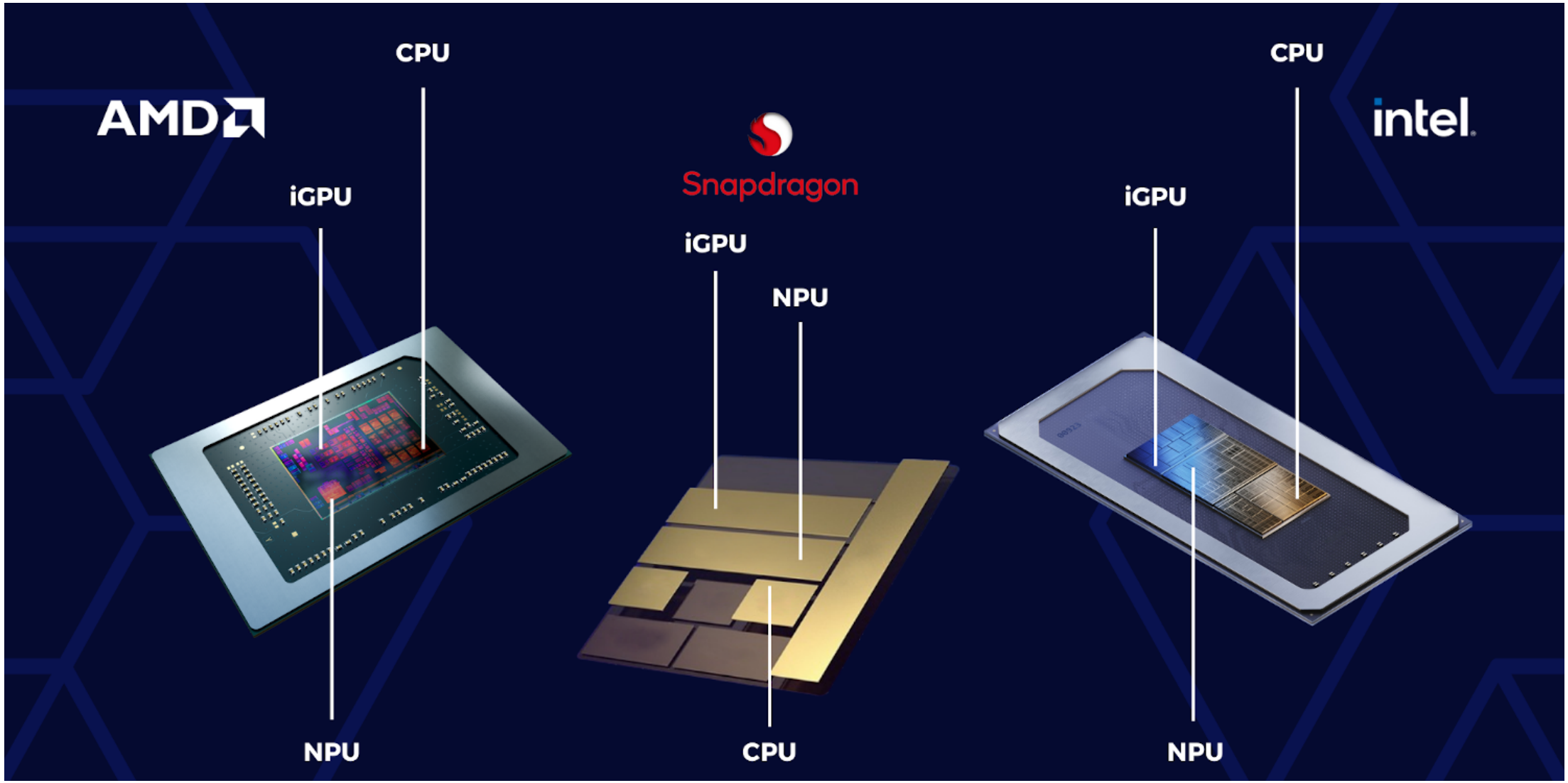
2024
The AI PC
Snapdragon X Elite, 45 TOPS
Copilot+ requires ≥ 40 TOPS

CPU vs GPU vs NPU

- **CPU** — general purpose and flexible; always available
- **GPU** — strong at parallel work: matrices, image processing
- **NPU** — purpose-built for neural network inference
- The NPU normally gives the **best performance per watt** for AI workloads



AI Inference Is Moving to the NPU

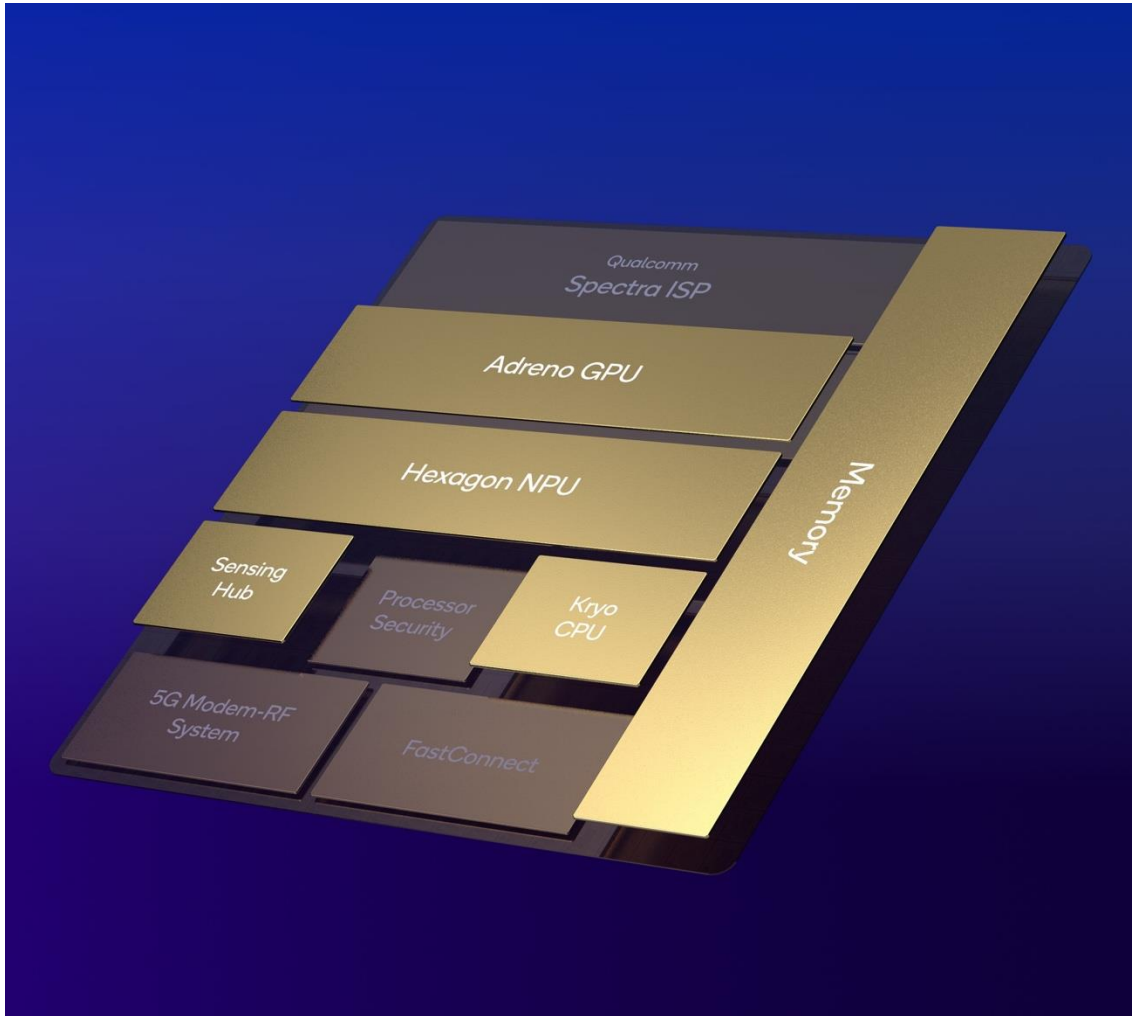


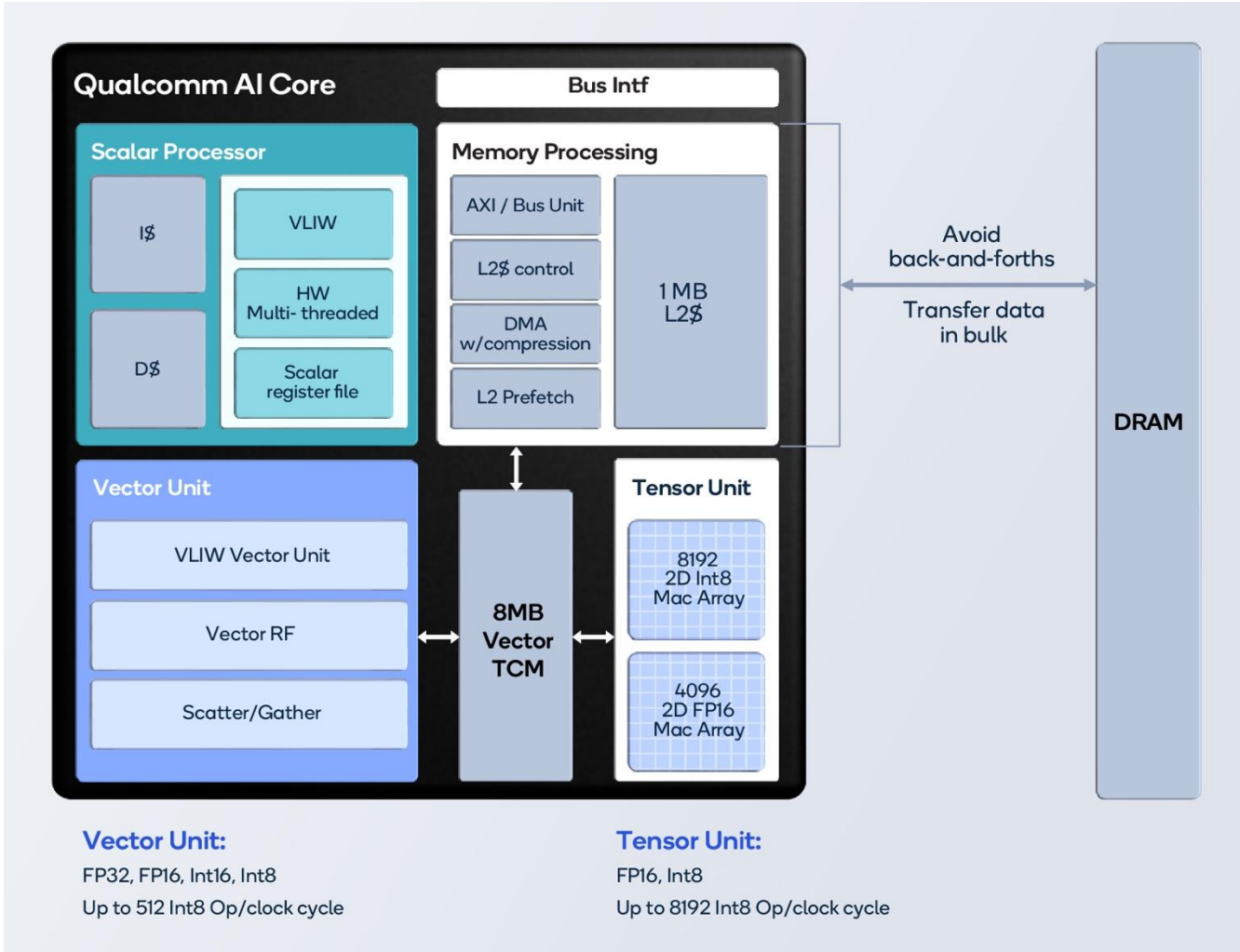
2. Qualcomm Edge AI Hardware

Qualcomm On-device AI

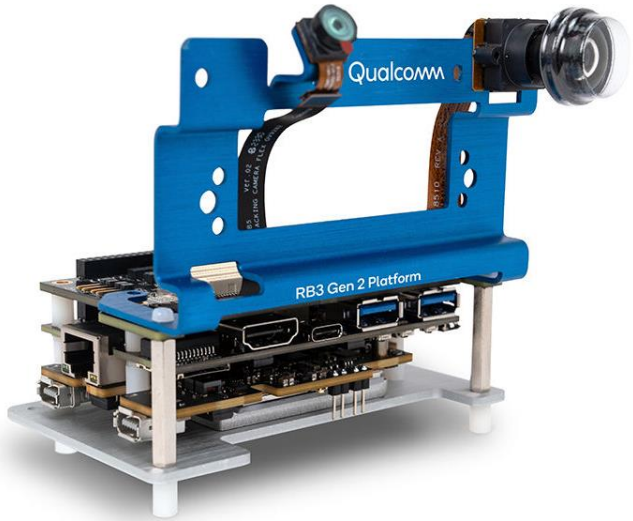
- Smartphone
- Laptop
- Robot
- Wearable device
- Automotive

