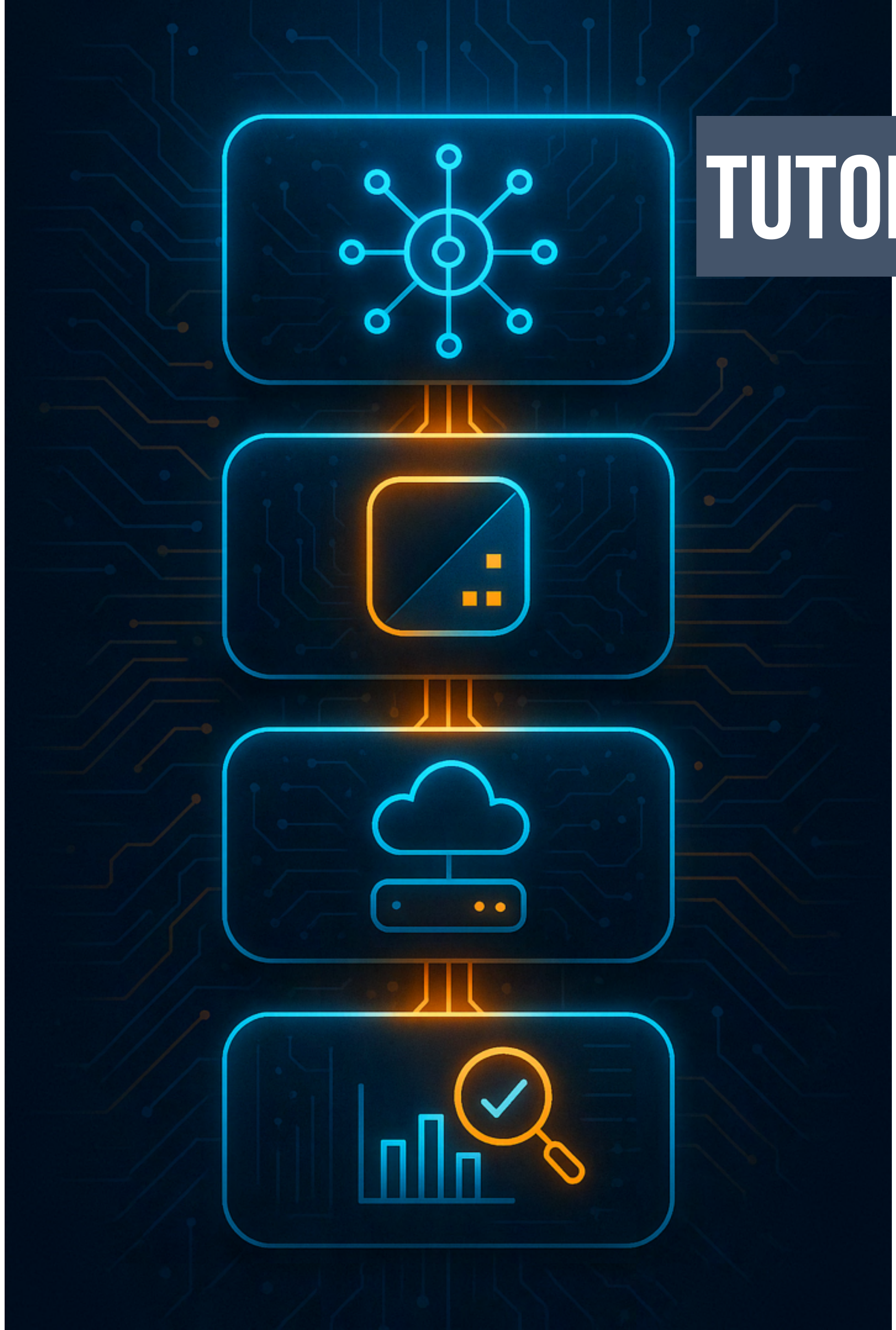
# MODEL DEPLOYMENT FOR EDGE AI

**CVPR 2025 Tutorial**

**The IEEE/CVF Conference on Computer Vision and Pattern Recognition 2025**

Nashville, TN, USA



# TUTORIAL AGENDA

1 Model Quantization with SNPE

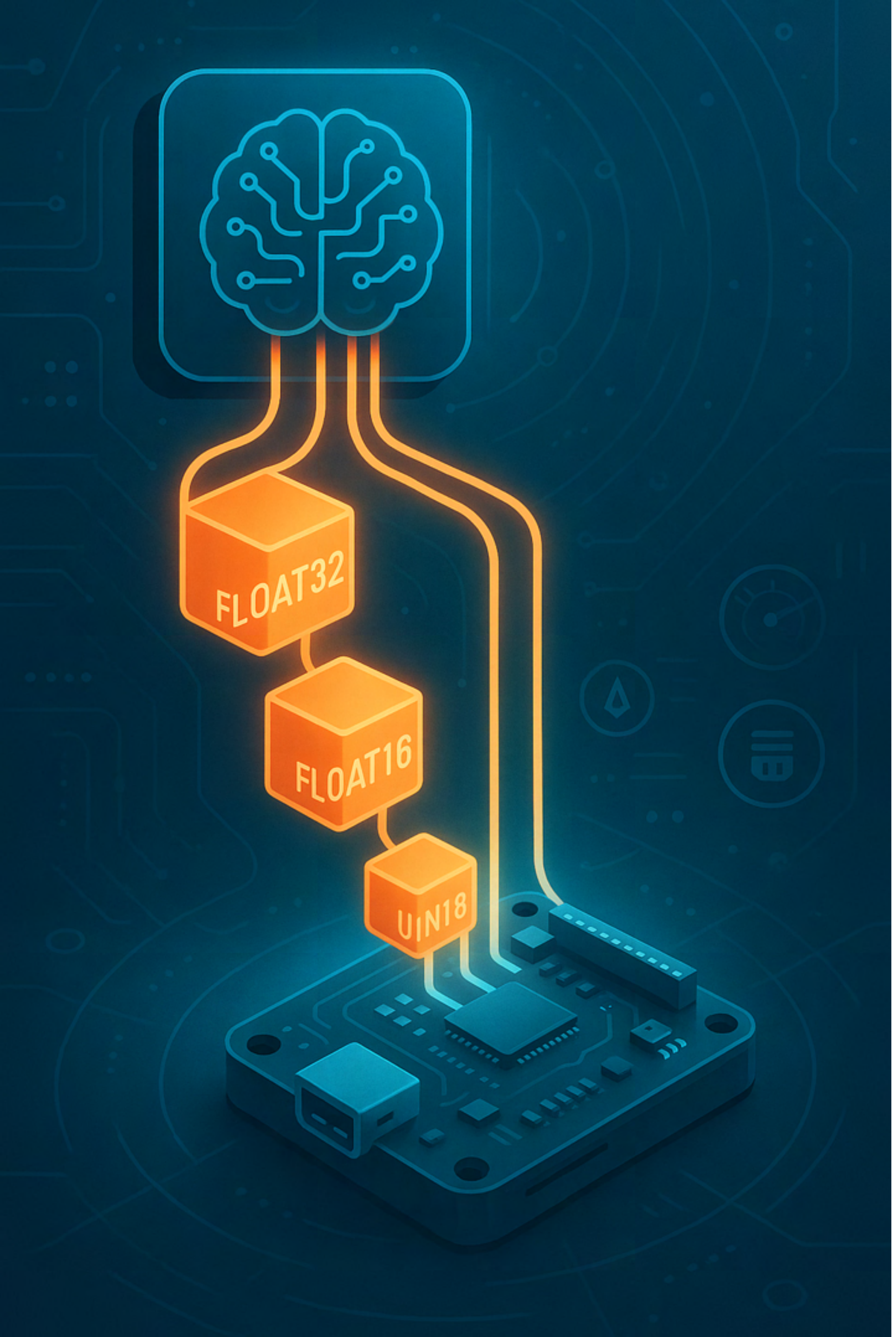
2 Qualcomm Neural Processing SDK

3 Model Deployment on Android

4 Hand Gestures Recognition Model

5 TensorFlow Lite on Android

# MODEL QUANTIZATION WITH SNPE




# INTRODUCTION TO QUANTIZATION

## Overview

Quantization is a key technique in model optimization for Edge AI, where the precision of model parameters and activations is reduced to accelerate inference and lower memory usage, often with minimal impact on accuracy.

### Float32

Offers high precision but requires more memory and computing power.

### Float16

Reduces model size and speeds up inference on hardware with native FP16 support, such as GPUs.

### Int8

Ideal for edge devices but may require careful calibration to preserve accuracy.

# AI PROCESSING UNITS

Hardware

Edge AI devices incorporate diverse processing units, each optimized for different types of computations. Quantization enables these units to run AI models more efficiently by aligning data precision with hardware capabilities.

### CPU

A general-purpose processor ideal for model control logic and less demanding inference.

### GPU

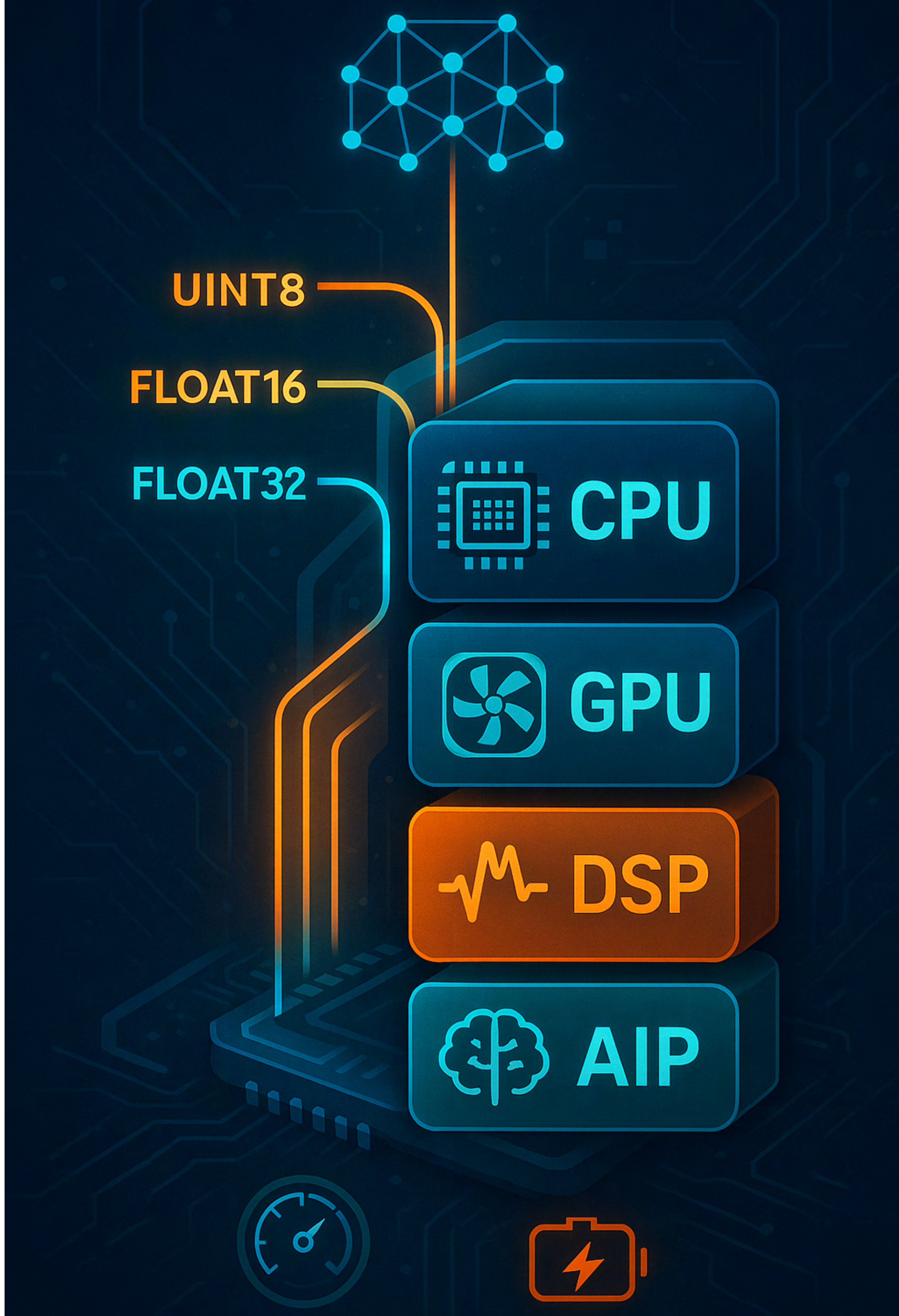
Optimized for parallel floating-point operations, well-suited for CV models with large matrix computations.

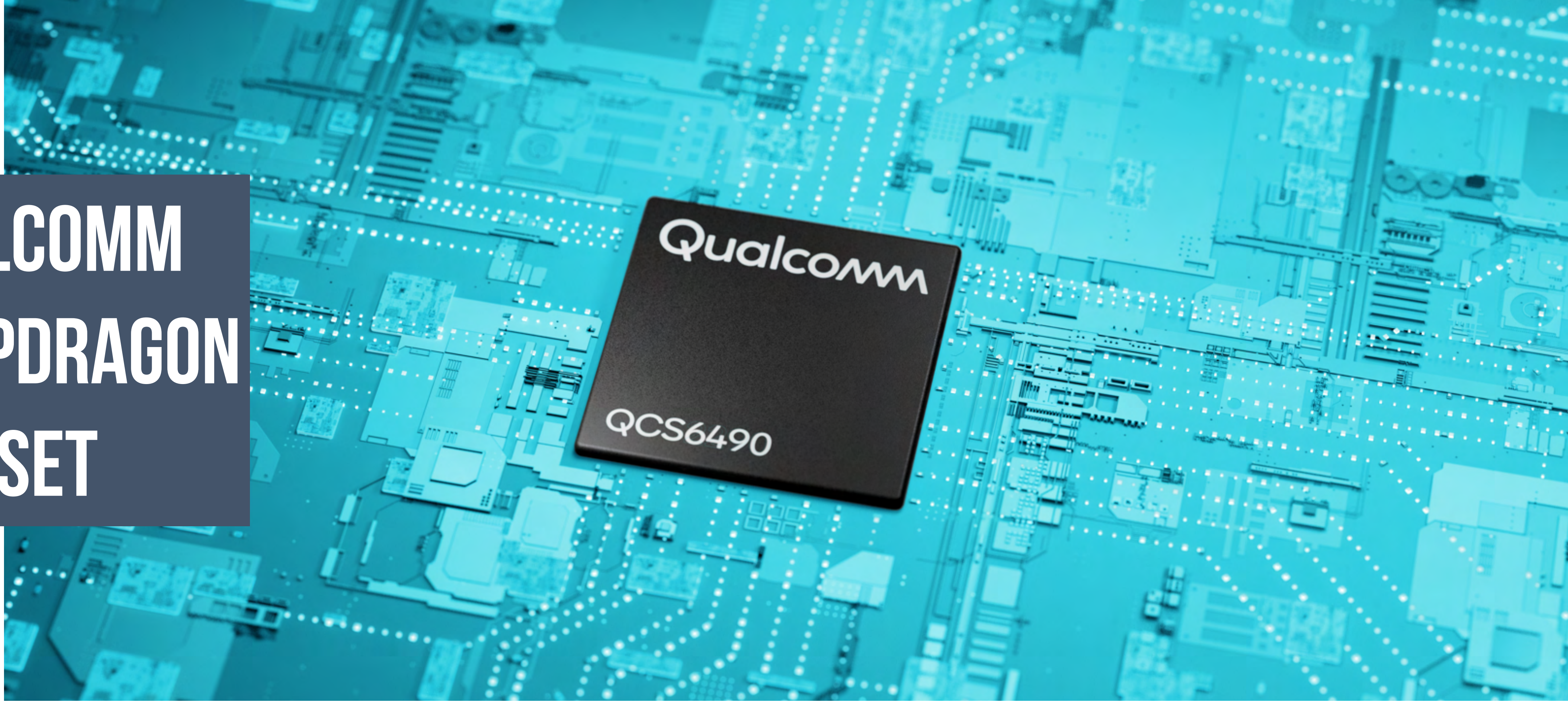
### DSP

Designed for low-power execution of quantized models (especially UINT8); provides efficient fixed-point computation for always-on tasks.

### AIP

Dedicated accelerator explicitly built for deep learning. It delivers high-throughput inference with minimal latency across quantized and mixed-precision models.

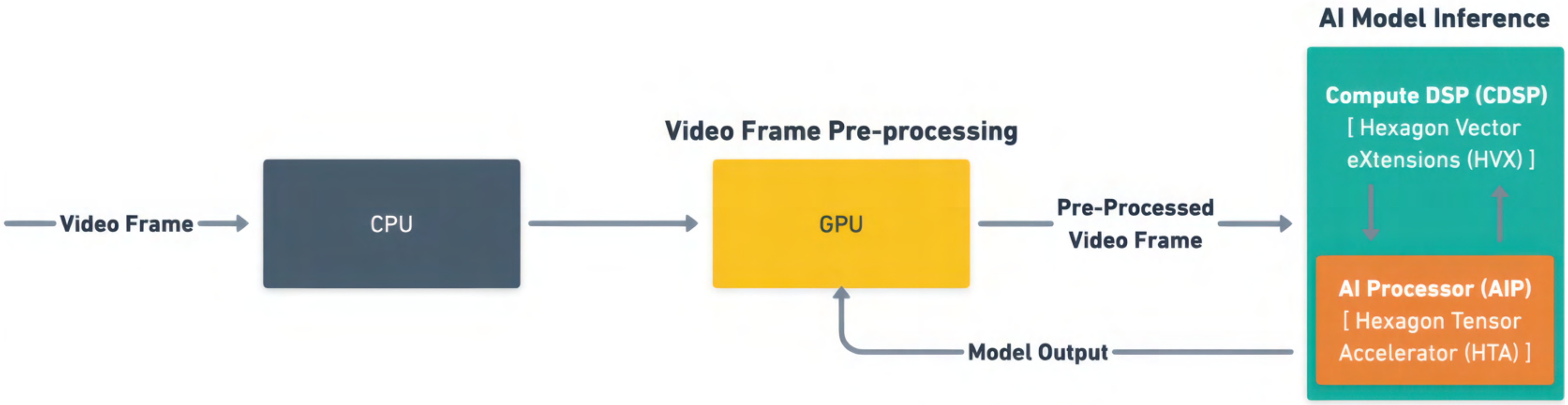





# QUALCOMM SNAPDRAGON CHIPSET

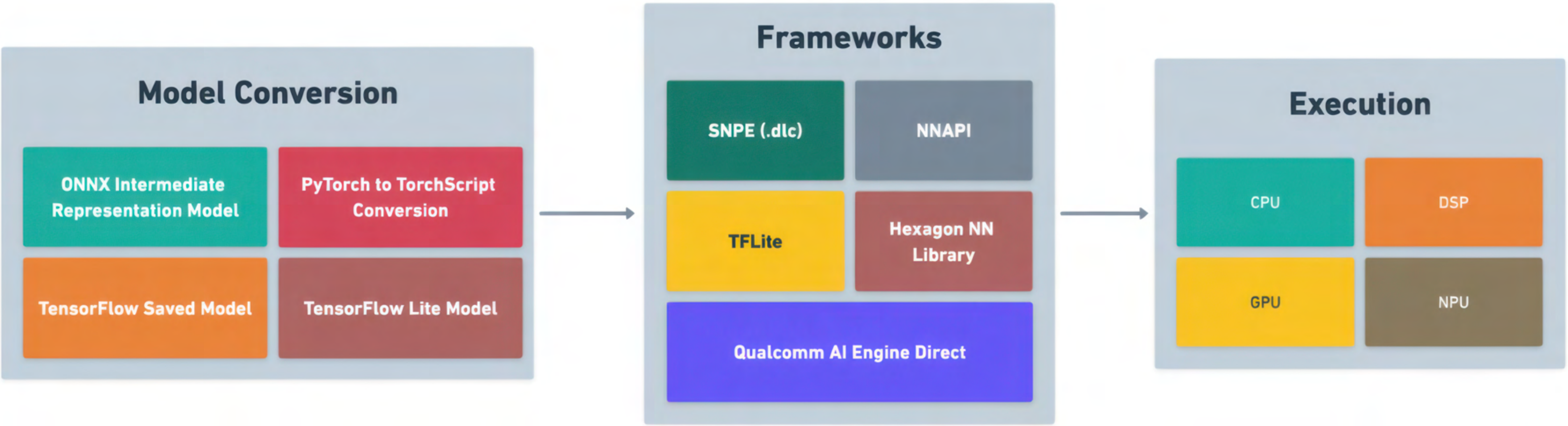
The Qualcomm QCS6490 is a high-performance chipset designed for intelligent edge computing. It integrates multiple processing engines—CPU, GPU, DSP, and a dedicated AI Processor (AIP)—allowing for efficient on-device AI inference. This makes it ideal for applications in robotics, industrial IoT, and smart cameras. Its compatibility with quantized models and optimization through the SNPE SDK makes it a powerful platform for deploying real-time AI models at the edge.

# QUALCOMM INFERENCE END-TO-END
## Workflow



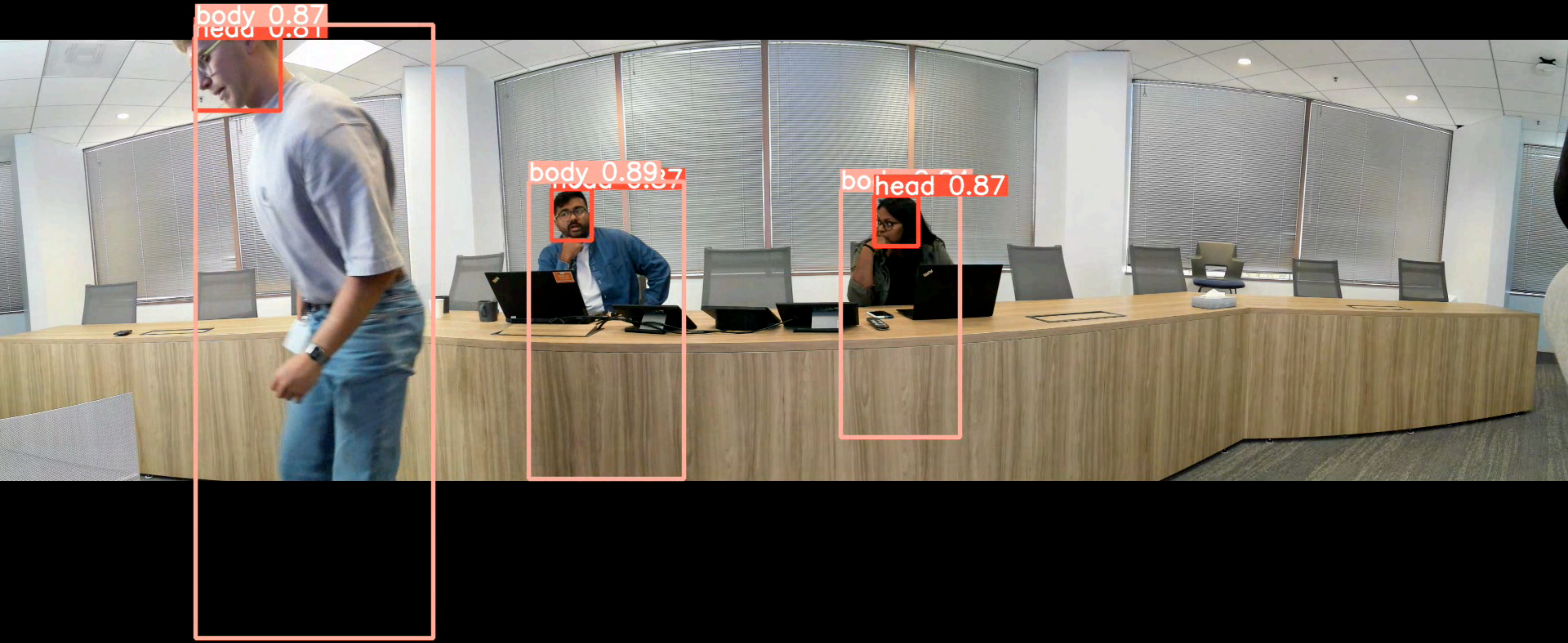
# WORKFLOW FOR MODEL DEPLOYMENT

## Deploying Machine Learning Models on Qualcomm Hardware



# MODEL IN ACTION

Jabra PanaCast 50 VBS



# INTELLIGENT MEETING SPACES

## Jabra PanaCast 50

# DEVICES WITH QCS6490

## Model Deployment

### I/O and Interfaces

Offers GPIO, UART, SPI, I2C, and USB interfaces for hardware debugging, prototyping, and integration with sensors and peripherals.

### Edge AI and DSP

Supports AI model deployment using the SNPE SDK, providing direct access to CPUs, GPUs, DSPs, and NPUs.

DSP

### Headless Configuration

Typically does not include a built-in front camera or display, focusing instead on edge deployment scenarios.

Qualcomm Dev Board

Android Fairphone 5 5G

### Display and Cameras

Comes with high-resolution screens, front/rear cameras, and touch input, making it ideal for real-world testing of vision models and user interfaces.

### Power Management

Includes mobile-optimized thermal and power regulation, enabling sustained AI workloads within user-friendly temperature thresholds.

### Real-World Application

Allows testing in actual user contexts, making it ideal for validating usability and responsiveness of AI models.


# ONNX VS DLC

## Model Formats

ONNX (*Open Neural Network Exchange*) and DLC (*Deep Learning Container*) are two key model formats in the Edge AI deployment pipeline. Converting from ONNX to DLC is a crucial step in model optimization.

### ONNX

Open format that enables models trained in different DL frameworks to be converted and reused across various tools and platforms.

### DLC

Qualcomm's optimized model format for deployment with SNPE, tailored for efficient execution on DSP, GPU, and AIP.

### Conversion Pipeline

Models are exported to ONNX and then converted to DLC using SNPE tools for quantization and hardware-specific tuning.

### Execution Compatibility

ONNX is used in general-purpose runtimes, while DLC is mandatory for leveraging full acceleration on Qualcomm devices.

# QUALCOMM NEURAL PROCESSING SDK

# SNPE SDK

## Qualcomm Neural Processing SDK for AI

The SNPE SDK is Qualcomm's official toolkit for deploying AI models on Snapdragon-powered devices.

### HOW TO OPTIMIZE AN AI MODEL USING SNPE V2.34.0.250424?

**CONVERSION TOOLS**

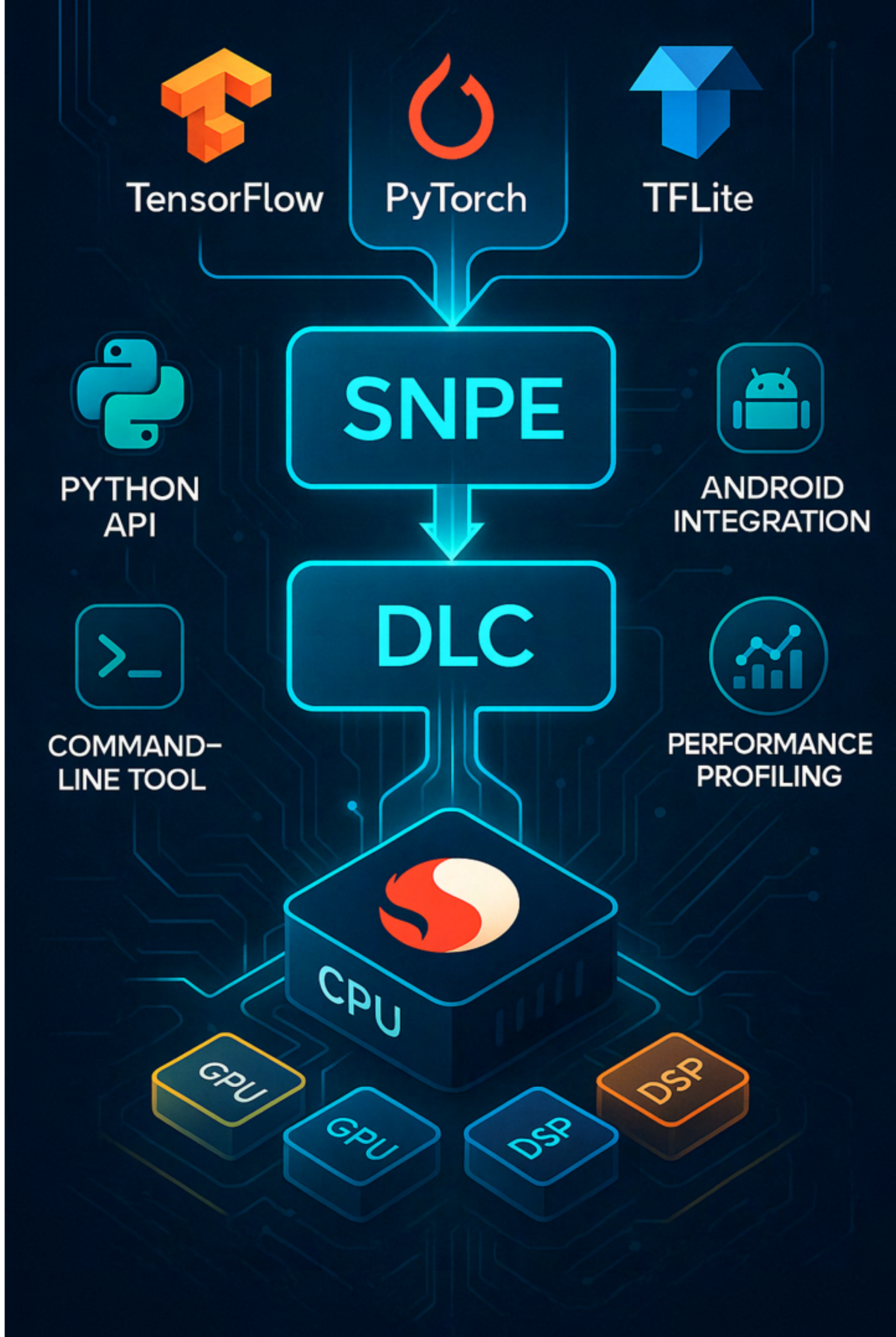
Converts models from ONNX/TF to DLC, with optional quantization for UInt8 inference and graph optimizations.

**COMMAND-LINE**

Offers interfaces for automation, scripting, and integration into custom workflows and CI/CD pipelines.

**PERFORMANCE PROFILING**

Tools for benchmarking inference speed, memory usage, and layer-by-layer diagnostics.



# SNPE SDK COMMANDS

https://www.qualcomm.com/developer/software/neural-processing-sdk-for-ai


1 **snpe-onnx-to-dlc**
Converts ONNX models to DLC (a widely used format-agnostic format).

2 **snpe-pytorch-to-dlc**
Converts PyTorch models into DLC format for Snapdragon inference.

3 **snpe-tensorflow-to-dlc**
Converts TensorFlow models to DLC.

4 **snpe-tflite-to-dlc**
Converts TFLite models to DLC.

5 **snpe-dlc-info**
Displays information about a DLC model.

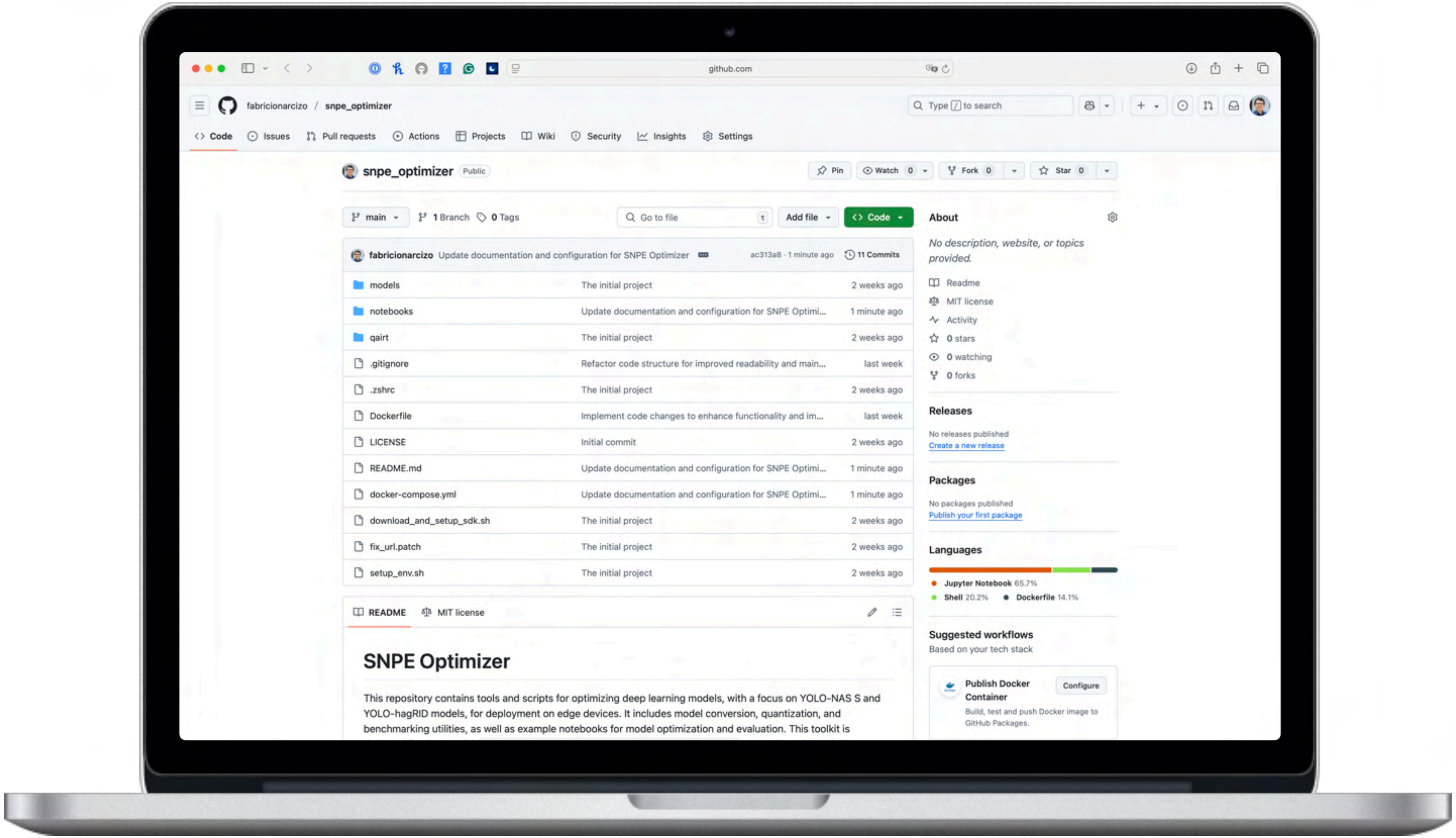
6 **snpe-throughput-net-run**
Measures inference speed and throughput.

7 **snpe-dlc-diff**
Compares two DLC models to detect changes.

8 **snpe-dlc-viewer**
GUI-based tool to inspect DLC structure.

# SNPE OPTIMIZER PLATFORM

https://github.com/fabricionarcizo/snpe_optimizer



# SNPE OPTIMIZER

## Local Optimization

It contains tools and scripts for optimizing DL models for deployment on edge devices. It includes model conversion, quantization, and benchmarking utilities, as well as example notebooks for model optimization and evaluation.

### Docker

SNPE Optimizer provides pre-configured Docker images for development and conversion.

### Linux (x86_64)

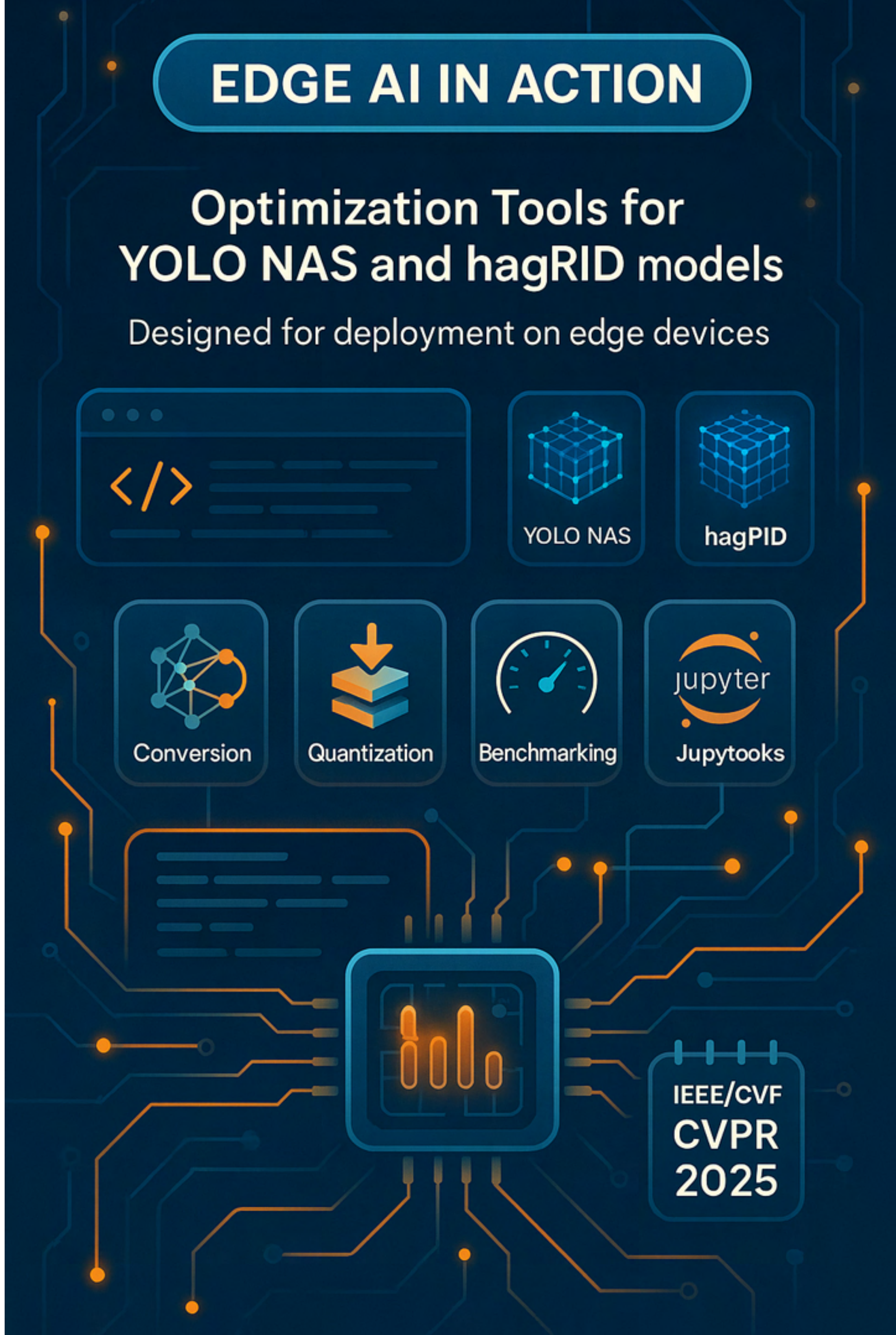
Official support for Ubuntu-based systems ensures stability and compatibility with the SNPE toolchain.

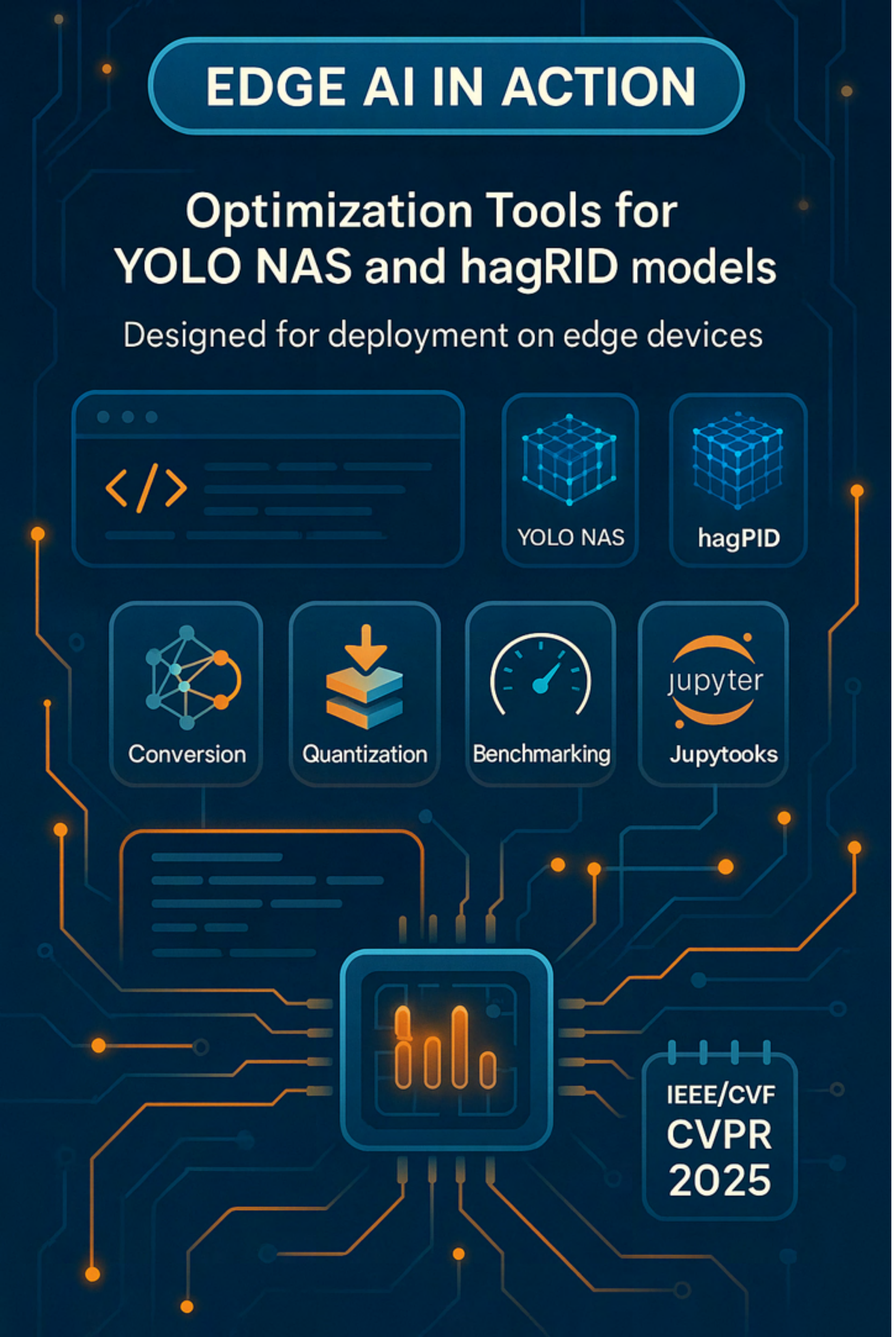
### Intel CPU Architecture

Required for compatibility with SNPE tools; ARM-based developer machines are not supported for model conversion.