Qualcomm AI Engine

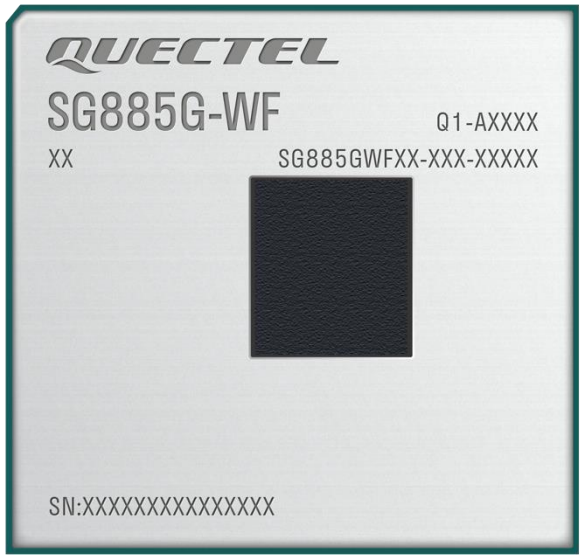




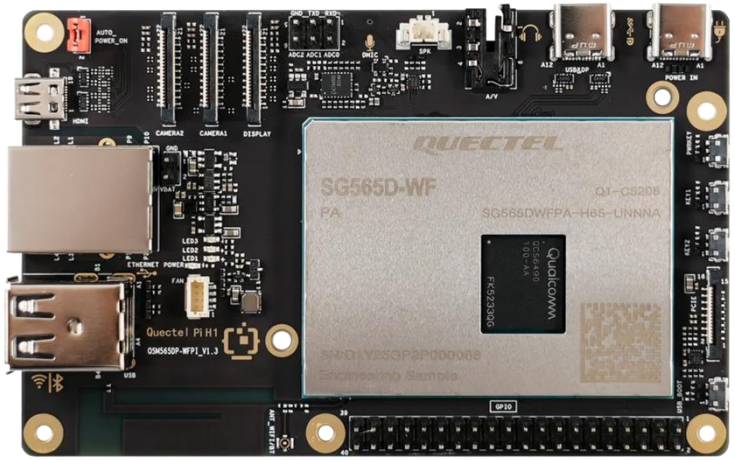
Thundercomm



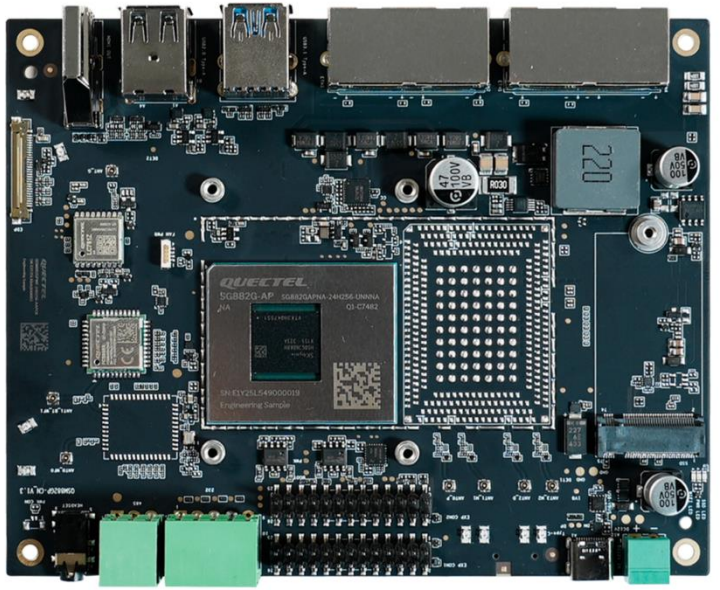
Quectel


QUECTEL
SG885G-WF
Q1-AXXXX
XX
SG885GWFXX-XXX-XXXXX
SN:XXXXXXXXXXXXXXX


DEBUG
SIM2
EAR
SUB CAM
LCM
Main CAM
Speaker
SMART-EVB_V1.3
SIM1
TF CARD
UART
JTAG


QUECTEL
SG565D-WF
PA
Quectel Pi H1


QUECTEL
SG560D-XX
Q1-XXXXX
XX
SG560DXXXX-XXX-XXXXXXX
SN:XXXXXXXXXXXXXXX
IMEI1:XXXXXXXXXXXXXXX
IMEI2:XXXXXXXXXXXXXXX

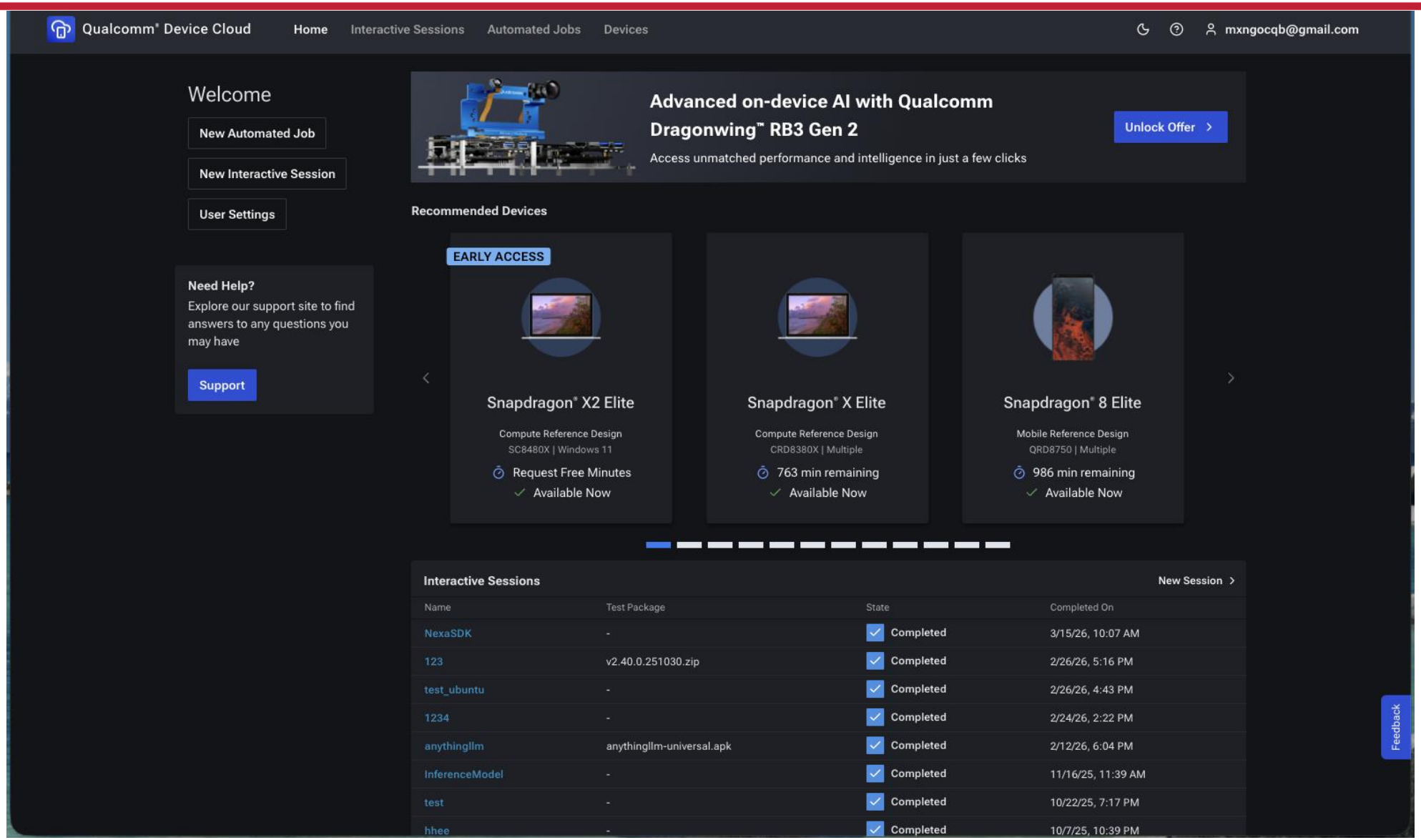

QUECTEL
SG882G-AP


KIMQ 8550
Qualcomm


KIMQ 6490


Qualcomm

Qualcomm Device Cloud



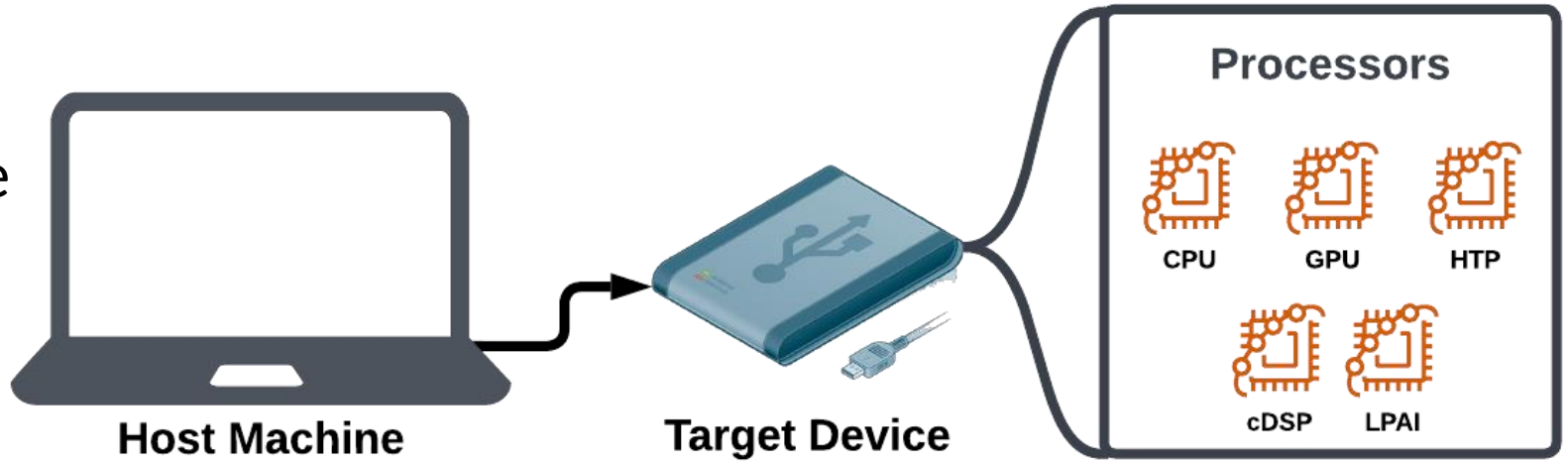
HUST

https://qdc.qualcomm.com/

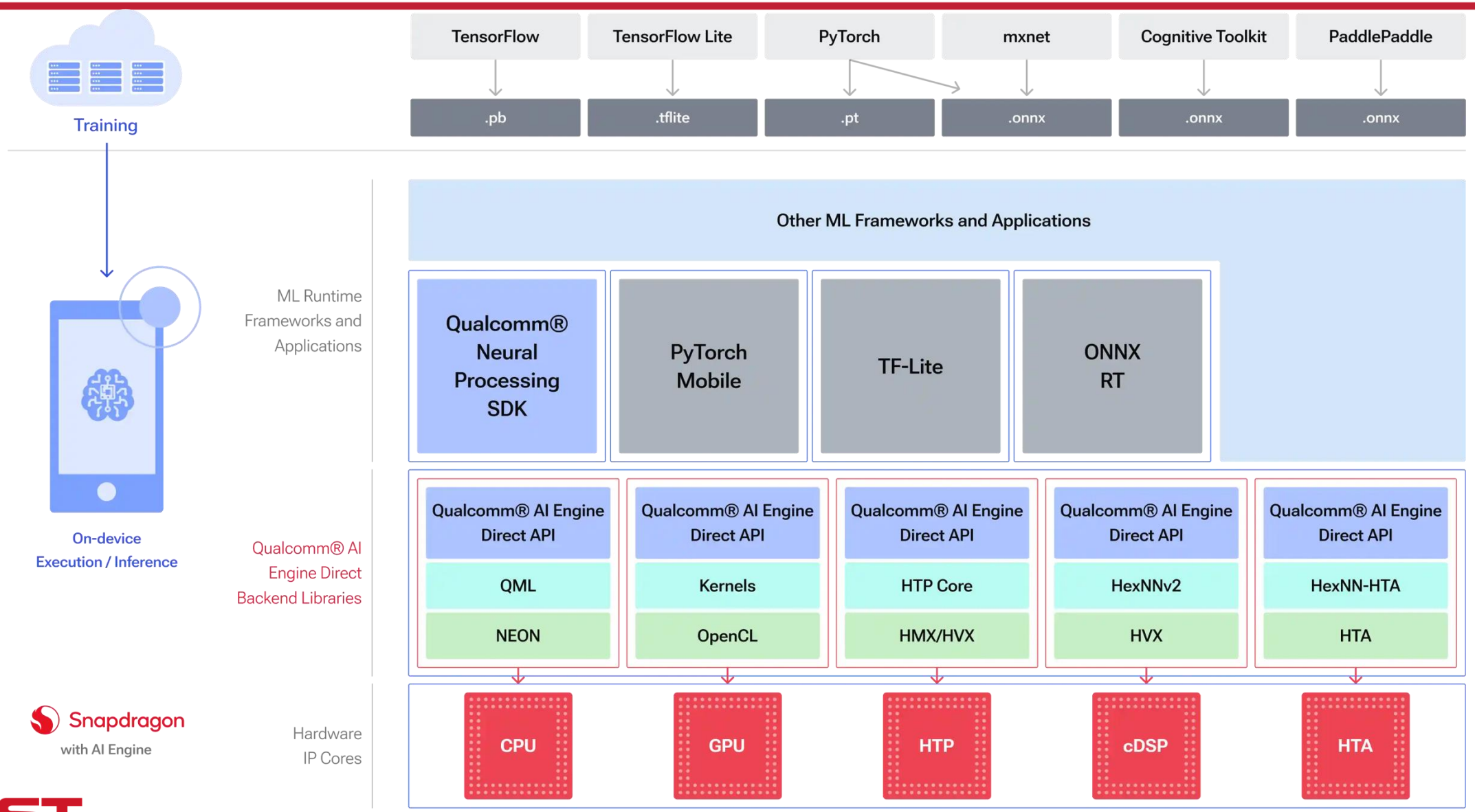
3. Qualcomm AI SDK

Qualcomm AI Engine Direct SDK

- **SNPE:** Simple multi-processor deployment
- **QNN:** Fine-grained hardware control
- **GENIE:** LLM/Generative AI runtime
- **QAIRT API:** Unified runtime for **CPU, GPU & HTP/NPU**