### Android NDK

Necessary to build and deploy Android-native binaries for executing models on actual devices using the SNPE runtime.






# SNPE OPTIMIZER

## Folder Structure

The SNPE Optimizer platform uses volumes to persist data stores implemented by the container engine. These are the primary volumes used in this platform:

### models

Pretrained and optimized model files (ONNX, DLC, TFLite, and TensorFlow SavedModel formats).

### notebooks

Jupyter notebooks for model optimization, export, and evaluation. Contains validation and raw data folders.

### qairt

SNPE SDK and tools for quantization and inference on the Qualcomm platform (QCS6490).

# MINICONDA ENVIRONMENTS

## Python Virtual Environments

Responsible for model optimization routine.

It requires specific versions of the Python packages.

Docker compose manages the environmental variables.

**SNPE**

The primary Python package is super-gradients.

We create a patch to correct the URL from pre-trained models.

**MODEL-ZOO**

# JUPYTER NOTEBOOKS

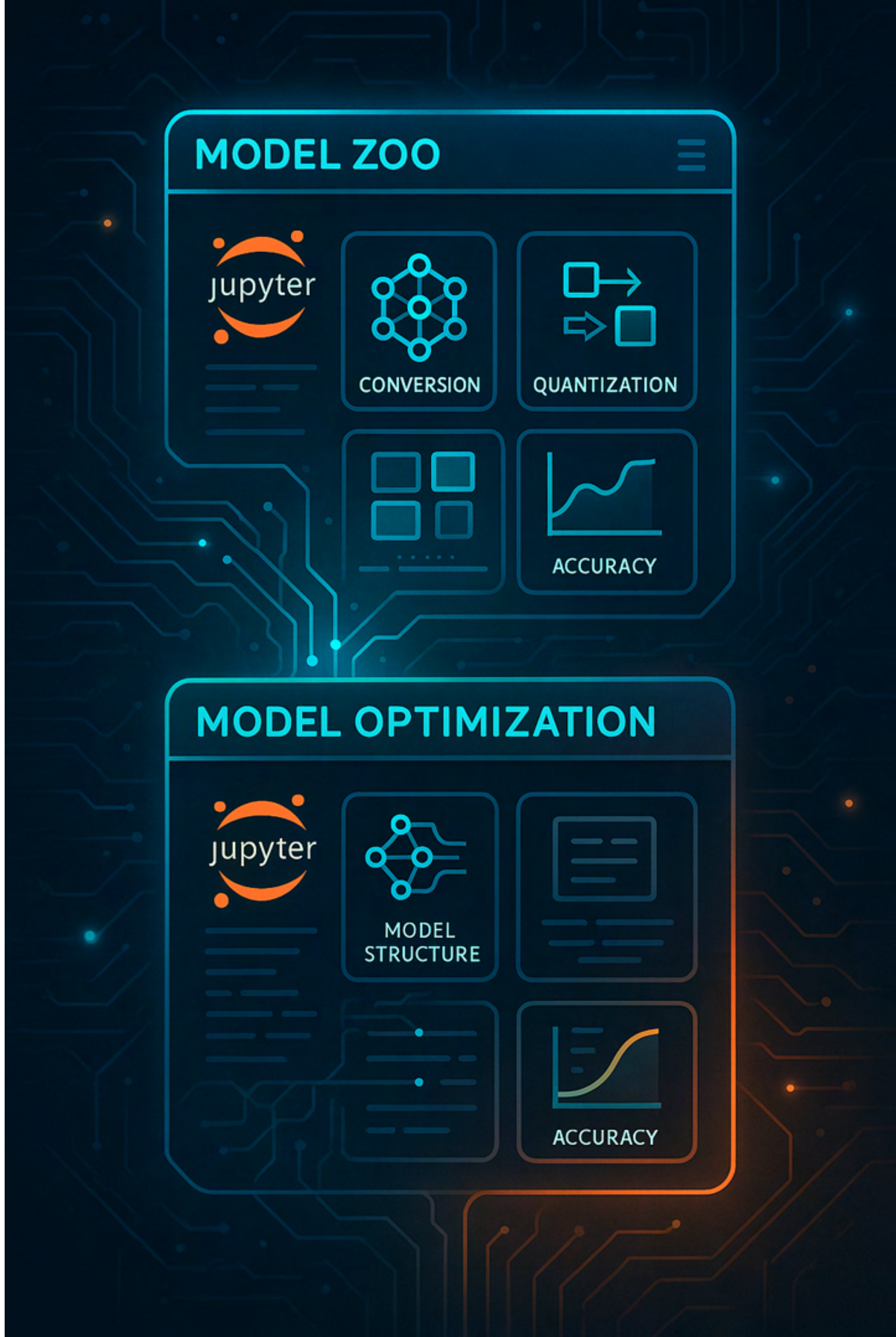
## SNPE Development Workflow

### MODEL OPTIMIZATION NOTEBOOK (PORT 8888)

Guides users through quantization, format conversion (ONNX → DLC), runtime selection, and performance tuning using SNPE APIs.

### MODEL ZOO NOTEBOOK (PORT 8889)

Provides a curated set of pretrained models (e.g., YOLO-NAS S) with loading, testing, and export routines for supported frameworks.



# SNPE OPTIMIZER SETUP

## Overview

**05 — Model Optimization**

Access the model optimization:

`http://localhost:8888/notebooks/model_optimization.ipynb`

**04 — Model Zoo**

Access the model zoo notebook:

`http://localhost:8889/notebooks/model_zoo.ipynb`

**03 — Docker Container**

Build and run the docker compose:

```
$ docker compose build
$ docker compose up -d
```

**02 — SNPE SDK**

Download and setup the SDK:

```
$ bash download_and_setup_sdk.sh
```

**01 — GitHub Clone**

Clone the GitHub repository:

```
$ gh repo clone fabricionarcizo/snpe_optimizer
```

# MODEL OPTIMIZATION STEPS

## Overview

### Export ONNX

Export a pre-trained model to the ONNX format, typically by using a tool like PyTorch ONNX exporter or a similar tool specific to your model's framework.

### Download Dataset

We will use the dataset to calculate the ranges for the quantization parameters.

### Model Conversion

Use the `snpe-onnx-to-dlc` conversion tools to convert a non-quantized model into a non-quantized DLC file.


### Hardware-Specific Graph

Use the `snpe-dlc-graph-prepare` tool to create a cache that contains an execution strategy to execute the optimized model DLC on an HTP hardware.

### Check Chipset Model

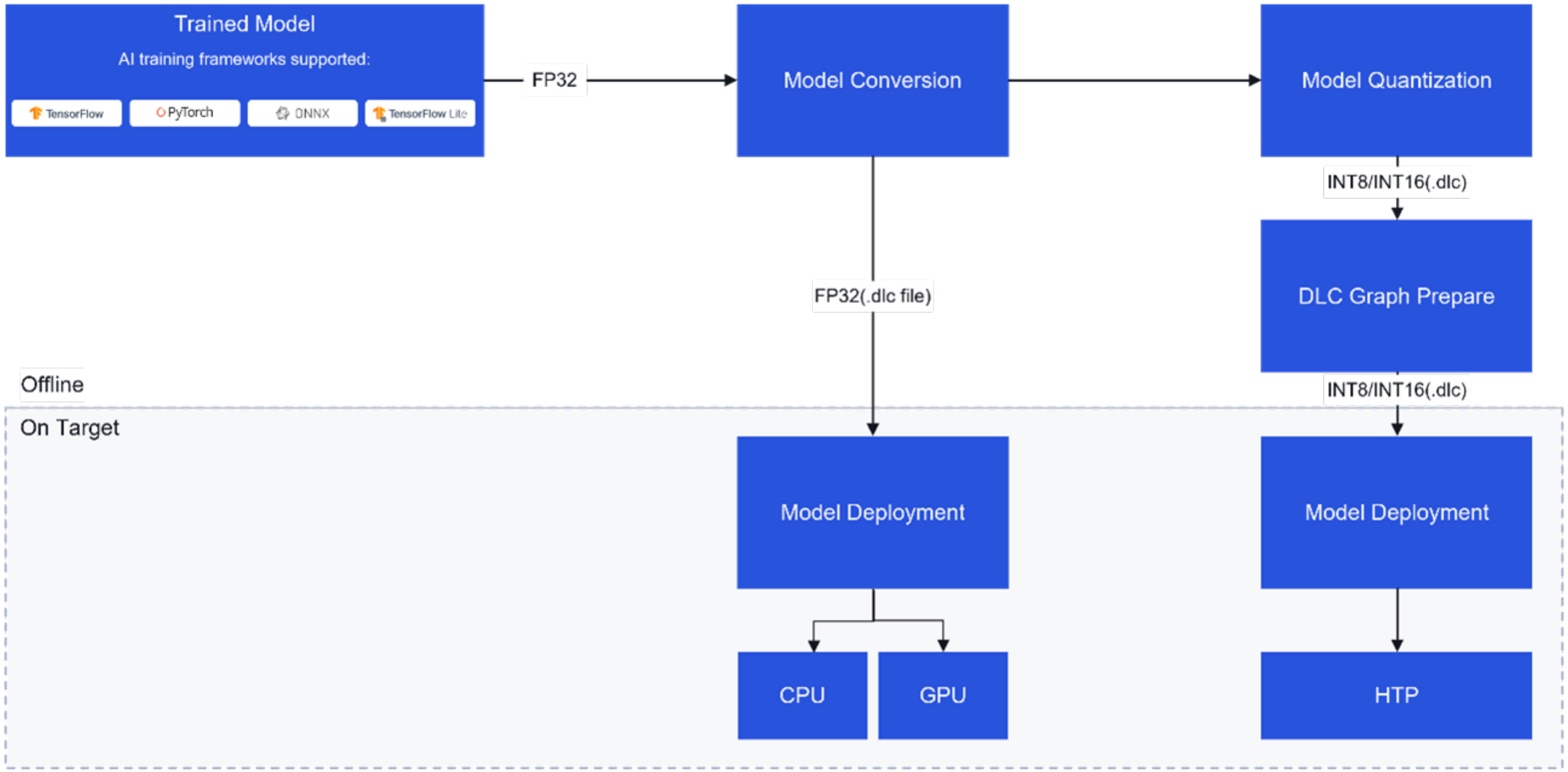
Get the chip name of the Android device using the `adb` command.

### Model Quantization

Use the `snpe-dlc-quantize` tool to quantize the model to one of the supported fixed-point formats (uint8).

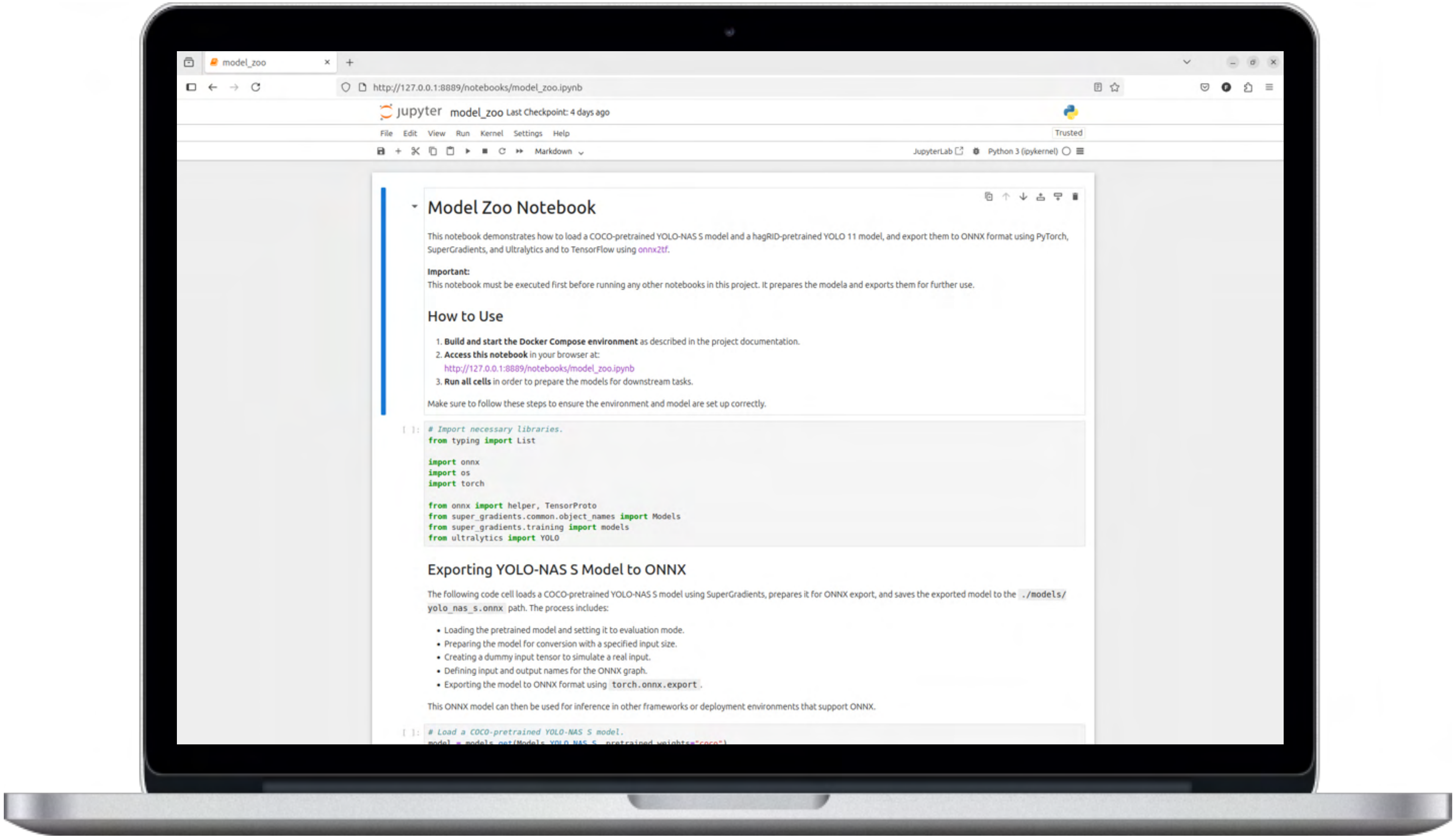
# MODEL OPTIMIZATION STEPS

## Official Documentation



# MODEL ZOO NOTEBOOK

http://127.0.0.1:8889/notebooks/model_zoo.ipynb




# YOLO-NAS MODELS

## Object Detection

YOLO-NAS S is a compact yet powerful object detection model developed by Deci AI, designed to deliver high accuracy with low latency.

### Results on NVIDIA's Tesla T4 GPU

| MODEL* | PRECISION* | $mAP^{val*}$ 0.5:0.95 | LATENCY* BS=1 (ms) | PARAMS (M) |
|---|---|---|---|---|
| YOLO-NAS S | FP16 | 47.5 | 3.21 | 19.0 |
| | INT-8 | 47.03 (-0.47) | 2.36 (+0.85) | |
| YOLO-NAS M | FP16 | 51.55 | 5.85 | 51.1 |
| | INT-8 | 51.0 (-0.55) | 3.78 (+2.07) | |
| YOLO-NAS L | FP16 | 52.22 | 7.87 | 66.9 |
| | INT-8 | 52.1 (-0.12) | 4.78 (+3.09) | |

*Image Size = 640

# COCO DATASET

Common Object in Context

## 80 Object Categories

Includes a broad range of everyday objects like people, vehicles, animals, and household items to support general-purpose detection.

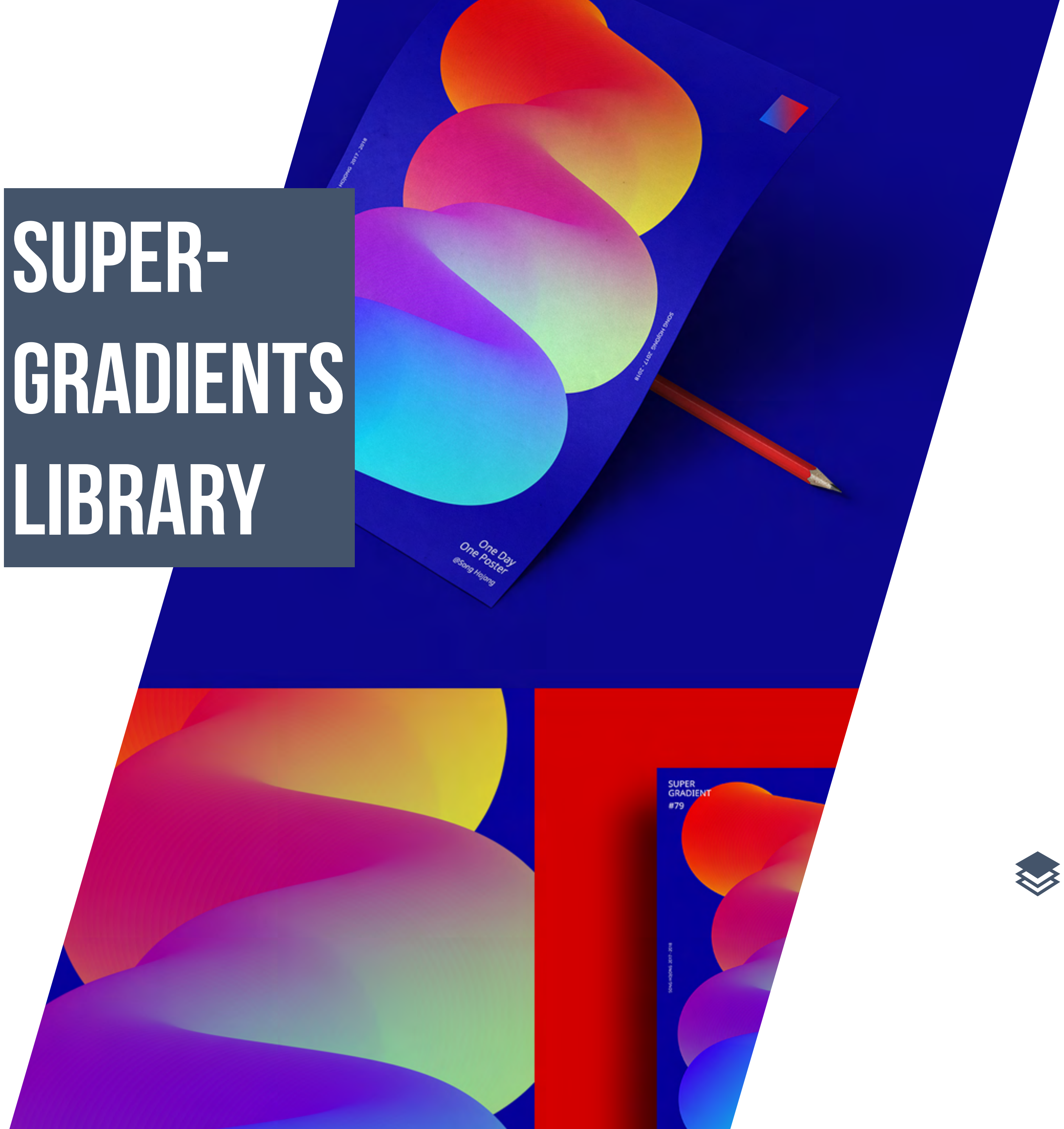
## Rich Annotations

Each image is annotated with bounding boxes, segmentation masks, keypoints, and contextual metadata.

## Real-World Scenarios

Features complex, cluttered scenes that closely resemble real-life environments—ideal for training edge-focused models.

# SUPER-GRADIENTS LIBRARY




## YOLO-NAS Training & Export

Provides pre-configured pipelines to train and export YOLO-NAS models for real-world detection tasks.

## ONNX Export Support

Enables seamless conversion of trained models into ONNX format, ready for further optimization and deployment.

## Efficient Training Utilities

Includes automatic mixed precision, advanced schedulers, and customizable training loops.

## Dataset Integration

Offers built-in support for standard datasets, such as COCO and Pascal VOC, thereby speeding up model development.