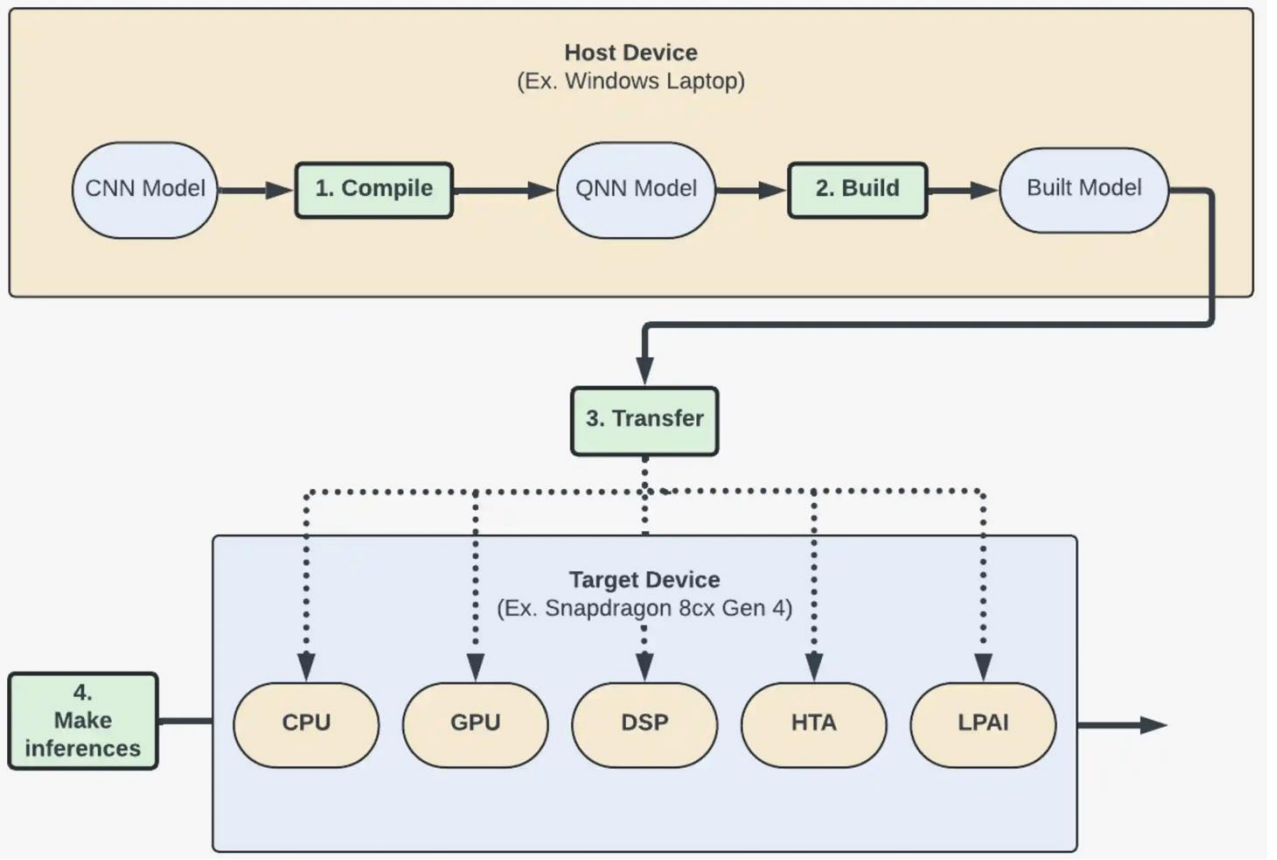


QNN SDK


Training
TensorFlow
TensorFlow Lite
PyTorch
mxnet
Cognitive Toolkit
PaddlePaddle
.pb
.tflite
.pt
.onnx
.onnx
.onnx
On-device
Execution / Inference
ML Runtime
Frameworks and
Applications
Other ML Frameworks and Applications
Qualcomm® Neural Processing SDK
PyTorch Mobile
TF-Lite
ONNX RT
Qualcomm® AI Engine
Direct Backend Libraries
Qualcomm® AI Engine Direct API
QML
NEON
Kernels
OpenCL
HTP Core
HMX/HVX
HexNNv2
HVX
HexNN-HTA
HTA
Snapdragon
with AI Engine
Hardware
IP Cores
CPU
GPU
HTP
cDSP
HTA

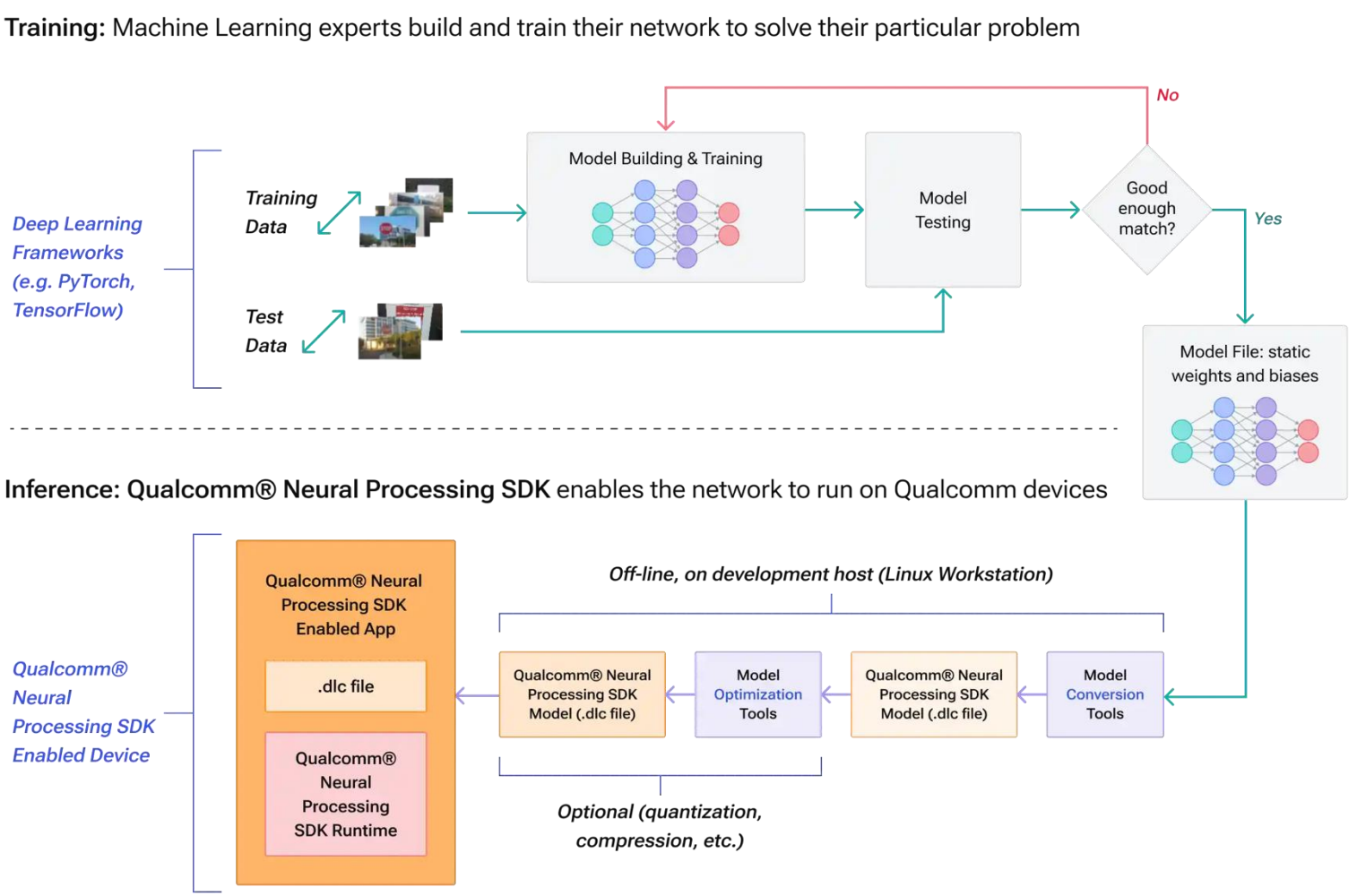
QNN SDK

Stage	QNN CLI Tools
Convert	qnn-onnx-converter, qnn-tensorflow-converter, qnn-pytorch-converter, qnn-tflite-converter
Quantize	Converter + calibration dataset
Compile	qnn-model-lib-generator
Prepare HTP	qnn-context-binary-generator
Deploy / Run	qnn-net-run
Benchmark	qnn-throughput-net-run
Profile	qnn-profile-viewer



SNPE

Stage	SNPE CLI Tool
Convert	snpe-onnx-to-dlc snpe-tensorflow-to-dlc
Quantize	snpe-dlc-quantize
Prepare HTP	snpe-dlc-graph-prepare
Run	snpe-net-run
Benchmark	snpe-throughput-net-run
Validate Device	snpe-platform-validator

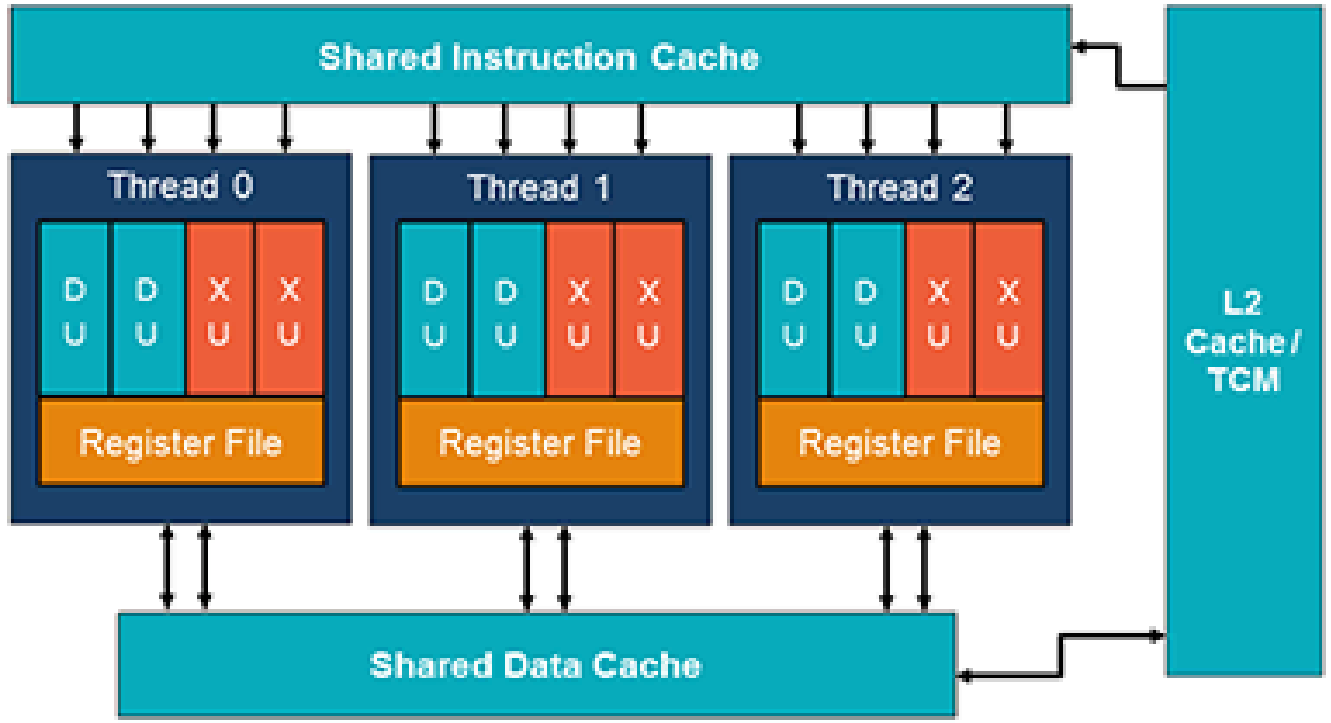


QAIRT SDK

	SNPE	QNN	QAIRT API	GENIE
Purpose	Easy AI deployment	Fine-grained optimization	**Unified low-level runtime**	Generative AI
Abstraction	High	Medium/Low	**Low**	High
Hardware	CPU/GPU/HTP	CPU/GPU/HTP/ DSP/LPAI	**CPU/GPU/HTP via one API**	Mainly HTP/GPU
Key feature	Simple workflow	Hardware-specific control	**Single C/C++ API**	LLM/VLM inference
Best for	AI applications	Performance optimization	**System integration**	LLM applications

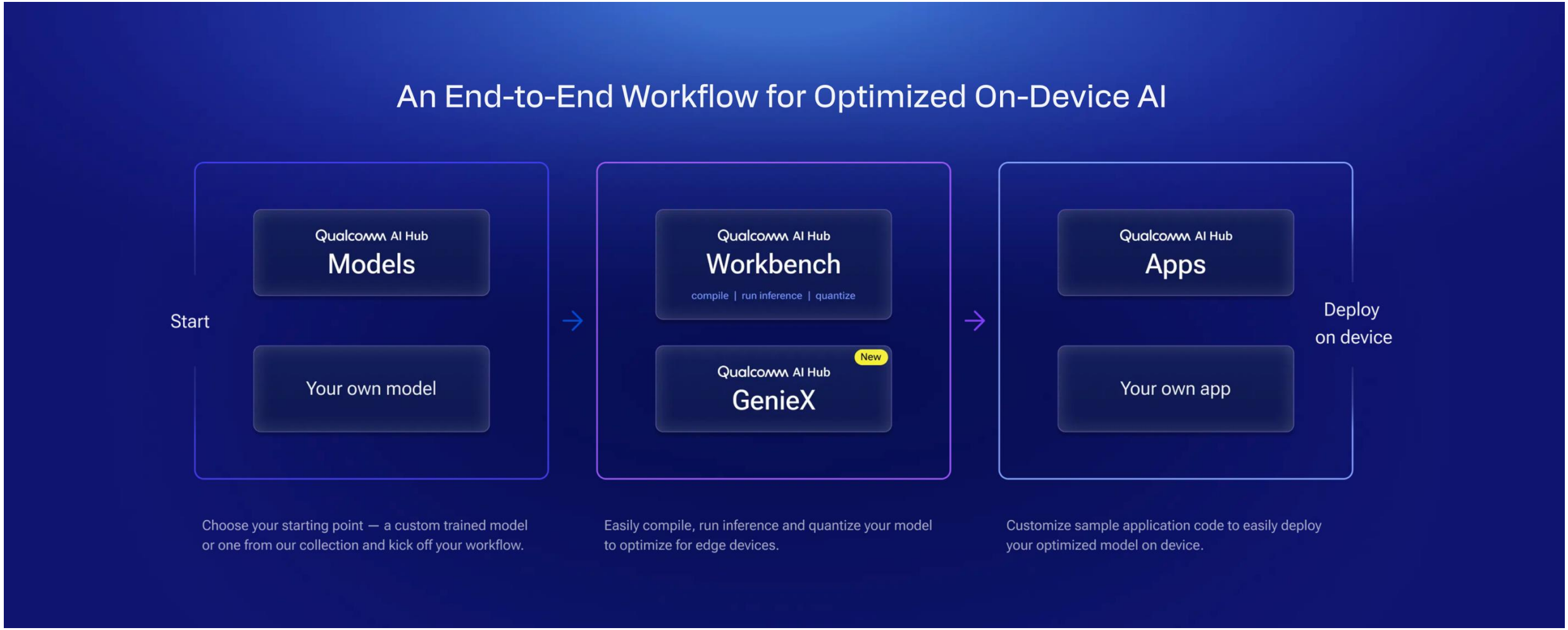
HEXAGON NPU SDK

- **Qualcomm Hexagon SDK** gives developers low-level access to the Hexagon DSP/NPU for optimized edge workloads.
- **Use cases**
  - Offload compute-heavy kernels from CPU
  - Optimize audio, vision, signal processing
  - Use HVX vector acceleration for custom operators
  - Improve latency and power efficiency on-device
- **Positioning**
  - **QNN / SNPE:** deploy AI models
  - **Hexagon SDK:** write custom optimized DSP/NPU kernels



4. Qualcomm AI Hub

Qualcomm AI Hub — An End-to-End Workflow

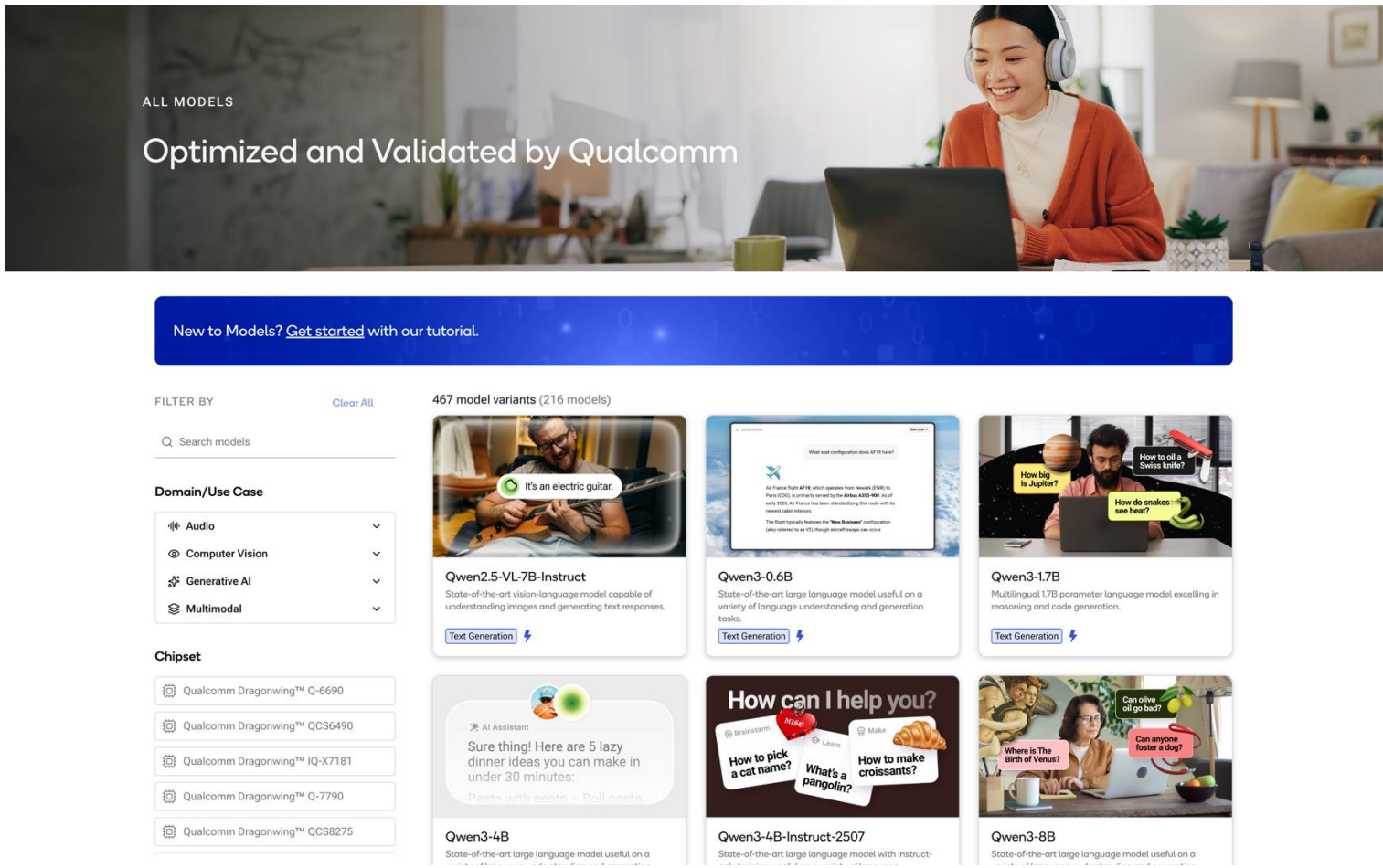


Qualcomm AI Hub — The Toolbox

Tool	What it is	What you use it for	Where
AI Hub Models	Catalogue of pre-optimised models — 216 models / 467 variants across vision, audio, multimodal and generative AI	Start from a model that is already exported, quantized and measured on Qualcomm silicon	aihub.qualcomm.com/models github.com/quic/ai-hub-models
AI Hub Workbench	Cloud service that compiles, quantizes, profiles and validates a model on a farm of real devices	Bring your own trained model and get back a device-ready binary plus a profiling report	workbench.aihub.qualcomm.com
GenieX (new)	On-device runtime for LLMs and VLMs — QAIRT plugin for the NPU, llama.cpp/GGML plugin for CPU and GPU	Run generative AI locally from CLI, Python, Java, Docker or an OpenAI-compatible server	geniex.aihub.qualcomm.com github.com/qualcomm/GenieX
AI Hub Apps	16 open-source reference applications for Android, Windows and Ubuntu, with full source	Start your application from code that already builds and runs on device	aihub.qualcomm.com/apps github.com/quic/ai-hub-apps
Device Cloud (adjacent)	Remote access to real Qualcomm boards and phones	Test, debug and benchmark when you do not have the target hardware on your desk	qdc.qualcomm.com