# SUPER-GRADIENTS URL BUG

patch -p1 <ENV_PACKAGES>/super_gradients/training/utils/checkpoint_utils.py < fix_url.patch

```
--- checkpoint_utils_old.py 2025-05-25 15:18:13.784853944 +0000
+++ checkpoint_utils.py 2025-05-25 15:33:07.357595243 +0000
@@ -1589,7 +1589,8 @@
     if url.startswith("file://") or os.path.exists(url):
         pretrained_state_dict = torch.load(
             url.replace("file://", ""), map_location="cpu")
     else:
-        unique_filename = url.split(
-            "https://sghub.deci.ai/models/"
-        )[1].replace("/", "_").replace(" ", "_")
+        url = url.replace(
+            "https://sghub.deci.ai",
+            "https://sg-hub-nv.s3.amazonaws.com"
+        )
+        unique_filename = url.split(
+            "https://sg-hub-nv.s3.amazonaws.com/models/"
+        )[1].replace("/", "_").replace(" ", "_")
         map_location = torch.device("cpu")
         with wait_for_the_master(get_local_rank()):
             pretrained_state_dict = load_state_dict_from_url(
                 url=url, map_location=map_location,
                 file_name=unique_filename
             )
```

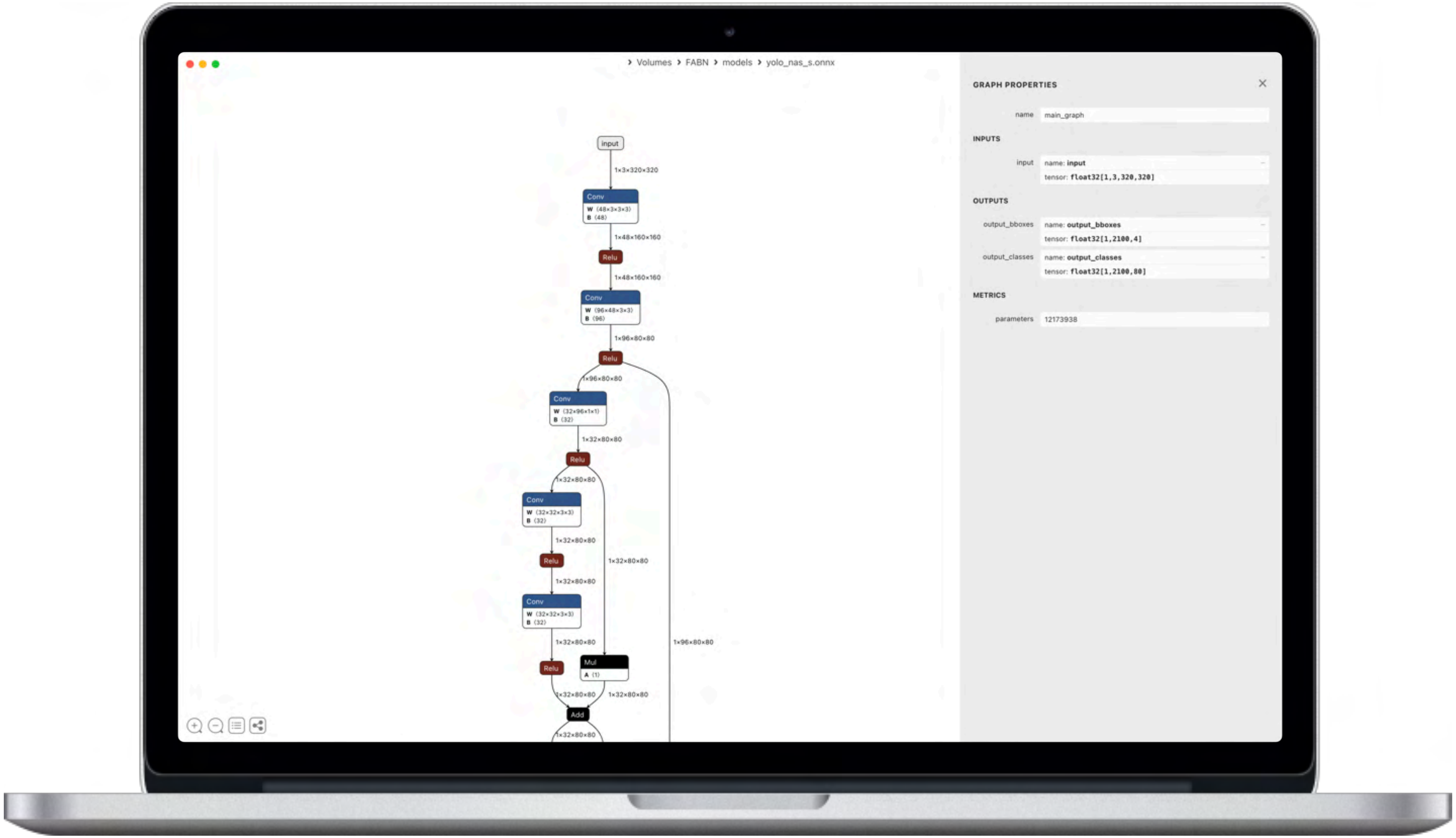
# EXPORTING YOLO-NAS S TO ONNX

http://127.0.0.1:8889/notebooks/model_zoo.ipynb

```
# Load a COCO-pretrained YOLO-NAS S model.
model = models.get(Models.YOLO_NAS_S, pretrained_weights="coco")
model.eval()

# Prepare the model for ONNX conversion.
model.prep_model_for_conversion(input_size=[1, 3, 320, 320])

# Define a dummy input tensor with the expected shape.
dummy_input = torch.randn([1, 3, 320, 320], device="cpu")

# Specify the input and output names for the ONNX model.
input_names = ["input"]
output_names = ["output_bboxes", "output_classes"]

# Export the model to ONNX format.
torch.onnx.export(
    model,
    dummy_input,
    "/models/yolo_nas_s.onnx",
    input_names=input_names,
    output_names=output_names,
    opset_version=11
)
```

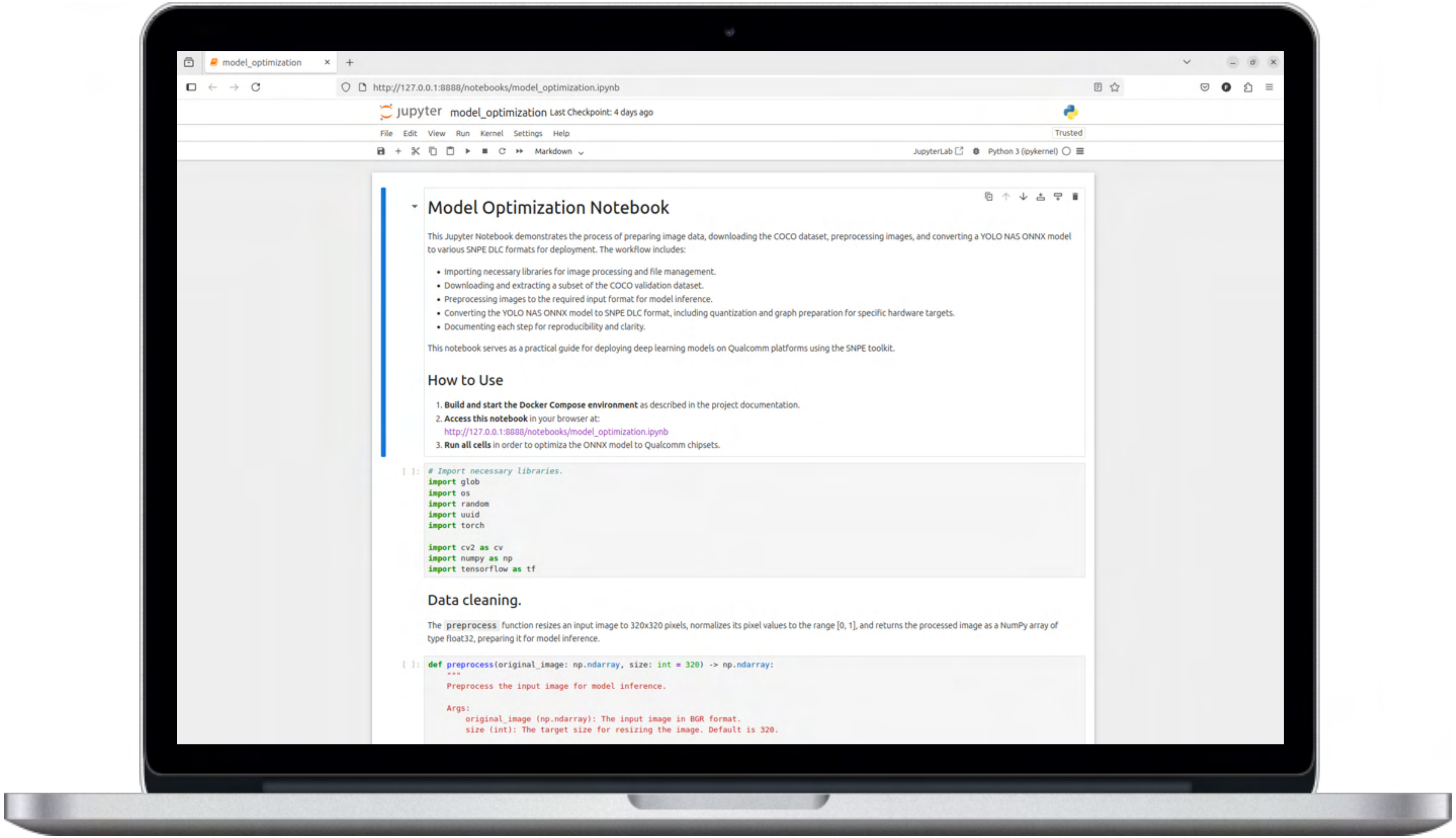
# VISUALIZING ONNX MODEL

http://netron.app



# MODEL OPTIMIZATION NOTEBOOK

http://127.0.0.1:8888/notebooks/model_optimization.ipynb



# GETTING COCO DATASET

## Validation Dataset

### Validation Dataset

To evaluate object detection models like YOLO-NAS S, downloading and preparing the COCO validation dataset is essential.

### Data Annotation

The validation set provides a standardized benchmark to assess model accuracy, bounding box quality, and object recall.

### Optimization Phase

The validation split (commonly `val2017`) enables faster iteration and tuning during the optimization phase.

# MODEL CONVERSION AND OPTIMIZATION

## SNPE Optimization

## CONVERSION

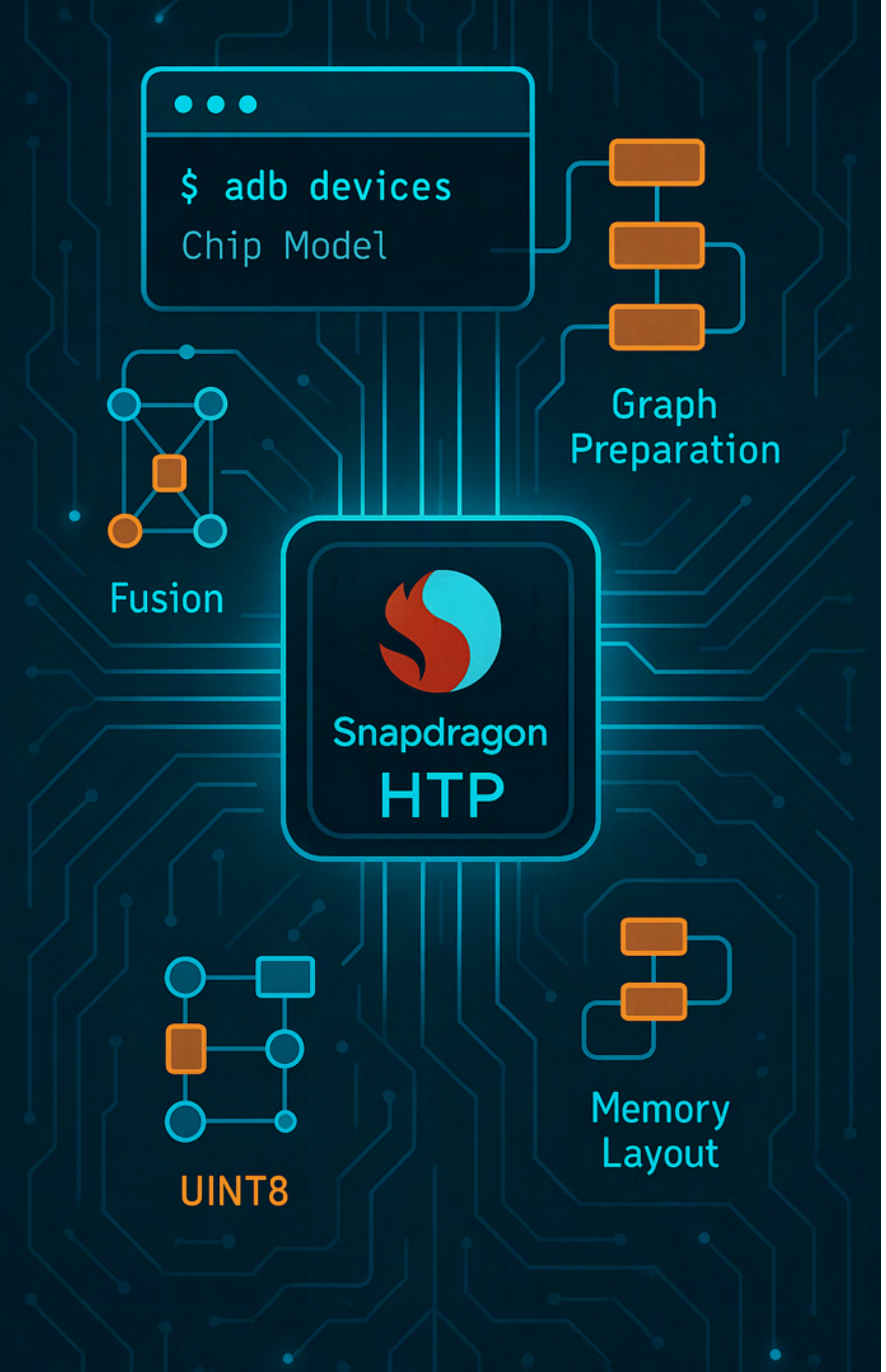
Convert the ONNX model to DLC (Float32)

```
$ snpe-onnx-to-dlc -i \
   yolo_nas_s.onnx -o \
   yolo_nas_s_fp32.dlc
```


## QUANTIZATION

Compresses the model by reducing parameter precision

```
$ snpe-dlc-quantize \
    --input_dlc \
    yolo_nas_s_fp32.dlc \
    --input_list input.txt \
    --output_dlc \
    yolo_nas_s_int8.dlc
```




# HARDWARE GRAPH PREPARATION

## Best model performance

To achieve optimal inference performance on Qualcomm devices, models must be tailored to the target hardware's capabilities—especially the Hexagon Tensor Processor (HTP).

### Get Qualcomm Chip Name (using ADB)

Use the command `adb shell getprop ro.soc.manufacturer` or `ro.soc.model` to retrieve the chip name and confirm device compatibility.

### Prepare for HTP Execution

Converts the model to a graph optimized for the Hexagon Tensor Processor, enabling low-latency, power-efficient inference.

# HARDWARE-SPECIFIC GRAPH PREPARATION

## Get Qualcomm Chip Name

# HARDWARE-SPECIFIC GRAPH PREPARATION

## Prepare for HTP Execution

Hardware-specific graph preparation ensures that the model runs efficiently by leveraging the device's supported layer fusions, memory layouts, and precision types.

### MODEL OPTIMIZATION

Optimize DLC model (uint8):

```
$ snpe-dlc-graph-prepare \
    --input_dlc \
    yolo_nas_s_int8.dlc \
    --set_output_tensors=\
    output_bboxes,output_classes \
    --htp_socs=sm7325 \
    --output_dlc=\
    yolo_nas_s_int8_htp_sm7325.dlc
```

### MODEL INSPECTION

Inspect DLC models:

```
$ snpe-dlc-info -i \
    yolo_nas_s_int8_htp_sm7325.dlc
```

# MODEL TABLE COMPARISON

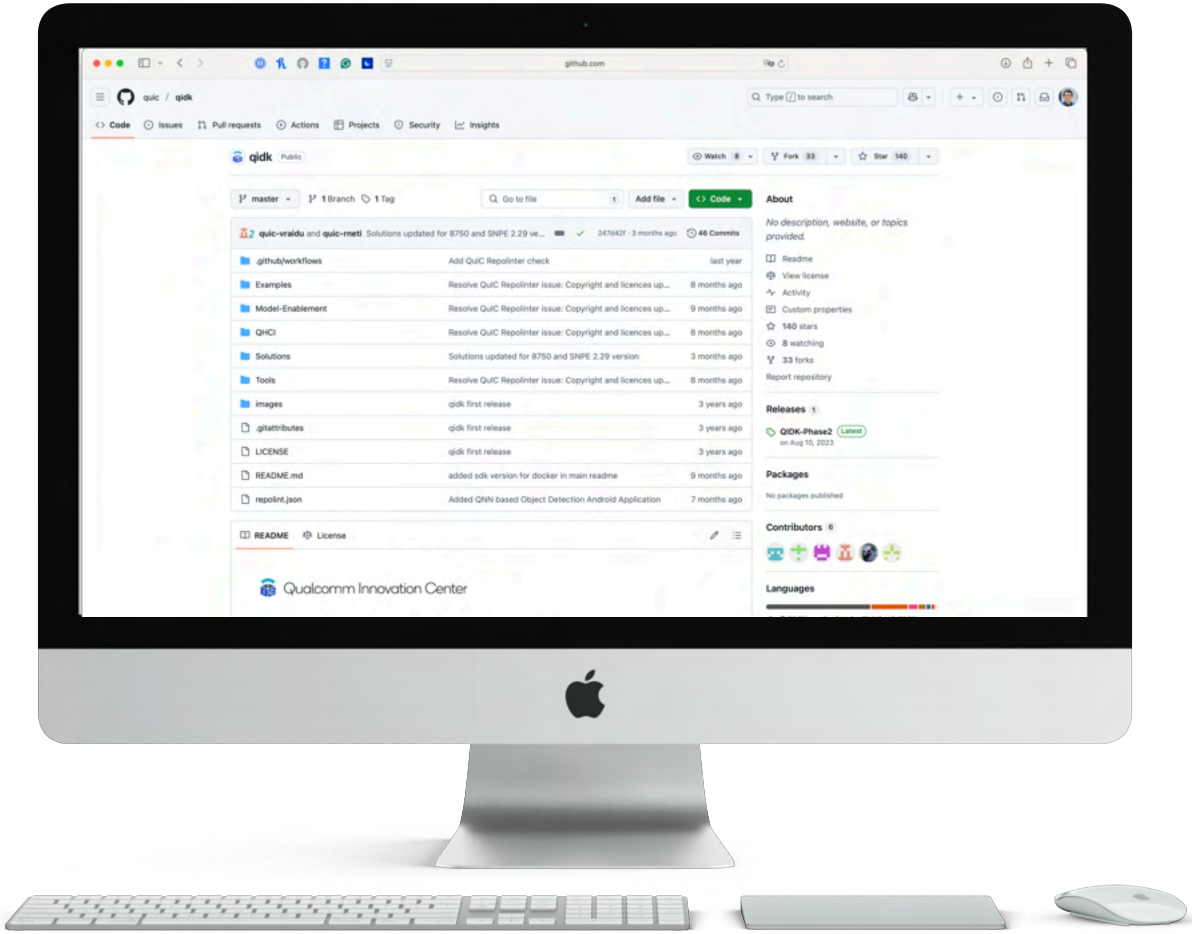
## Results

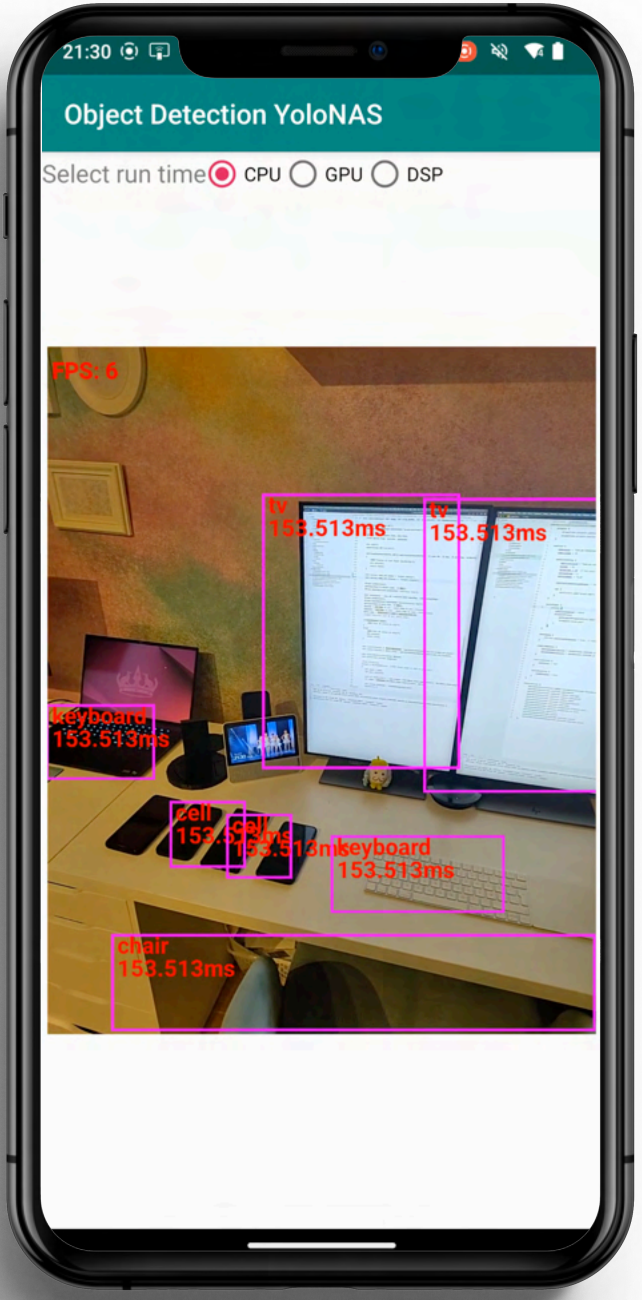
| Model Variant | File Size (MB) | Precision | MAC Operations | Expected Speedup | Accuracy Drop (if any) |
|---|---|---|---|---|---|
| yolo_nas_s.onnx | 46.53 | FP32 | General (CPU/GPU) | Baseline | 0% (baseline) |
| yolo_nas_s_fp32.dlc | 46.74 | FP32 | Qualcomm CPU/GPU | Slight | ~0% |
| yolo_nas_s_int8.dlc | 11.92 | INT8 | Qualcomm CPU/GPU/DSP | 2-4x | Typically <1% |
| yolo_nas_s_int8_htp_sm7325.dlc | 23.98 | INT8 | HTP (SM7325/6490 SoC) | 5-10x | Typically <1% |