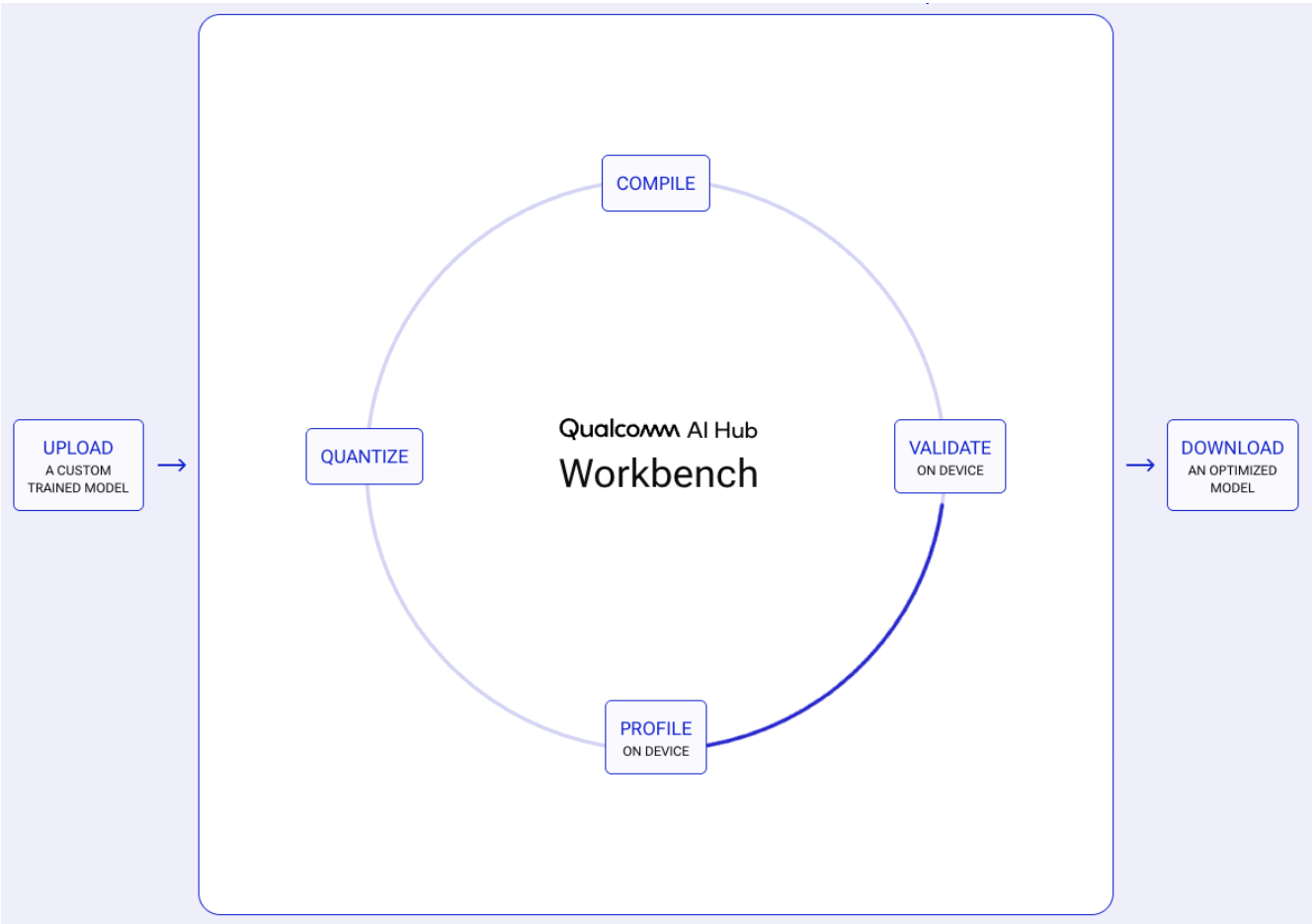
Qualcomm AI Hub Models

- **216 models / 467 variants** — computer vision, audio, multimodal, generative AI
- Every model is already **exported, quantized and measured** on real silicon
- Each model page lists **latency, memory and the NPU/GPU/CPU split** per chipset
- Download as **LiteRT (.tflite)**, **ONNX** or **QNN context binary (.bin)**
- Filter by **domain, chipset and runtime**
- Export recipes are open source — you can change them



Qualcomm AI Hub Workbench

- The optimisation service — **upload a trained model, get back an optimised on-device binary**
- Source formats: **PyTorch ExportedProgram (.pt2)**, **ONNX**, **LiteRT/TFLite** (TorchScript now deprecated)
- Target runtimes: **Qualcomm AI Runtime (QNN)**, **LiteRT**, **ONNX Runtime**
- Jobs run on a fleet of **real Snapdragon and Dragonwing devices** — not simulators
- Returns **inference time, load time and the per-layer compute-unit split**
- Free tier; driven from **Python API, CLI or the web UI**



AI Hub Workbench — The Five Job Types

Job	What it does	In → Out	Why you run it
Compile Job	Converts and compiles the graph for one target runtime and one device	PyTorch / ONNX / TFLite → .tflite · .onnx · QNN .bin	Produce the artefact you will actually ship
Quantize Job	Post-training quantization driven by a calibration dataset	float model + calibration data → w8a8 / w4a16 model	The HTP is an integer engine — float graphs fall back
Profile Job	Runs the compiled model on a real device and instruments it	compiled model → latency, load time, per-layer unit	Prove the graph really landed on the NPU
Inference Job	Runs your own input tensors on the device and returns the outputs	model + input data → output tensors	Check numerical accuracy against the host model
Link Job	Merges several context binaries that share weights into one	several .bin → one multi-graph .bin	LLMs: prompt-processing and token-generation graphs

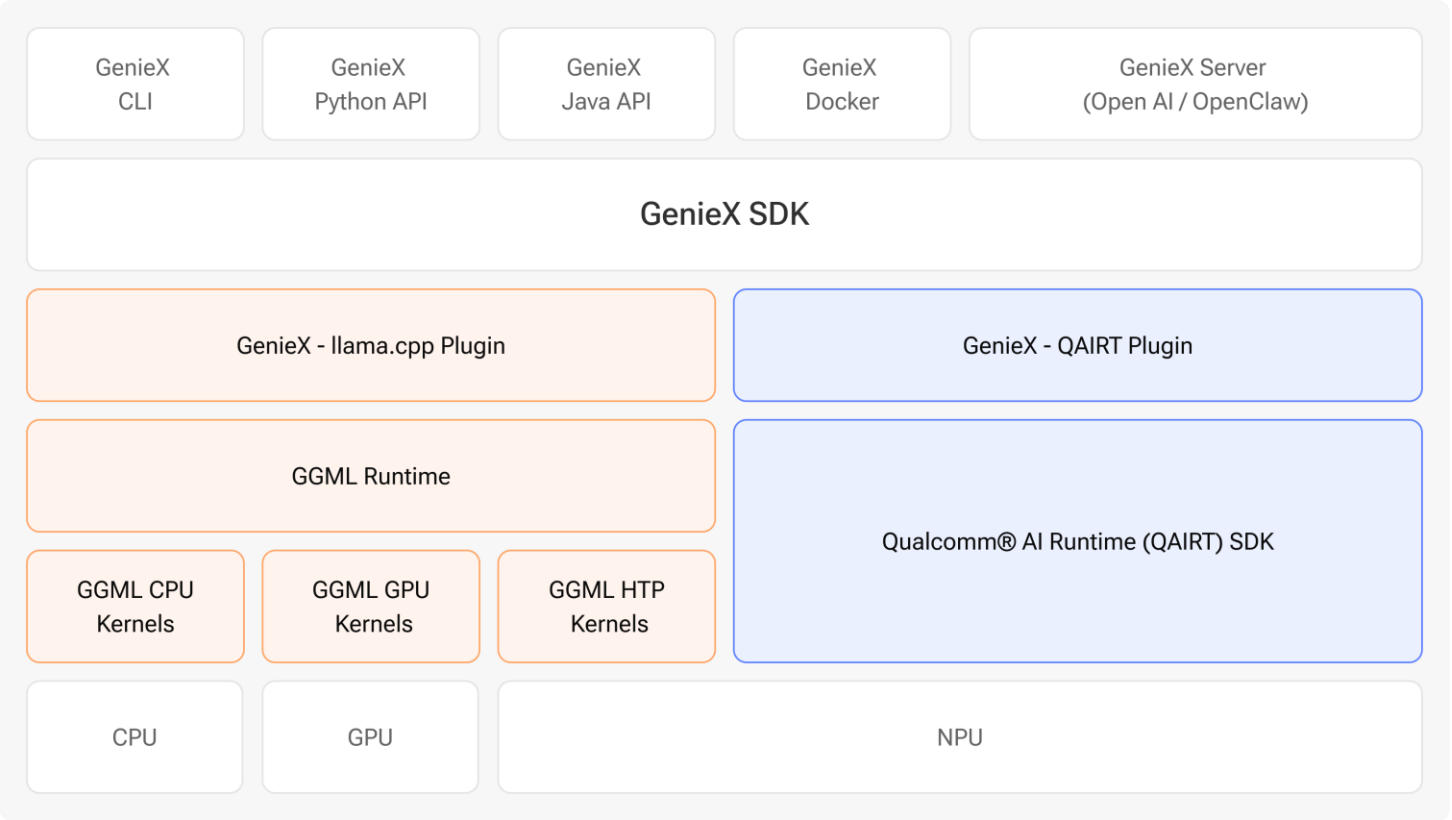
AI Hub Workbench — The Python API

```
pip install qai-hub
qai-hub configure --api_token <YOUR_TOKEN>
import qai_hub as hub
# 1) compile for one device + one runtime
compile_job = hub.submit_compile_job(
    model="yolov8n.onnx",
    device=hub.Device("QCS8550 (Proxy)"),
    input_specs=dict(image=(1, 3, 640, 640)),
    options="--target_runtime qnn_context_binary",
)
target_model = compile_job.get_target_model()
# 2) measure it on real hardware
profile = hub.submit_profile_job(
    model=target_model, device=hub.Device("QCS8550 (Proxy)"))
print(profile.download_profile()["execution_summary"])
# 3) check the numerics on device
infer = hub.submit_inference_job(
    model=target_model, device=hub.Device("QCS8550 (Proxy)"),
    inputs=dict(image=[sample]))
target_model.download("yolov8n.bin")
```

- Jobs are **asynchronous** — submit, keep working, collect results later
- Every job gets a **web page with logs, metrics and the artefacts**
- hub.get_devices() lists the fleet; (Proxy) entries stand in for a chipset family
- The .bin you download is a **QNN context binary** — load it with QAIRT on the target
- Same three calls behind qai-hub-models export
- CI-friendly: pin the QAIRT version in options

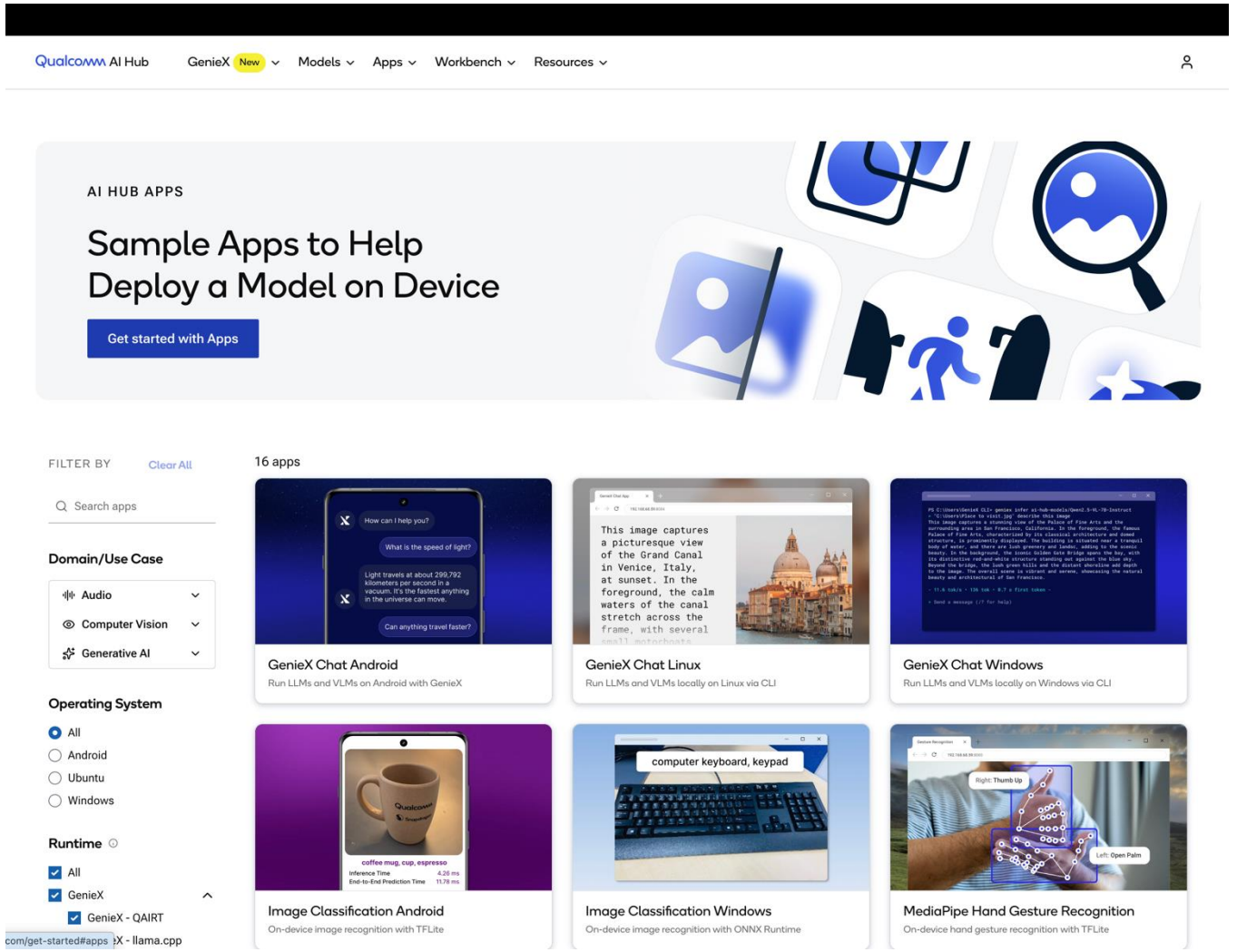
Qualcomm AI Hub GenieX

- On-device runtime for **LLMs and VLMs** on Snapdragon and Dragonwing
- One SDK, two plugins:
  - **QAIRT plugin → Hexagon NPU**
  - **llama.cpp / GGML plugin → CPU, Adreno GPU, HTP**
- Runs both **community GGUF models** and **pre-compiled AI Hub NPU bundles**
- Interfaces: **CLI · Python · Java/Kotlin · Docker · OpenAI-compatible server**
- Platforms: **Windows ARM64 · Android · Linux ARM64 · Dragonwing IoT**
- The successor to **Genie** in the AI Engine Direct SDK — currently developer preview



Qualcomm AI Hub Apps

- **16 open-source sample applications**, full source, ready to build
- **Android** — image classification, object detection, semantic segmentation, super resolution (LiteRT); WhisperKit; ChatApp (Genie); GenieX Chat
- **Windows** — classification, detection, super resolution, SAM3, Whisper, Stable Diffusion (ONNX Runtime); ChatApp; GenieX Chat
- **Ubuntu** — hand gesture recognition, PoseNet, YamNet (LiteRT)
- Each app = **model + pre/post-processing + runtime glue + camera/audio I/O** — which is the part that actually takes the time
- This is the recommended starting point for your own application



5. Application Development

Frameworks for Building Applications on Edge AI Devices

- **Ubuntu / Yocto (Qualcomm Linux)**
  - Languages: **C/C++**, **Python**
  - Inference: **QNN / QAIRT**, **LiteRT (TFLite)**, **onnxruntime-qnn**
  - Generative AI: **GenieX**, **llama.cpp** (OpenCL on Adreno)
- **Windows on Snapdragon**
  - Languages: **C/C++**, **Python**, **C#**
  - Inference: **QNN / QAIRT**, **LiteRT**, **onnxruntime-qnn**
  - Generative AI: **GenieX**, **Genie**
- **Android**
  - Languages: **Kotlin / Java**, native **C/C++ via JNI**
  - Inference: **QNN / QAIRT**, **LiteRT with the QNN delegate**, **onnxruntime-qnn**
  - Generative AI: **GenieX**, **Genie**, **LiteRT-LM**

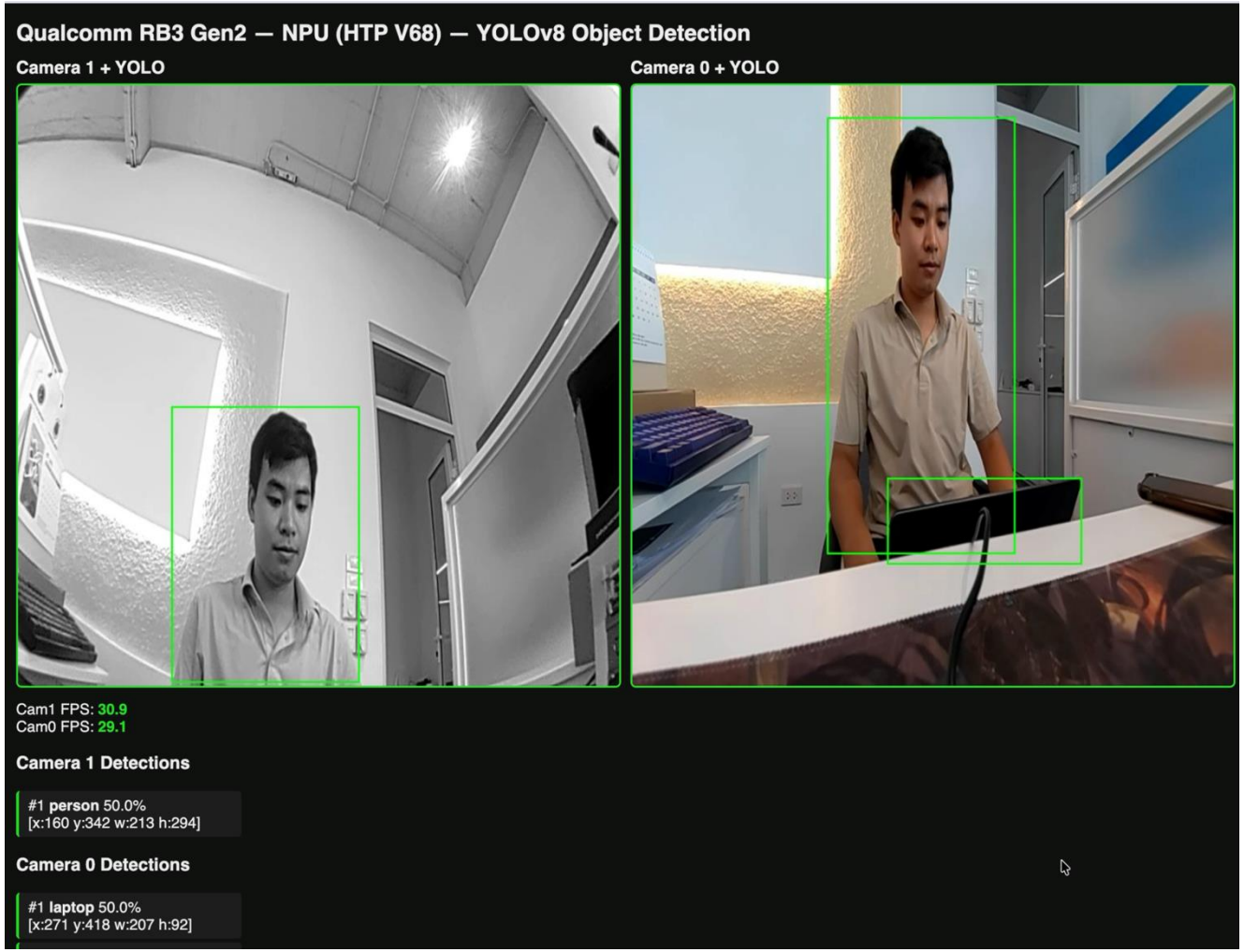
Reference Workflow — the Shape of Every Project

0 · setup	1 · pull model	2 · compile	3 · app	4 · run
QAIRT SDK, aarch64 cross-toolchain, Python venv, AI Hub API token, board access over SSH/ADB	Fetch weights, tokenizer and assets; export to ONNX or .pt2 with pre/post-processing embedded where possible	Submit compile + quantize jobs to AI Hub Workbench for the target SoC → QNN context binary (.bin)	Write the application: create the context once, reuse it for every frame or token; wire camera / audio I/O	Deploy to the board, run, profile on device, iterate on precision and graph partitioning

- All three projects that follow use exactly this layout — the directory names **are** the pipeline: 0-setup/, 1-pull-model/, 2-compile/, 3-app/, 4-run/.
- **Stage 2** is where AI Hub replaces a toolchain you would otherwise install, version and maintain yourself — and it is the only stage that needs the cloud.
- **Stage 3** is where the performance is won or lost: create the QNN context **once** at start-up and reuse it, rather than per frame or per token.
- Keeping the stages separate is what makes the project reproducible — a new QAIRT release means re-running stage 2 only.