GN

# QUALCOMM INNOVATION CENTER

https://github.com/quic/qidk

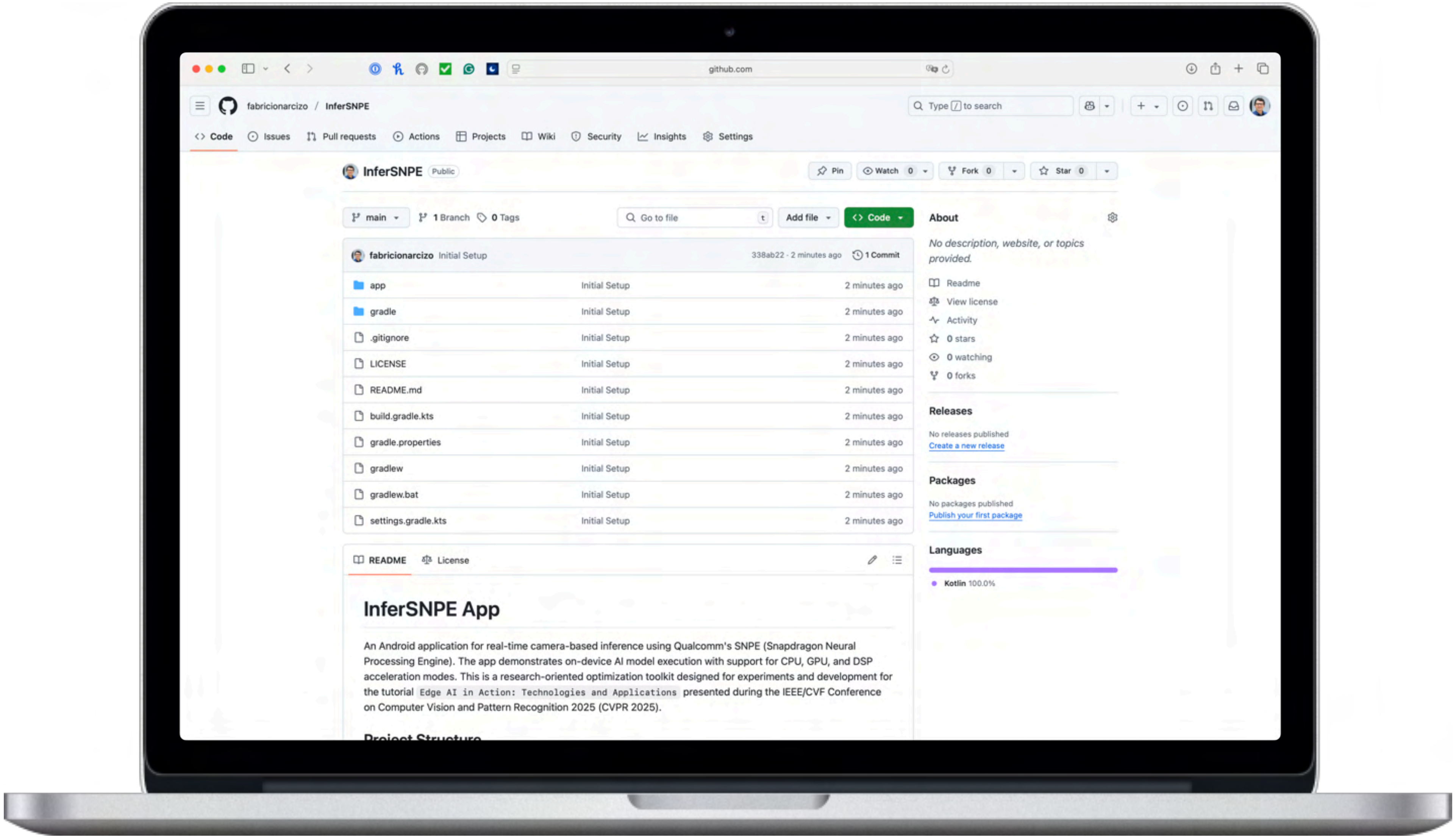


# MODEL DEPLOYMENT ON ANDROID

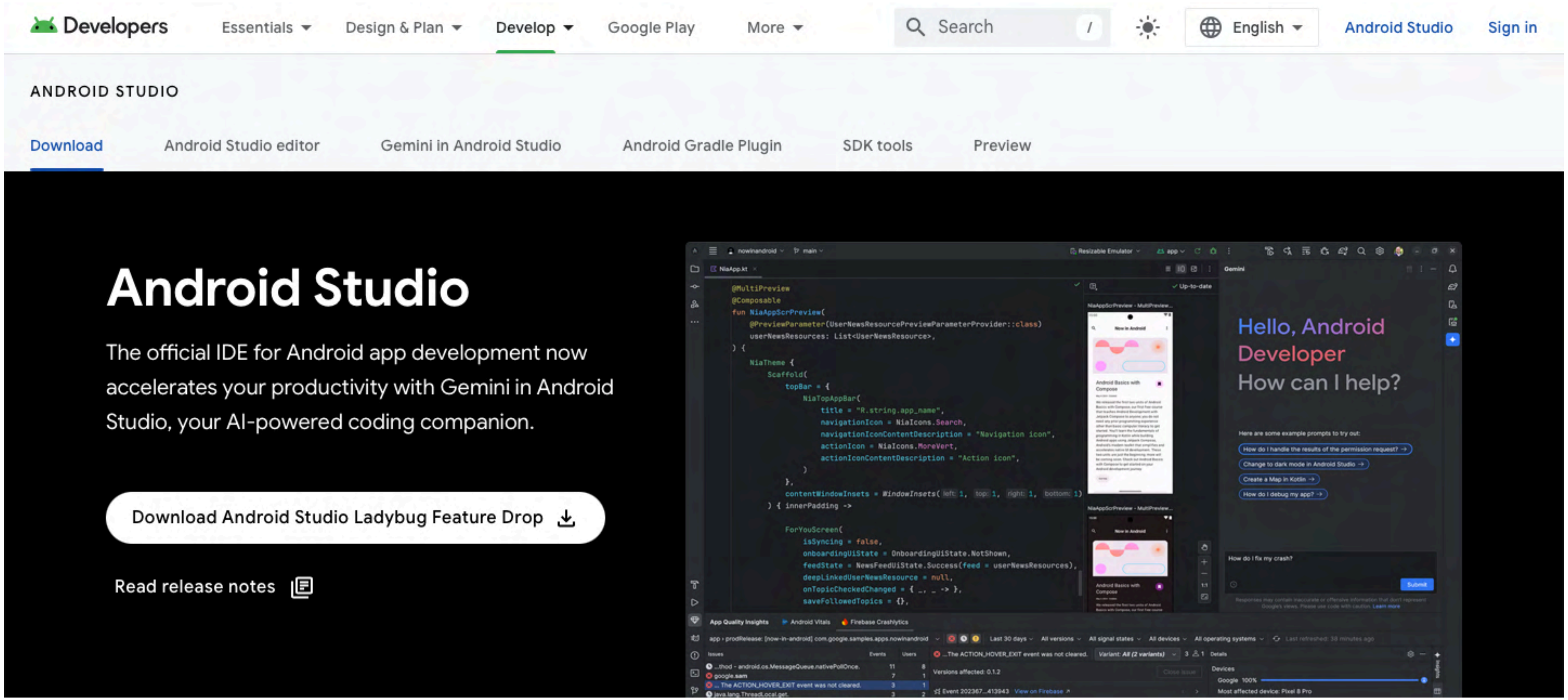
# INFERSNPE APP

https://github.com/fabricionarcizo/InferSNPE



# DEVELOPING ANDROID APPS

## Android Studio



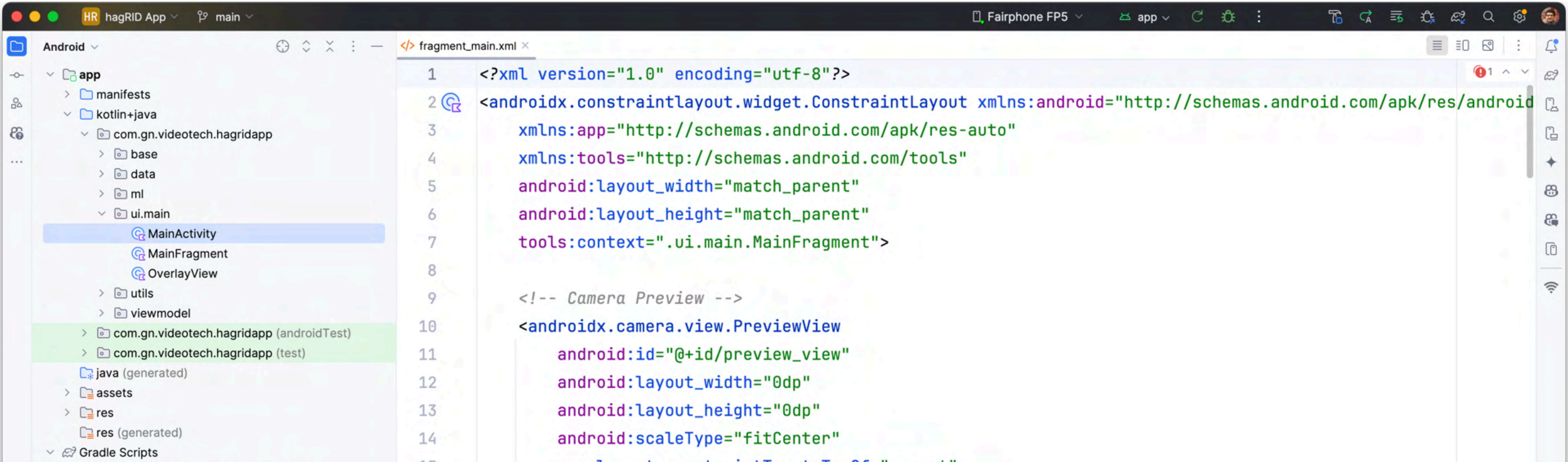

HTTPS://DEVELOPER.ANDROID.COM/STUDIO

# ANDROID STUDIO

## Integrated Development Environment

---

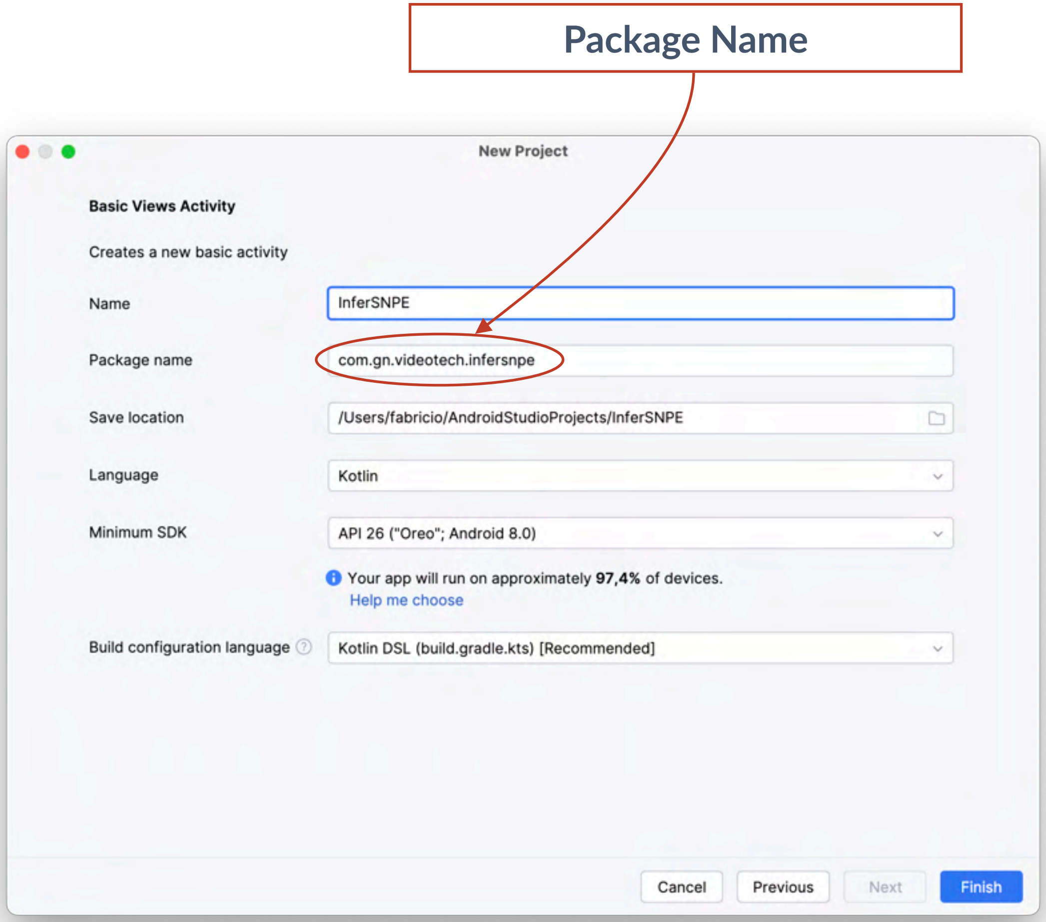
Android Studio is the official integrated development environment for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS, and Linux-based operating systems.



# ANDROID STUDIO SETTINGS

## Important Information

On June 05, 2025, the released version of Android Studio was Ladybug Feature Drop v2024.2.2. From time to time, IntelliJ Platform updates Android Studio. For this tutorial, we implemented an app called the InferSNPE App using *Android SDK Build-Tools* v36.0.0 and *Android SDK Platform-Tools* v35.0.2.




### Configure your project

- Start a new project using the *Basic Views Activity*
- Package name based on reverse domain name notation
- Save the location without spaces in the folder name
- Use the minimum API level for Android 8.0 (API 26 Oreo)
- Select Kotlin DSL for the build configuration language

# ANDROID APP MANIFEST

## Overview

The AndroidManifest.xml is the primary configuration file of your app project. It describes essential information about your app to the Android build tools, the Android operating system, and Google Play.

### Package Name

Determine the location of code entities when building your project.

### Components

Information about activities, services, broadcast receivers, and content providers.

### Permissions

Set permissions to access content from the app.

### Requirements

Requirements of hardware and software.

HTTPS://DEVELOPER.ANDROID.COM/GUIDE/TOPICS/MANIFEST/MANIFEST-INTRO

# ANDROID APP MANIFEST

## AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.camera.any" />

    <application
        android:name=".base.InferSNPE"
        android:allowBackup="true"
        android:extractNativeLibs="true"
        android:hardwareAccelerated="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="“@style/Theme.InferSNPE"
        tools:targetApi="31">
        ...
    </application>
</manifest>
```

# GRADLE BUILD TOOL

## Android Project Backbone

Gradle is an advanced build toolkit for Android development. It automates and manages the build process, ensuring efficient and dependency handling.

### POWERFUL BUILD AUTOMATION

Gradle automates compiling, testing, and packaging, simplifying project workflows.

### DEPENDENCY MANAGEMENT

Handles external libraries and frameworks with Maven or JCenter repositories.

### CUSTOMIZABLE BUILD LOGIC

Allows fine-tuning of build processes with Groovy/Kotlin DSL.

# APP GRADLE FILE

## build.gradle.kts

```
android {
    namespace = "com.gn.videotech.infersnpe"
    compileSdk = 35

    defaultConfig {
        applicationId = “com.gn.videotech.infersnpe"
        minSdk = 26
        targetSdk = 30  // Up to API 30 enables GPU and DSP modes.
        versionCode = 1
        versionName = "1.0"
        ...
        ndk {
            abiFilters.add("arm64-v8a")  // Compile APK only for ARM64.
        }
    }

    packaging {
        jniLibs.useLegacyPackaging = true  // Enable DSP support.
    }

    ...

}
```

IMPORTANT

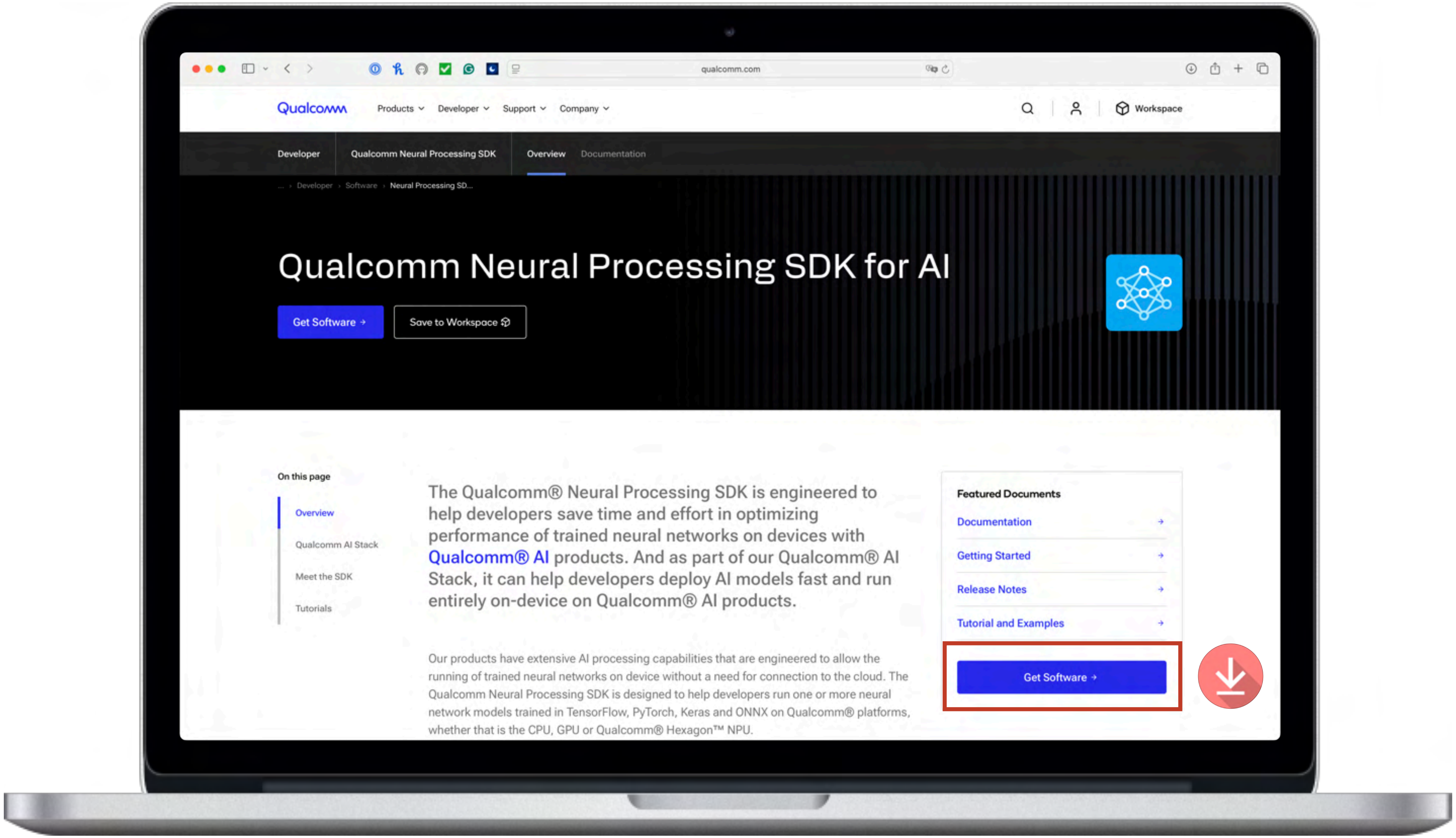
# APP GRADLE FILE

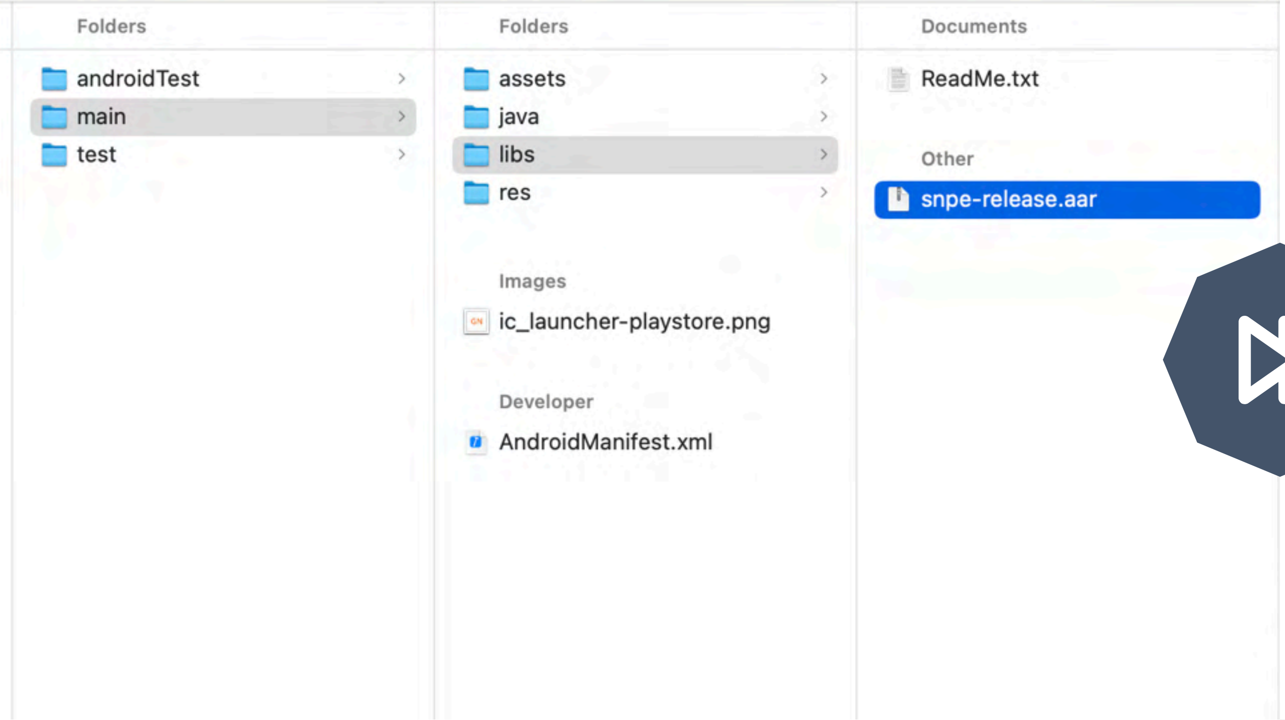
## build.gradle.kts