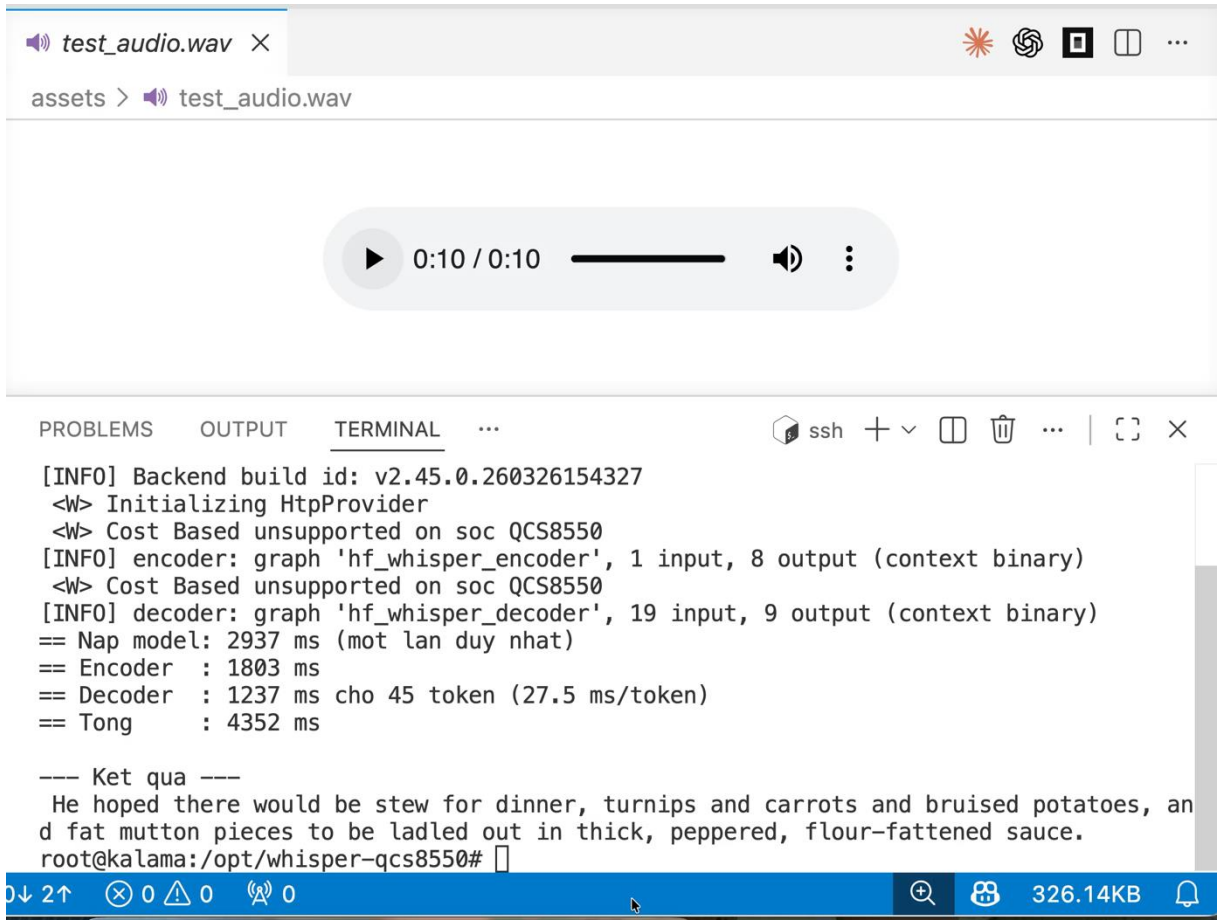
Computer Vision — YOLOv8 Object Detection on the NPU

- **Board:** RB3 Gen 2 — **QCS6490 / QCM6490**, Hexagon **V68** HTP (12 TOPS INT8), Qualcomm Linux aarch64, kernel 6.6
- **Model:** YOLOv8n exported to ONNX with post-processing embedded, quantized **w8a8** through AI Hub → QNN context binary
- **Application:** C + **GStreamer** CSI camera capture → NPU inference → MJPEG web stream
- Ships a **CPU vs GPU vs NPU benchmark** (Python + LiteRT) so the backend choice is measured, not assumed
- Source code: http://github.com/mxngocqb/Qualcomm_Robotics_NPU



Speech Recognition — Whisper on the NPU

- **Board: QCS8550**, Hexagon **V73** HTP — 48 TOPS INT8
- **Model: Whisper Large V3 Turbo**, compiled through AI Hub into **two QNN context binaries**: encoder and decoder
- **Pipeline:** audio → log-mel → **encoder on NPU (once)** → KV cache → **decoder on NPU (per token)** → text
- **NPU-only:** the model exists only as QNN context binaries — there is no LiteRT/ONNX fallback path
- Source code: https://github.com/mxngocqb/Whisper_QNN



Large Language Models — GenieX on QCS8550

- **Board:** QCS8550 (Kalama) — Kryo CPU (X3 @3.19 GHz + A715/A710 @2.80 GHz + 3× A510 @2.02 GHz), **Hexagon HTP v73**, **Adreno 740** (OpenCL 3.0), 14 GB RAM
- **Stack: GenieX 0.3.17 / QAIRT 2.45**, Ubuntu 22.04 arm64 on an Android kernel, Docker deployment; llama.cpp built with OpenCL for the GPU path
- **Model: Qwen3-0.6B GGUF (Q4_0)**, served over an **OpenAI-compatible API** — GenieX on :18181, llama.cpp on :18182
- Source code: https://github.com/mxngocqb/GenieX

HEXAGON NPU SDK

1. **Hexagon SDK & NPU Kernel Programming**
   Example tutorial for direct interaction with Qualcomm Hexagon DSP/NPU kernels:
   https://github.com/taowen/hexagon-tutorial
2. **Post-Quantum Cryptography on Qualcomm NPU**
   Our collaborative work with **Dr. Ta Duy Hoang**:
   https://github.com/Hiiamming/pqc-hqc-npu

6. Production

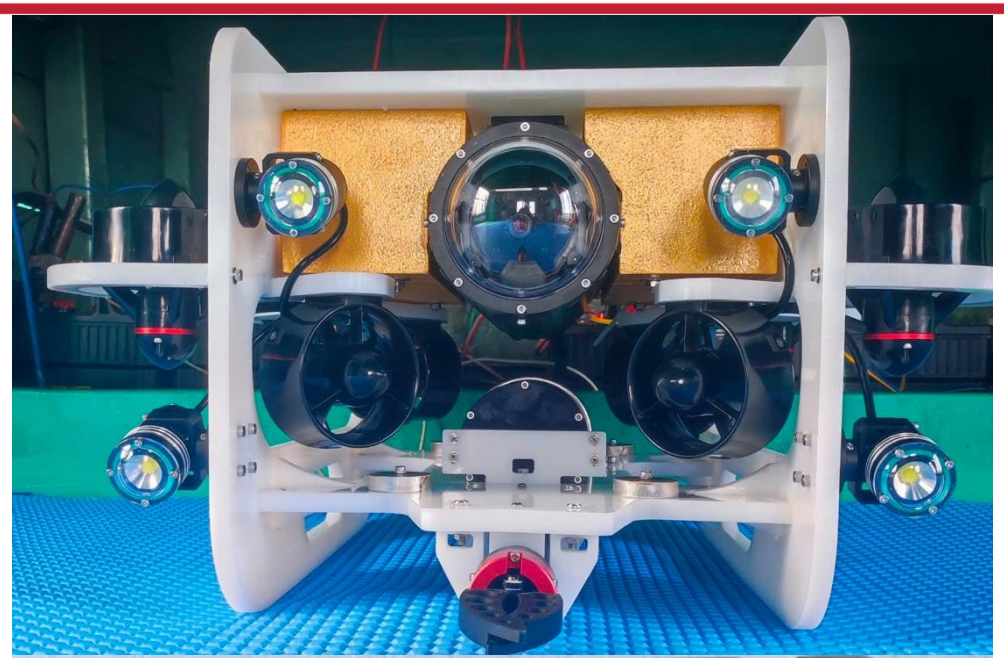
BKMeeting Device

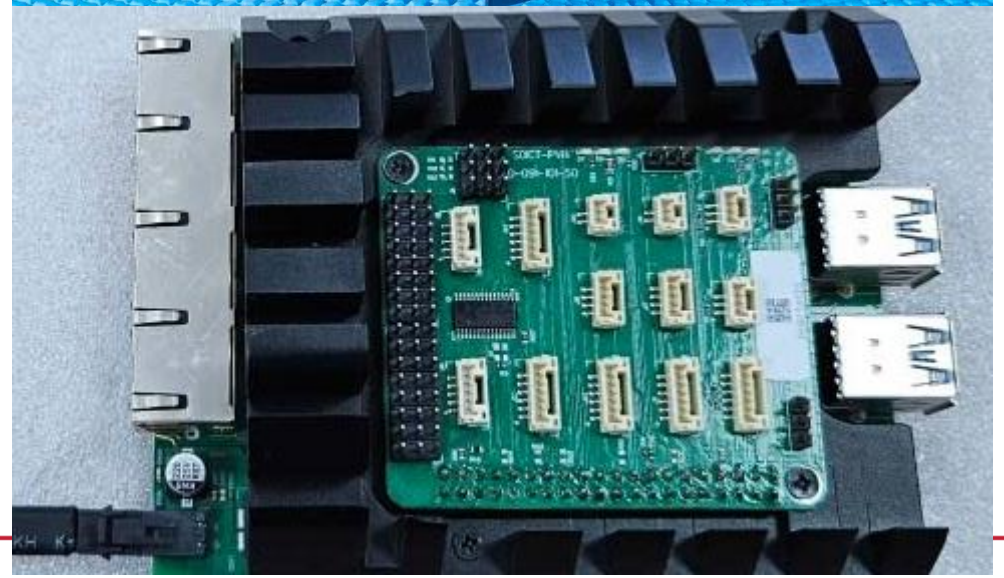
- **Platform:** QCS8550 — 48 TOPS INT8
- **Integrated:** microphone array, camera
- **AI features:**
  - Face recognition and attendance
  - Automatic speech recognition (Whisper)
  - On-device SLM: Qwen3-4B, Llama 3B, Gemma 3 4B

PROV-01 Underwater Vehicle

- The 8550 board is a high-performance edge computing device built on the **KIMQ 8550** platform around the **Qualcomm QCS8550** chipset.

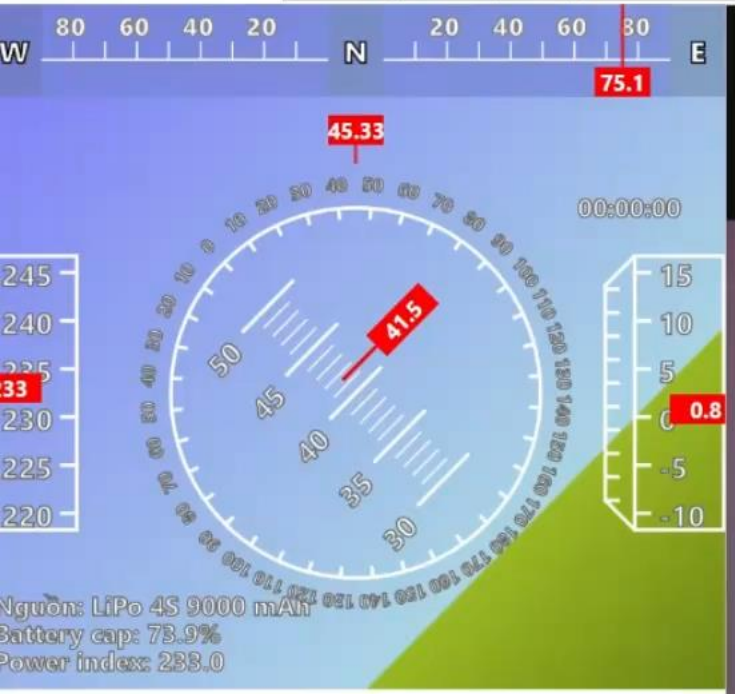
- It carries a **6-core Kryo CPU** and a dedicated **NPU delivering 48 TOPS (INT8)** or **12 TFLOPS (FP16)**.