```
android {
    ...
    buildFeatures {
        viewBinding = true
    }
}

dependencies {
    implementation(files("src/main/libs/snpe-release.aar"))
    implementation(libs.androidx.appcompat)
    implementation(libs.androidx.camera.camera2)
    implementation(libs.androidx.camera.lifecycle)
    implementation(libs.androidx.camera.view)
    implementation(libs.androidx.constraintlayout)
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.navigation.fragment.ktx)
    implementation(libs.androidx.navigation.ui.ktx)
    implementation(libs.material)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
}
```

# QUALCOMM NEURAL PROCESSING SDK FOR AI

https://www.qualcomm.com/developer/software/neural-processing-sdk-for-ai



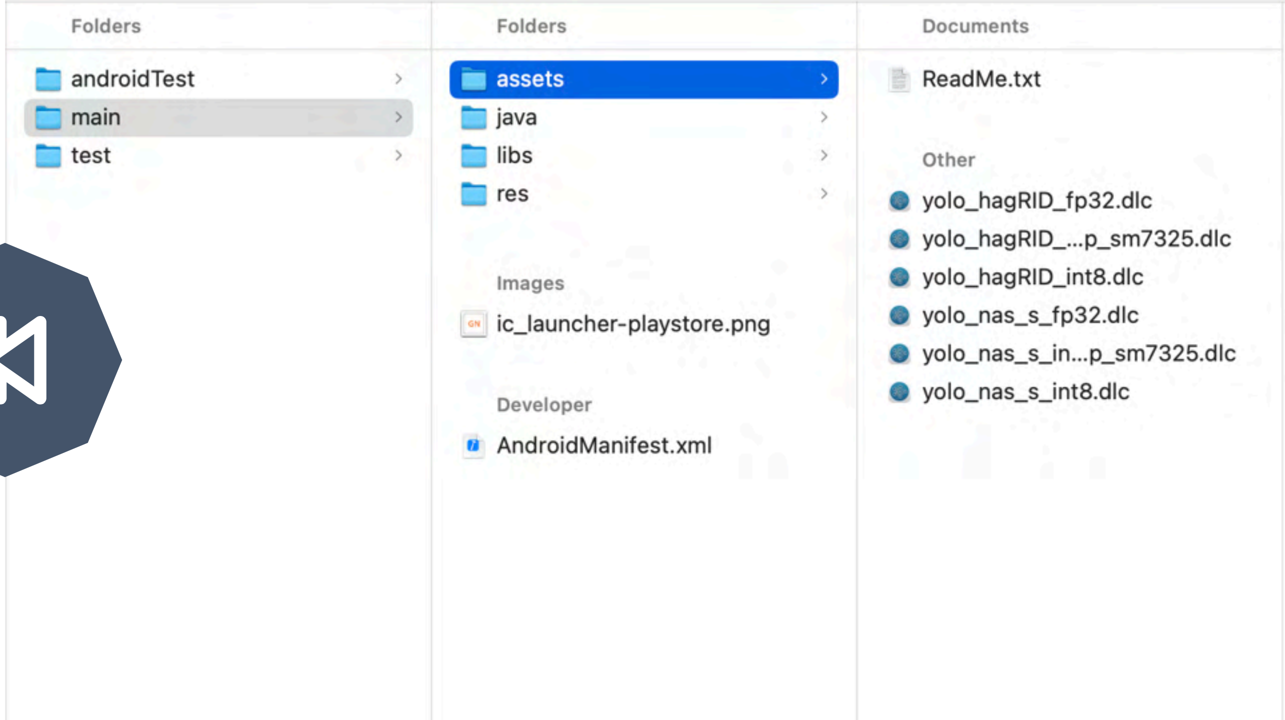



## SNPE-RELEASE.AAR (LIBS)

Android library package that includes all necessary SNPE binaries, Java interfaces, and native shared libraries for runtime execution.

## DLC FILES (ASSETS)

Hardware-optimized model files generated by SNPE tools from ONNX or TensorFlow sources; required for deployment on Qualcomm chipsets.



# INFERSNPE APP
## User Interface

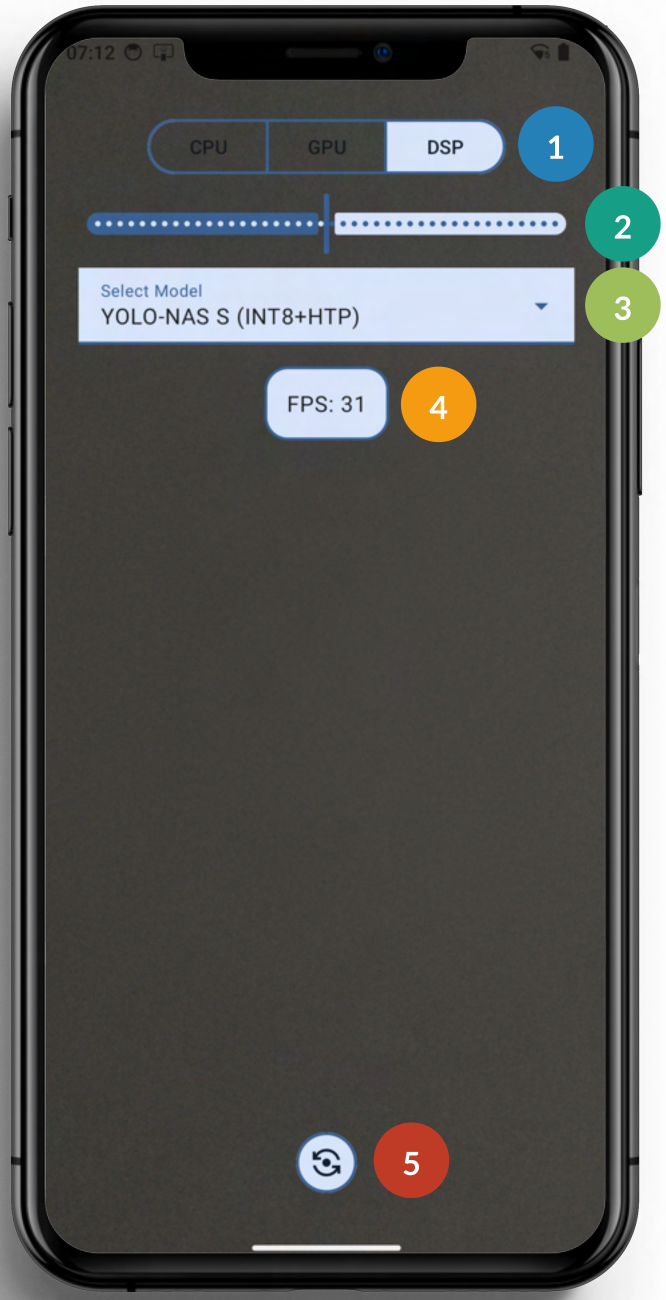
### 1 AI PROCESSING UNITS
Lets users select the target backend (CPU, GPU, DSP, or NNAPI) to observe performance trade-offs across hardware accelerators.

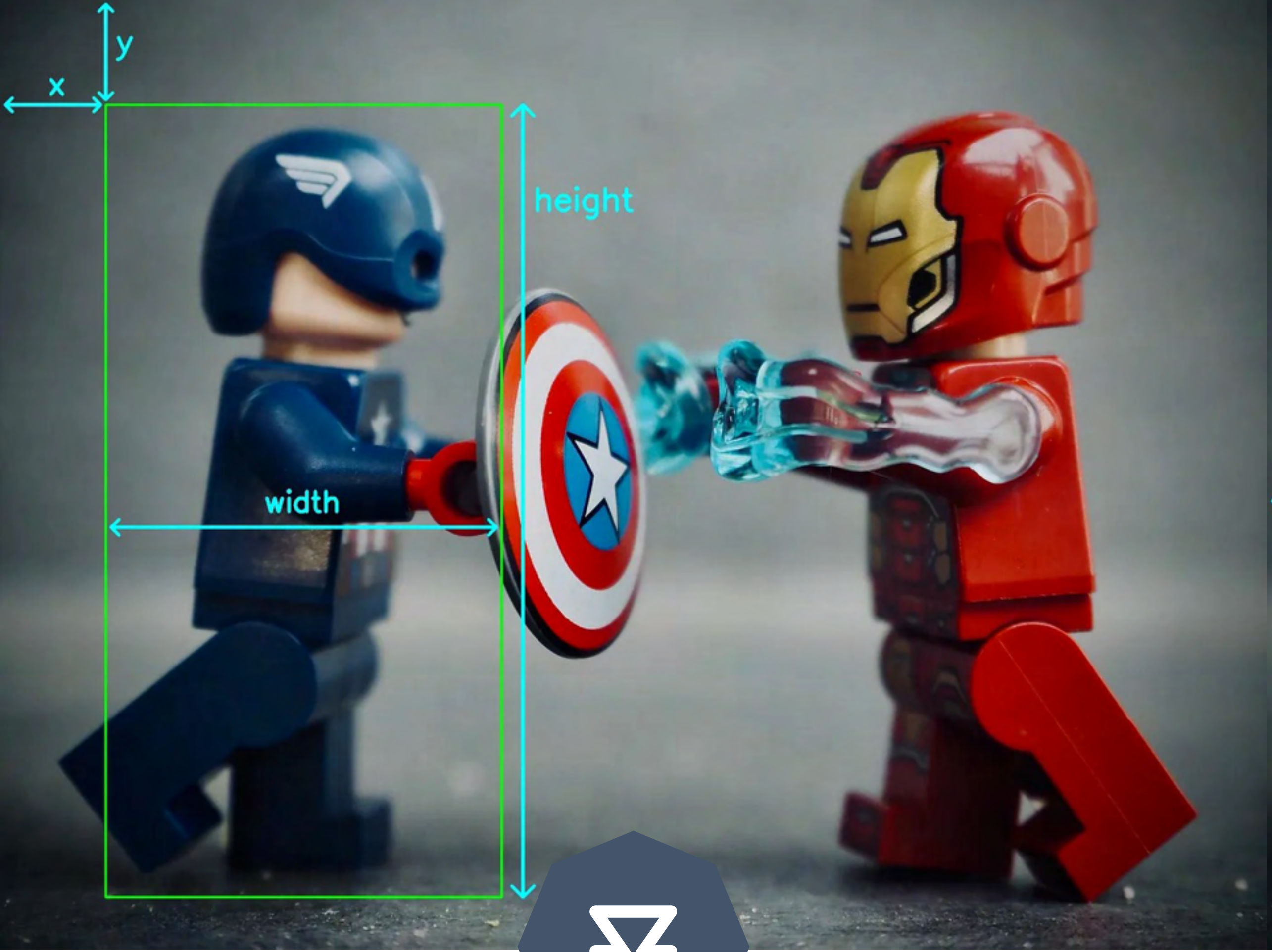
### 2 CONFIDENCE LEVEL
Sets the prediction confidence threshold for each detected object, aiding quick model validation during testing.

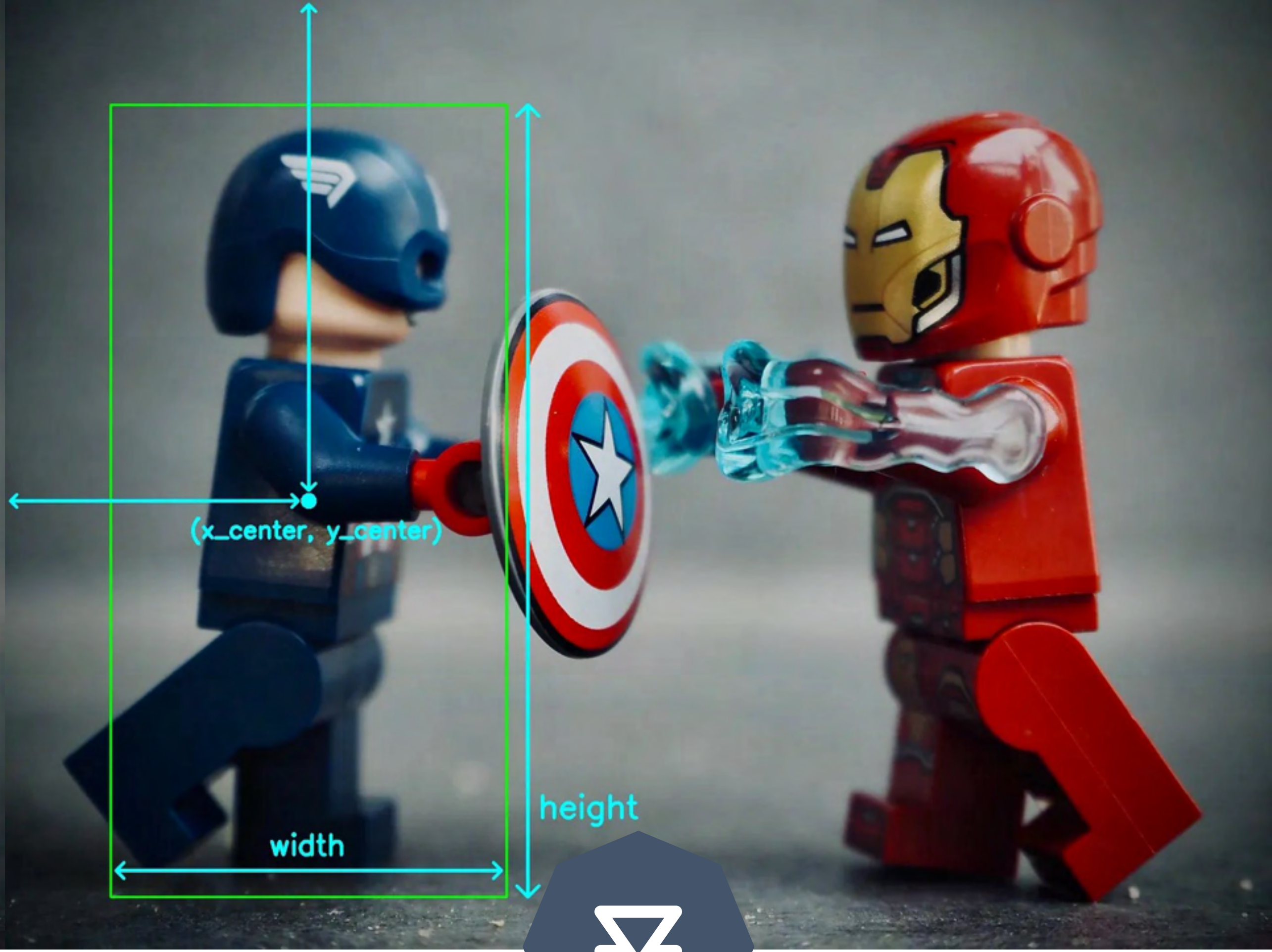
### 3 MODEL SELECTOR
Allows switching between multiple DLC models, supporting benchmarking and comparison of different architectures.

### 4 FRAMERATE
Shows the real-time frame processing speed (FPS), critical for evaluating inference latency and throughput.

### 5 CAMERA SWITCHER
Enables toggling between front and back cameras to test gesture or object detection under different use cases.

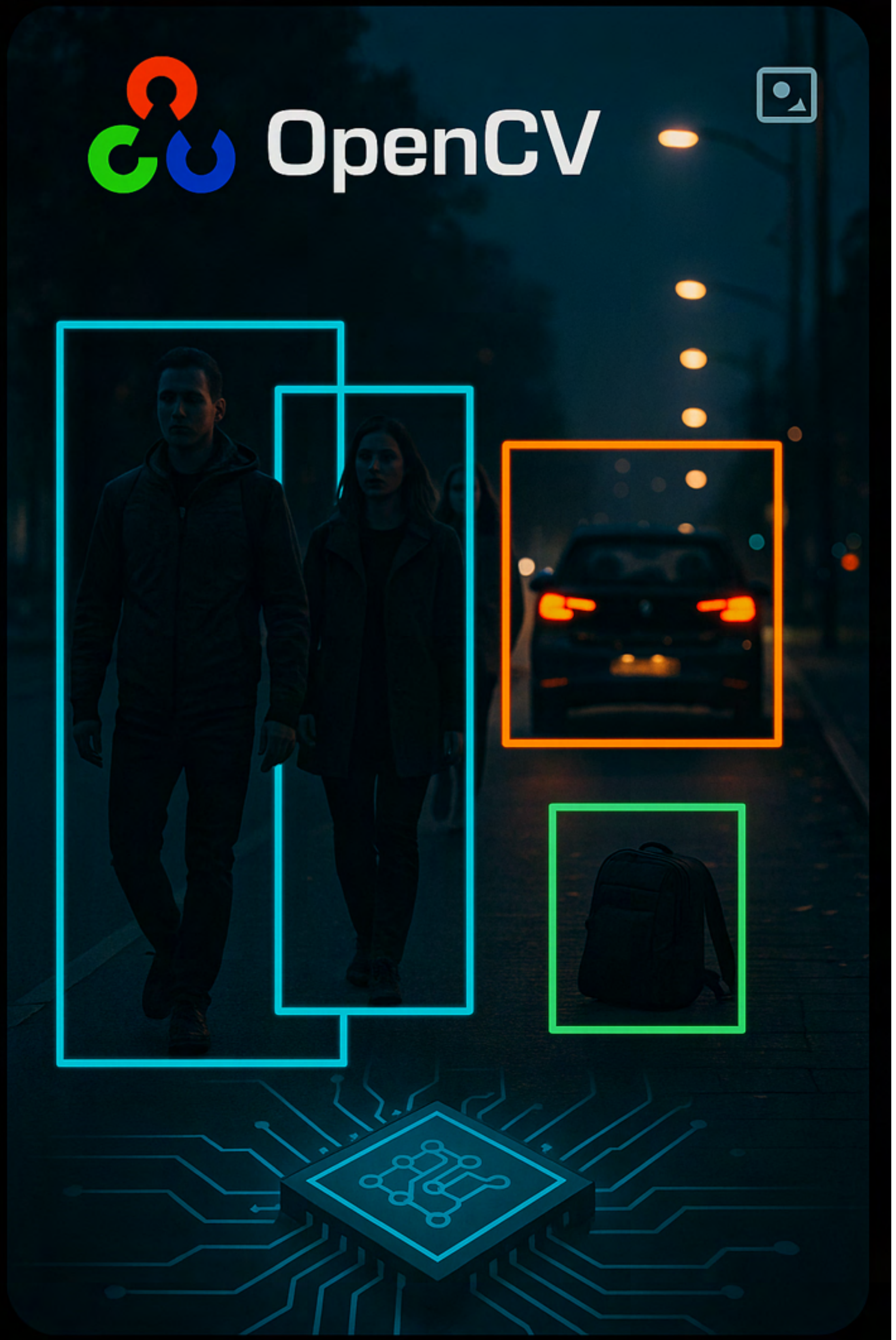





## CORNER-BASED BOUNDING BOX

## CENTER-BASED BOUNDING BOX




# DRAWING RECTANGLES WITH OPENCV

## Overview

In real-time object detection applications, drawing bounding boxes is essential for visualizing model predictions. OpenCV offers simple utilities to render these rectangles directly on the camera feed.

### cv2.rectangle()

A simple OpenCV method to draw rectangles using coordinates and color values on image frames.

### Performance

Drawing directly on CPU frames can slow down the pipeline, especially when combined with high-resolution frames.

### OverlayView

Android-native drawing, utilizing Canvas and SurfaceView, provides improved performance and responsiveness.

# CUSTOM OVERLAY VIEW

## Drawing Bounding Boxes

### Front Camera Support

Flips boxes horizontally if using a front-facing camera.

### Dynamic Scaling

Maps bounding boxes from image to view coordinates while maintaining aspect ratio.

### Styled Drawing

Uses customizable Paint objects for boxes, text, and background.

### Custom View

Draws detection results, such as bounding boxes and labels, over the camera preview or images.

### Live Updates

The method updateDetections() triggers redraws with new detection results.

# SNPEHELPER CLASS

## Overview

SNPEHelper is a custom utility class designed to simplify interaction with the SNPE runtime on Android. It also ensures consistency in how DLC models are handled across different app components.

### Model Initialization

Loads the DLC file, configures runtime (CPU, GPU, or DSP), and sets up input/output layers.

### Input Preprocessing

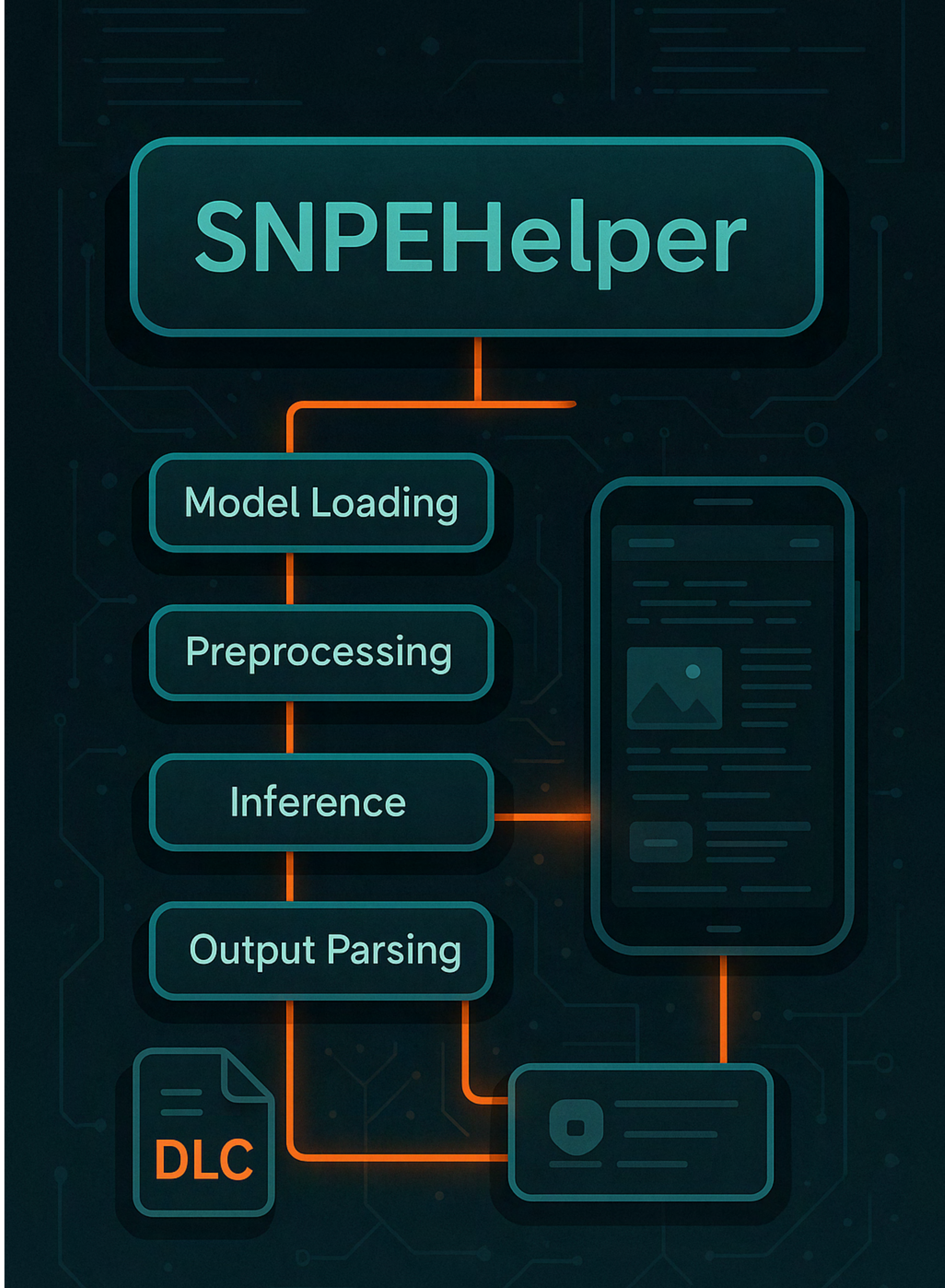
Handles resizing, normalization, and data formatting of images before feeding them to the model.

### Inference Execution

Executes the model and retrieves raw output tensors in a hardware-optimized and asynchronous manner.

### Output Parsing

Interprets model outputs into usable objects like bounding boxes, labels, and confidence scores.



# MODEL INITIALIZATION

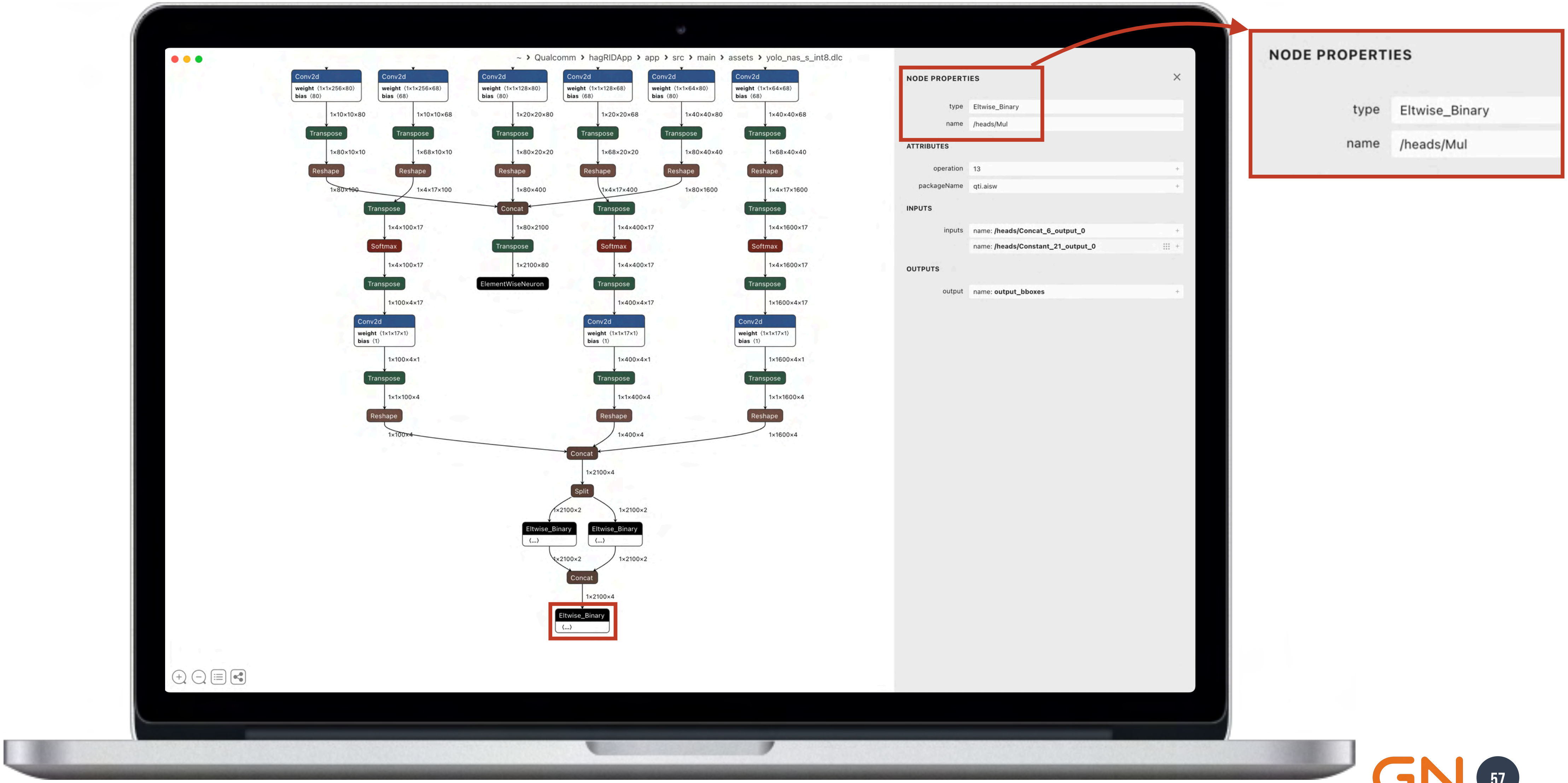
## SNPEHelper

```kotlin
private fun loadModelFromAssets(runtime: NeuralNetwork.Runtime) = try {
    val filePath = selectedModel.filePath
    application.assets.open(filePath).use { stream ->
        val outputLayers = arrayOf("/heads/Mul", "/heads/Sigmoid")
        val model = SNPE.NeuralNetworkBuilder(application)
            .setRuntimeCheckOption(
                if (isUnsignedPD)
                    NeuralNetwork.RuntimeCheckOption.UNSIGNEDPD_CHECK
                else NeuralNetwork.RuntimeCheckOption.NORMAL_CHECK
            )
            .setOutputLayers(*outputLayers)
            .setModel(stream, stream.available())
            .setPerformanceProfile(
                NeuralNetwork.PerformanceProfile.DEFAULT)
            .setRuntimeOrder(runtime)
            .setCpuFallbackEnabled(false)
            .build()
        model
    }
} catch (e: Exception) {
    Log.e("SNPEHelper", "Model loading error", e)
    null
}
```

IMPORTANT

# MODEL OUTPUT NAME

## Netron



# INPUT PREPROCESSING AND INFERENCE EXECUTION

## SNPEHelper

```kotlin
private fun runModel(bitmap: Bitmap): Map<String, FloatTensor>? {
    val resized = bitmap.resized(getInputWidth()) // 320x320
    if (
        resized.width != getInputWidth() ||
        resized.height != getInputHeight() ||
        inputTensor == null || inputMap == null || neuralNetwork == null
    ) return null

    return runCatching {
        bitmapUtility.convertBitmapToBuffer(resized)
        val floats = bitmapUtility.bufferToFloatsRGB() // [0, 1]

        // Skip black frames.
        if (bitmapUtility.isBufferBlack()) return null

        inputTensor?.write(floats, 0, floats.size, 0, 0)
        neuralNetwork?.execute(inputMap)
    }.onFailure {
        Log.e("SNPEHelper", "Inference error", it)
    }.getOrNull()
}
```

# OUTPUT PARSING

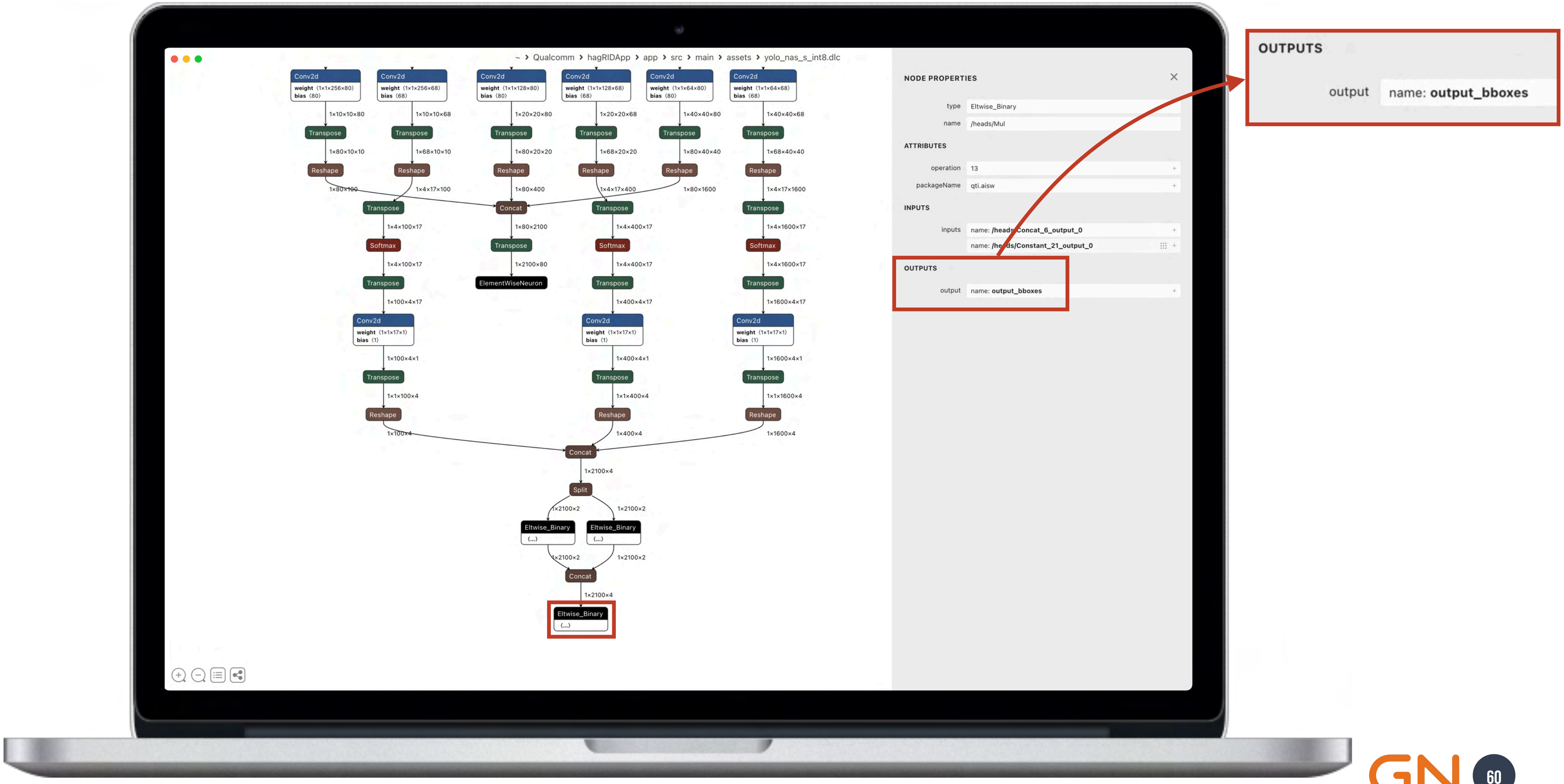
## SNPEHelper

```
fun inference(bitmap: Bitmap, threshold: Float = 0.5f):
              List<DetectionResult> {
    val output = runModel(bitmap) ?: return emptyList()
    val outputNames = arrayOf("output_bboxes", "output_classes")

    val boxes = output[outputNames[0]] ?: return emptyList()
    val classes = output[outputNames[1]] ?: return emptyList()

    val numDetections = boxes.shape[1]
    val numCorners = boxes.shape[2]
    val numClasses = classes.shape[2]

    val boxArray = FloatArray(numDetections * numCorners)
    val classArray = FloatArray(numDetections * numClasses)
    boxes.read(boxArray, 0, boxArray.size)
    classes.read(classArray, 0, classArray.size)