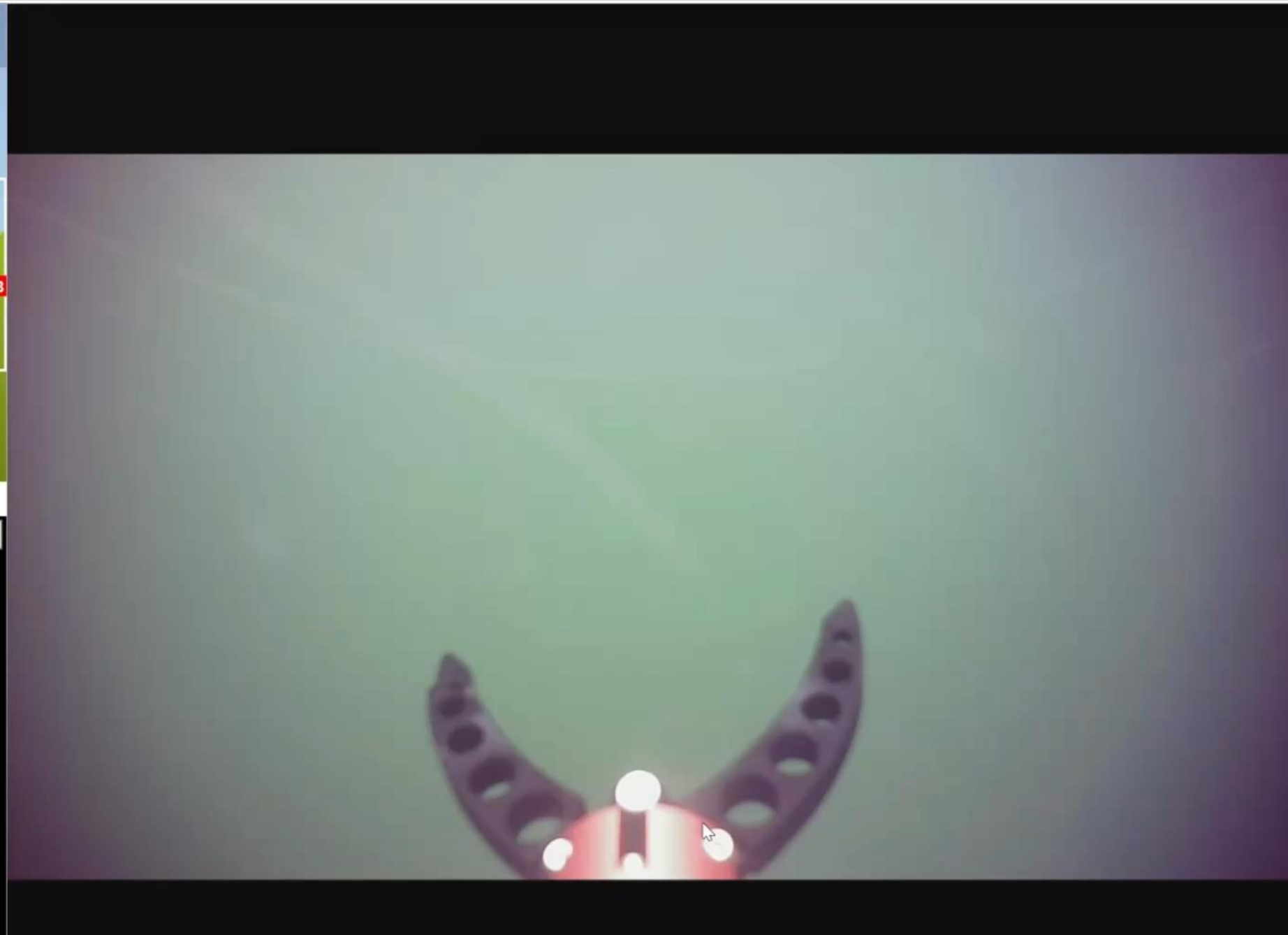


PVN-BK-ROV GCS VERSION 2025.001

Màn hình điều khiển | Hiển thị Sonar | Thông tin chương trình

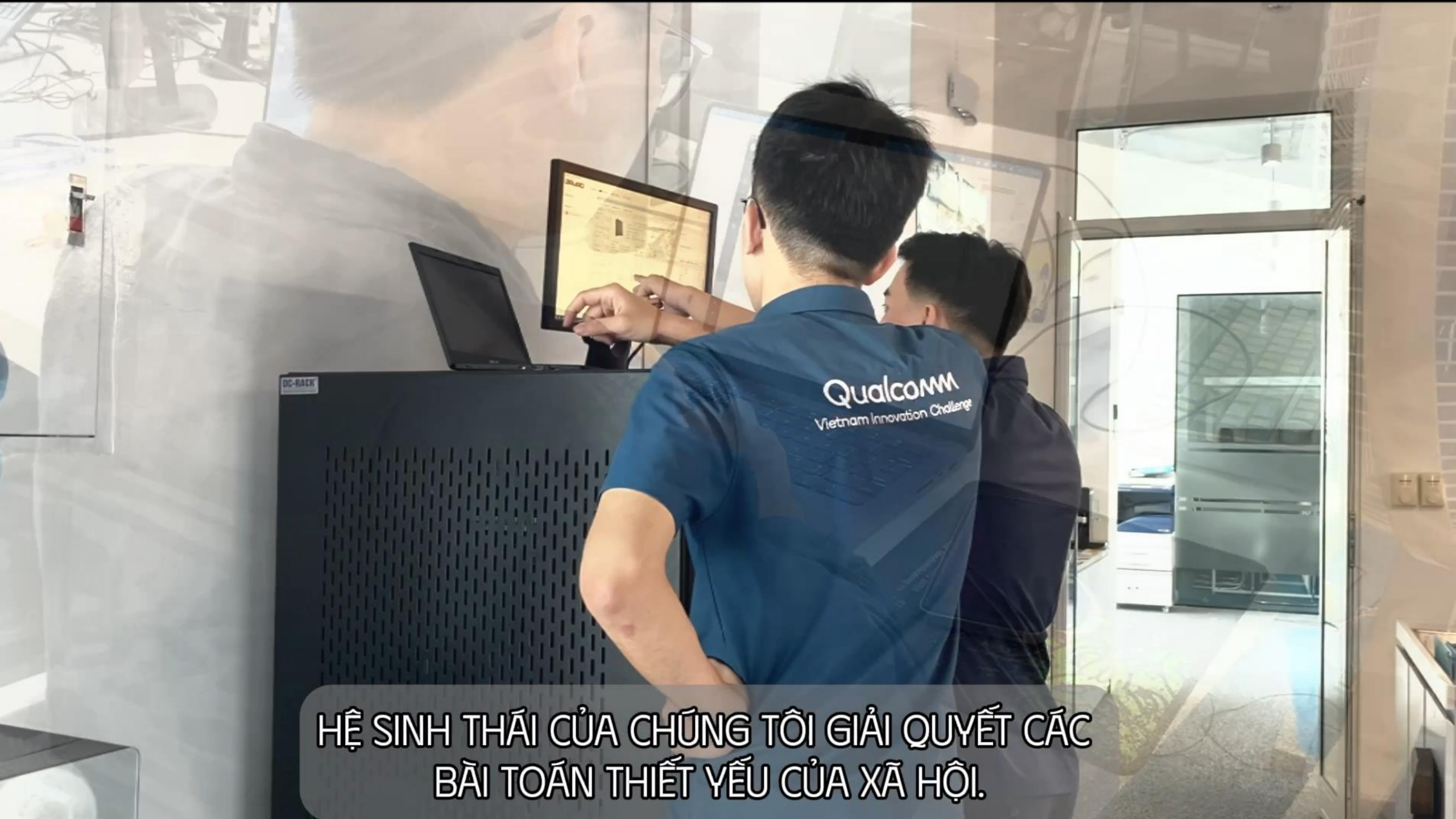


+ -



BKACS and GSStation

- **QCS6490 — 12 TOPS INT8**
- **BKACS** — multi-modal biometric access control device:
  - Face recognition
  - Fingerprint
  - National ID card reading
- **GSStation** — fingerprint enrolment station



HỆ SINH THÁI CỦA CHÚNG TÔI GIẢI QUYẾT CÁC
BÀI TOÁN THIẾT YẾU CỦA XÃ HỘI.

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !

6. Hands - on

Qualcomm AI Hub Workbench

1. Create account on Qulacomm AI Hub Workbench
2. Get API Token
3. Interact with Workbench via the Python library

   Example: https://colab.research.google.com/drive/1RI3ACc8n4wWi6EX00ut2mX5VexoiDF34?usp=sharing

Connect to the Device via RJ45

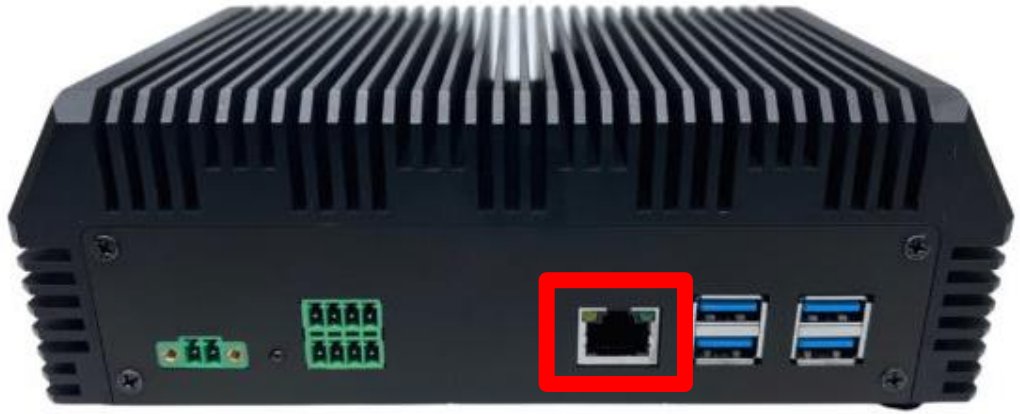
1. Connect the device to the network using the RJ45 Ethernet port

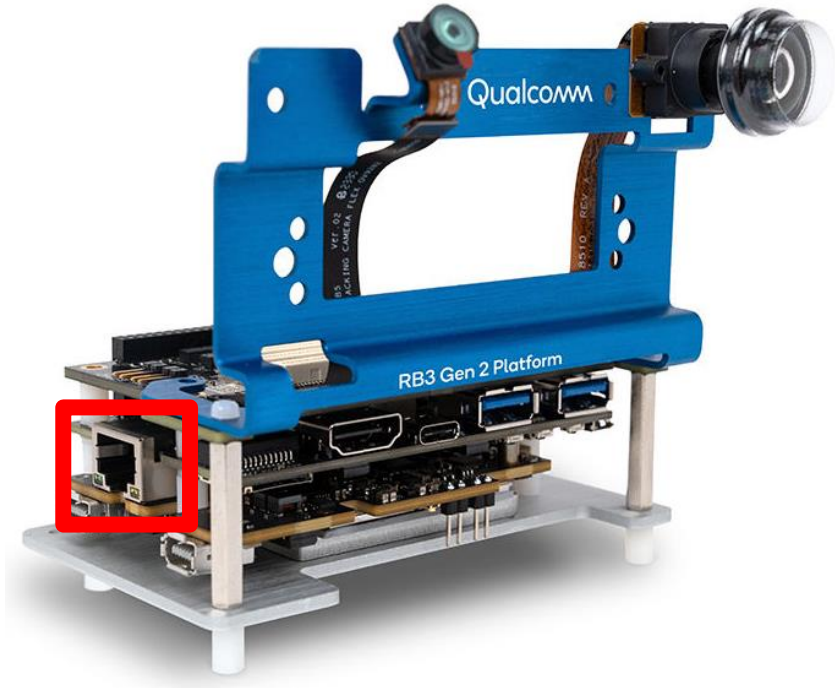
2. Use Advanced IP Scanner to scan the network and find the device's IP address. Match the scanned device with the MAC address printed on the device box.

3. Connect to the device via SSH:

   ssh root@<ip_address>

   passwd: oelinux123





Connect to the Device via ADB

1. Connect the device to your computer using a USB Type-C cable.
2. Install the Qualcomm Userspace Driver if required: https://softwarecenter.qualcomm.com/catalog/item/Qualcomm_Userspace_Driver
3. Install Android Platform Tools (ADB).
    1. Windows: https://dl.google.com/android/repository/platform-tools-latest-windows.zip
    2. Linux: sudo apt install adb
    3. Macos: brew install android-platform-tools
4. Open an ADB shell: adb shell