    val scaleX = bitmap.width.toFloat() / getInputWidth()
    val scaleY = bitmap.height.toFloat() / getInputHeight()
    val rectFormat = selectedModel.rectFormat
    val classNameMap = getClassNameMapping()
    ...
}
```

IMPORTANT

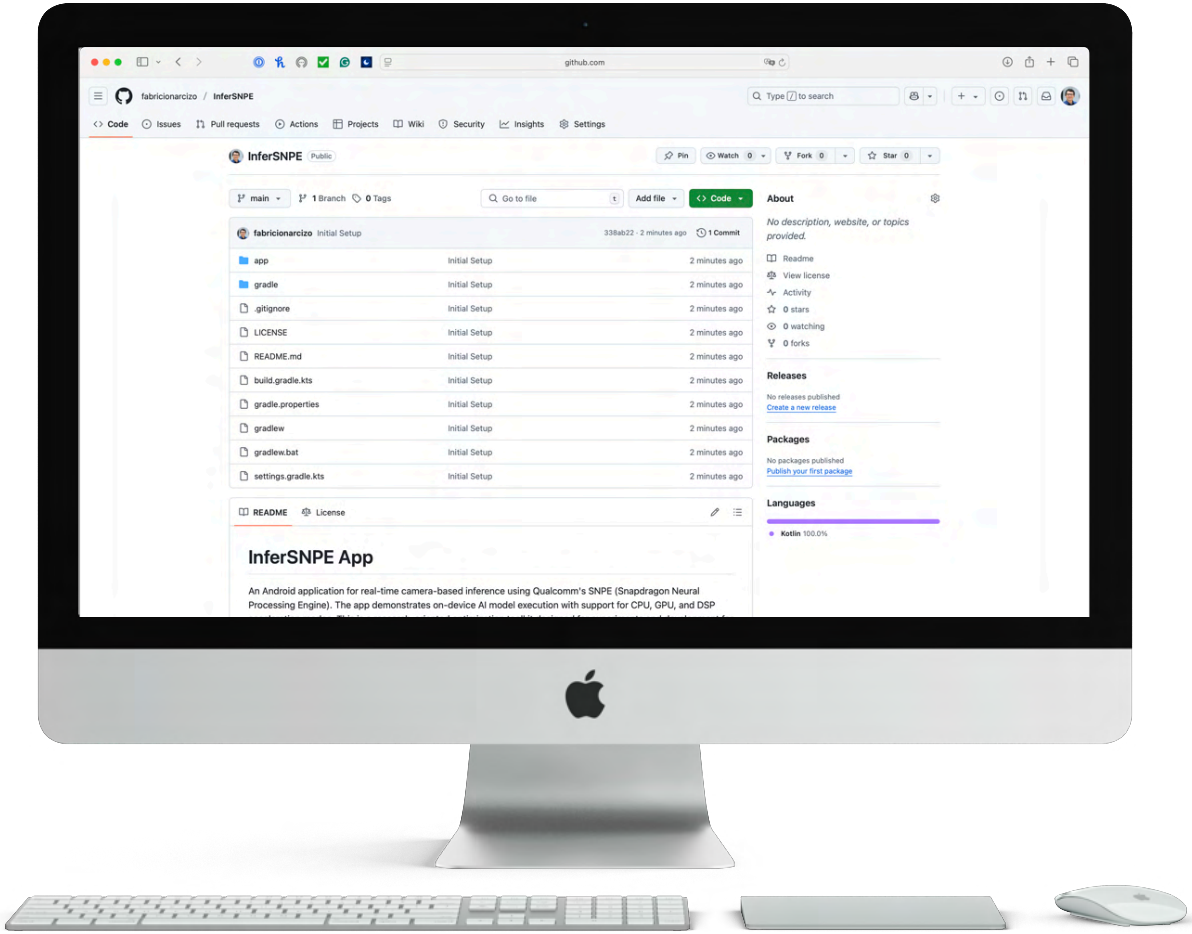
# MODEL OUTPUT NAME

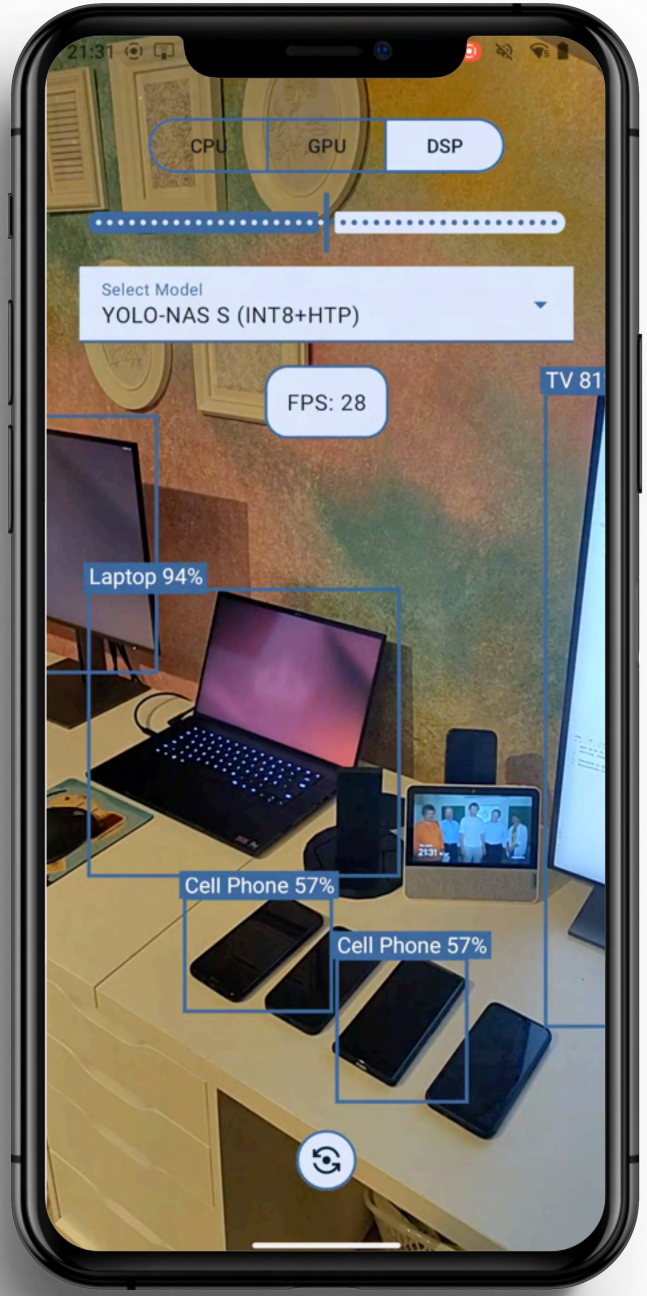
## Netron



# INFERSNPE APP

https://github.com/fabricionarcizo/InferSNPE



# HAND GESTURES RECOGNITION

# WHY HAND GESTURES

Hand gestures are everywhere

### AGRICULTURE AND INDUSTRY

Equipment Operation
On-Site Inspections

### HEALTHCARE

Surgical Assistance
Rehabilitation
Assistive Devices

### SMART HOME AND ENTERTAINMENT

Interactive Media Control
Home Automation and Control
Security System
Access Control

### ROBOTICS AND AUTOMATION

Service Robots
Collaborative Robots
Machine Control
Quality Inspection
Training Simulations

### HUMAN-COMPUTER INTERACTION

Assistive Technology
Interactive Learning
Virtual Classrooms
Special Education

### FITNESS AND SPORTS

Exercise Monitoring
Sports Training

### PUBLIC INTERACTION

Interactive Displays
Kiosks and Information Terminals
Exhibits and Museums
Transportations Hubs

### AUTOMOTIVE

In-Car Infotainment Systems
Driver Assistance

### VIRTUAL REALITY AND GAMING

Virtual Reality (VR)
Augmented Reality (AR)

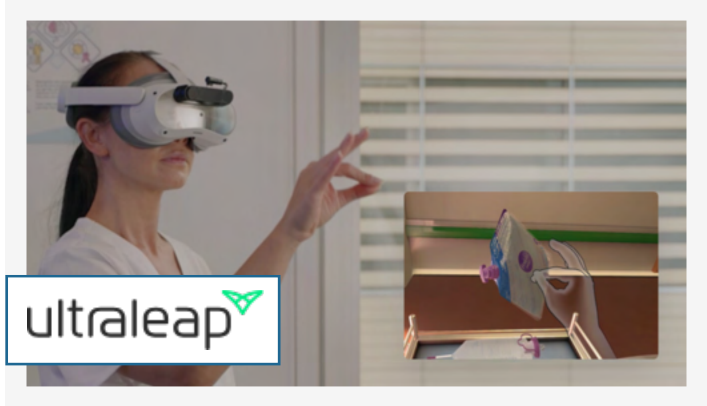
### CONSUMER

Smartphones and Tablets
Virtual Assistants
Selfie cameras

Several Fields of Applications

# HAND GESTURE PRODUCTS

## Example in different industries



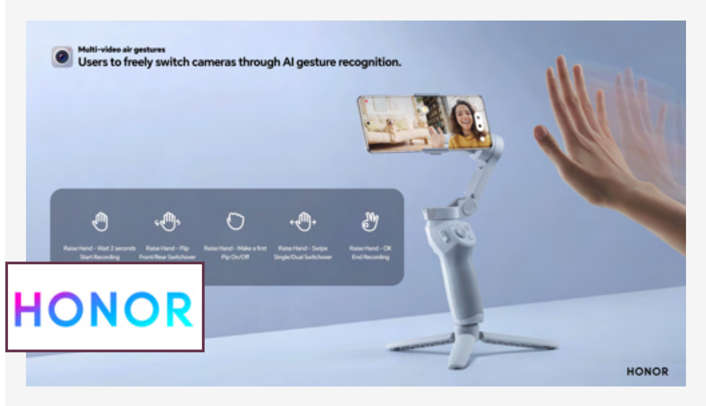

Leap Motion Controller 2


HoloLens 2


Echo Show


Gesture Control Armband


AIR Neo Selfie Pocket Drone




HONOR Cellphone Camera

# HAND GESTURES IN HYBRID MEETINGS

## Enhancing Multimodal Hybrid Meeting Control


> "Hand gestures offer a promising way to enhance shared digital meeting spaces by overcoming the limits of unimodal interaction and improving engaging and control."

# BENEFITS OF HAND GESTURES

## Overview

HG supports immersive experiences of entertainment and control by providing more natural and engaging ways to interact with digital environments, systems and devices.

### Enhances User Experience

Provides multimodal interaction methods, making systems more user-friendly and versatile.

### Promotes Accessibility

Offers alternative communication methods for individuals with disabilities, enhancing inclusivity and usability.

### Enables Touchless Control

Enables hygienic interaction by eliminating the need for physical contact, ideal for public and shared environments.

### Increases Efficiency

Allows for quick and efficient execution of commands through simple gestures, reducing reliance on traditional input devices.

# HAND-BASED TECHNOLOGY

## General view

Hand-based technology uses cameras or other sensors to capture the users’ hand gestures and movements.
Algorithms or Machine Learning models then analyze and interpret the hand poses or performances from the captured data.

HAND DETECTION

HAND TRACKING

GESTURE RECOGNITION

# HAND GESTURE RECOGNITION

Looking into the pipeline process

HAND DETECTION — HAND TRACKING — GESTURE RECOGNITION

### DATA AQCUISITION
Capturing data using one or more sensors

### PREPROCESSING DATA
Processing the raw data to focus on the relevant part

### FEATURE EXTRACTION
Identifying and isolating key characteristics

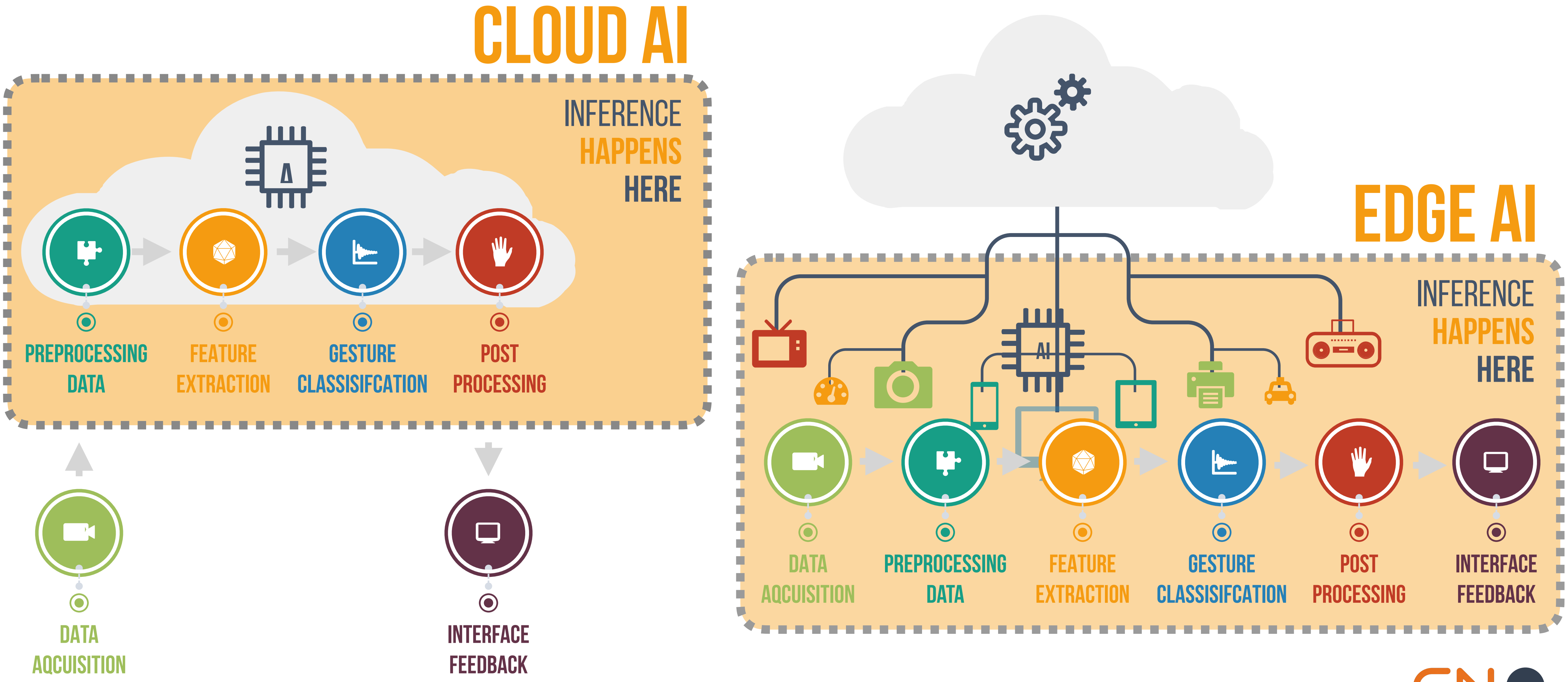
### GESTURE CLASSISIFCATION
Classifying the extracted features into predefined gesture categories

### POST PROCESSING
Refining the classification results and mapping the gestures to actions

### INTERFACE FEEDBACK
Providing system feedback to the user

# HAND GESTURE RECOGNITION

## Cloud versus Edge AI



# CHALLENGES IN HAND GESTURES

## Technical problems

Improving performance in these areas is essential for making hand gesture recognition systems more practical, reliable, and widely applicable in real-world scenarios.

### Datasets x Data Privacy

Ensuring datasets used for training gesture recognition models are diverse and representativity

### Model Size

It must be compressed and optimized without significant loss of accuracy

### Real-Time Processing

Low-latency processing to provide immediate feedback and smooth interaction in real-time applications

### Gesture Vocabulary

Common shared hand gestures vocabulary for contexts or systems actions

# CHALLENGES IN HAND GESTURES

## Cross-cutting problems

The most critical challenges in hand gesture recognition today include

### HG Education

Is it enough to rely on users' experience and intuitiveness?

### Fluidity

Depends on the perfect integration between the user and the system

### Cultural Prism

Hand gesture recognition must account for the cultural prism, as the meaning and interpretation of gestures can vary significantly across different cultures.

### Shared Vocabulary

A lack of shared vocabulary in hand gesture recognition can lead to inconsistencies and misunderstandings, as different systems and users may interpret gestures differently.

# HAND GESTURE RECOGNITION MODEL


# YOLO11 MODELS

## Object Detection

YOLO11 is an upcoming model in the Ultralytics YOLO family, aiming to push the boundaries of real-time object detection. While official benchmarks are still emerging, YOLO11 builds upon the speed and accuracy of its predecessors with architectural improvements focused on robustness, dynamic input, and better generalization.

| Model | size (pixels) | $mAP^{val}$ 50-95 | Speed CPU ONNX (ms) | Speed T4 TensorRT10 (ms) | params (M) | FLOPs (B) |
|---|---|---|---|---|---|---|
| YOLO11n | 640 | 39.5 | 56.1 ± 0.8 | 1.5 ± 0.0 | 2.6 | 6.5 |
| YOLO11s | 640 | 47.0 | 90.0 ± 1.2 | 2.5 ± 0.0 | 9.4 | 21.5 |
| YOLO11m | 640 | 51.5 | 183.2 ± 2.0 | 4.7 ± 0.1 | 20.1 | 68.0 |
| YOLO11l | 640 | 53.4 | 238.6 ± 1.4 | 6.2 ± 0.1 | 25.3 | 86.9 |
| YOLO11x | 640 | 54.7 | 462.8 ± 6.7 | 11.3 ± 0.2 | 56.9 | 194.9 |

# HAGRID DATASET

## HAnd Gesture Recognition Image Dataset

### 34 Gestures

Includes a broad range of hand gestures like call, dislike, mute, ok, palm, peace, rock, stop, timeout, holy, point, x-sign, among others.

### Multiple Users & Cultures

Captures variations in gesture execution across diverse participants to support cross-cultural generalization.

### Realistic Conditions

Includes varying backgrounds, lighting conditions, and camera angles to train models that perform well in hybrid meeting rooms.

# HAGRIDV2 DATASET

## 1M Subset

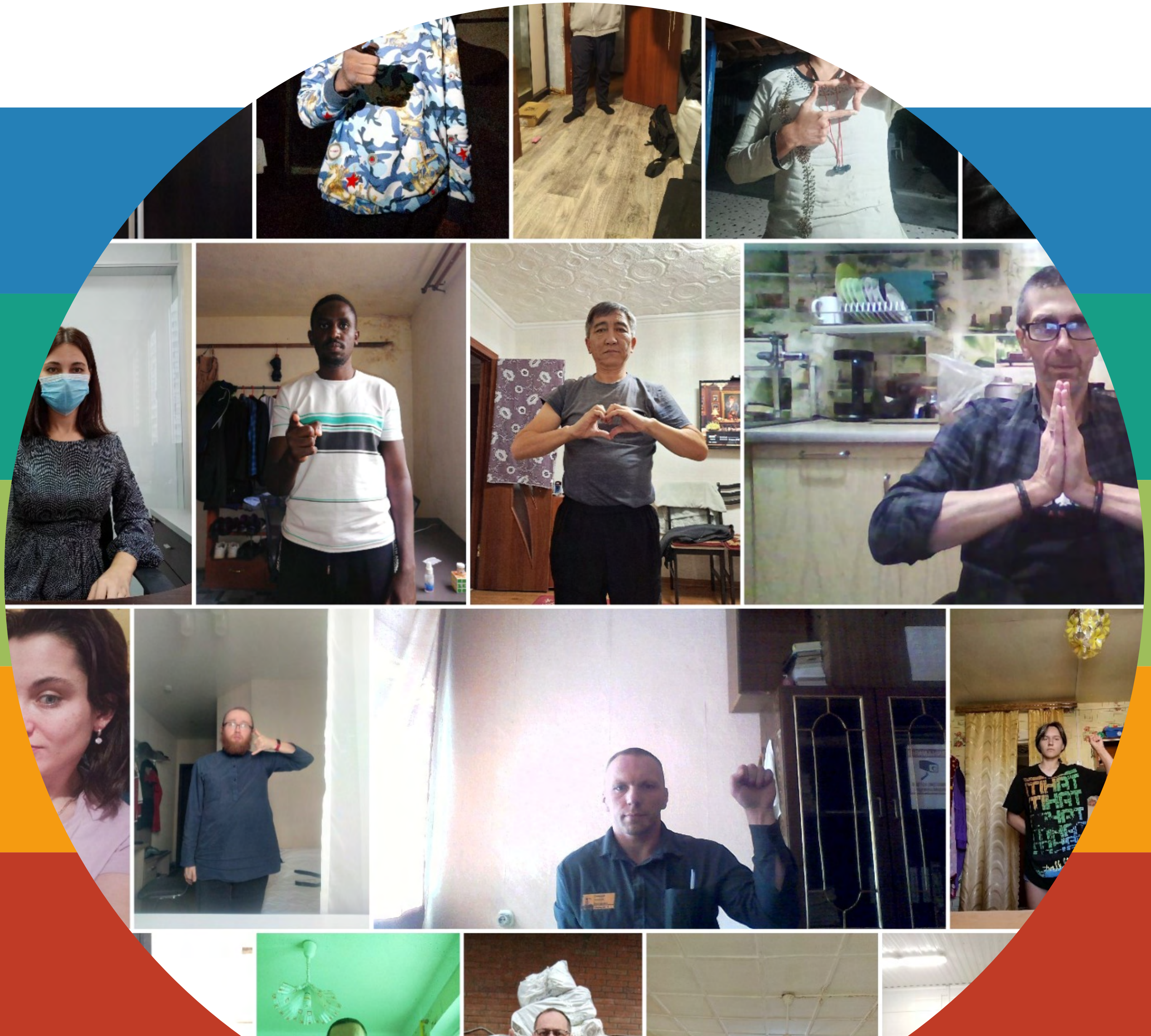
The original hagRIDv2 dataset contains 1 million samples across 33+1 hand gestures, including no-gesture images.

All the images are provided with hand BBOXs and also hand landmarks.

In our subset of the dataset, we randomly choose unto 2500 images per gesture for training and their hand BBOXs.

The subset is available on hugging face

testdummyvt/hagRIDv2_512px_10GB



# YOLO HAGRID MODEL
## Overview

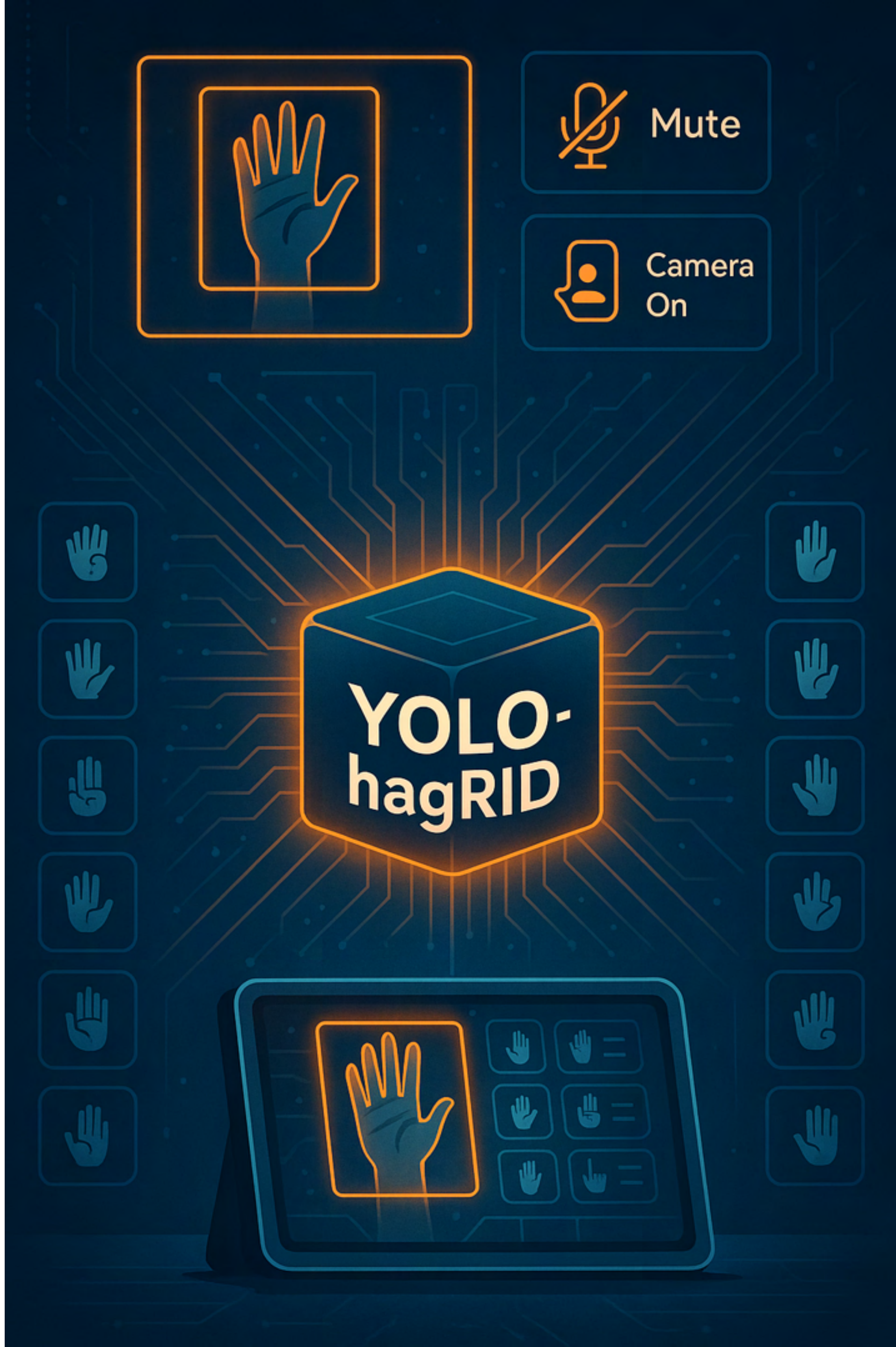
YOLO-hagRID is a customized object detection model trained specifically on the hagRID dataset, which includes 34 hand gesture classes tailored for hybrid meeting interactions.

### 34 Gesture Classes
Trained to recognize a set of hand gestures mapped to meeting platform commands and user interactions.

### Based on YOLO Arch
We utilize a modified version of YOLO for fast and lightweight inference, making it ideal for mobile or embedded deployment.

### hagRID Dataset
Tuned to perform well on real-world gesture data collected across multiple users and settings (including UCP).

### Supports Edge Deployment
Quantized and exportable to ONNX or DLC formats for execution on Snapdragon and other edge devices.



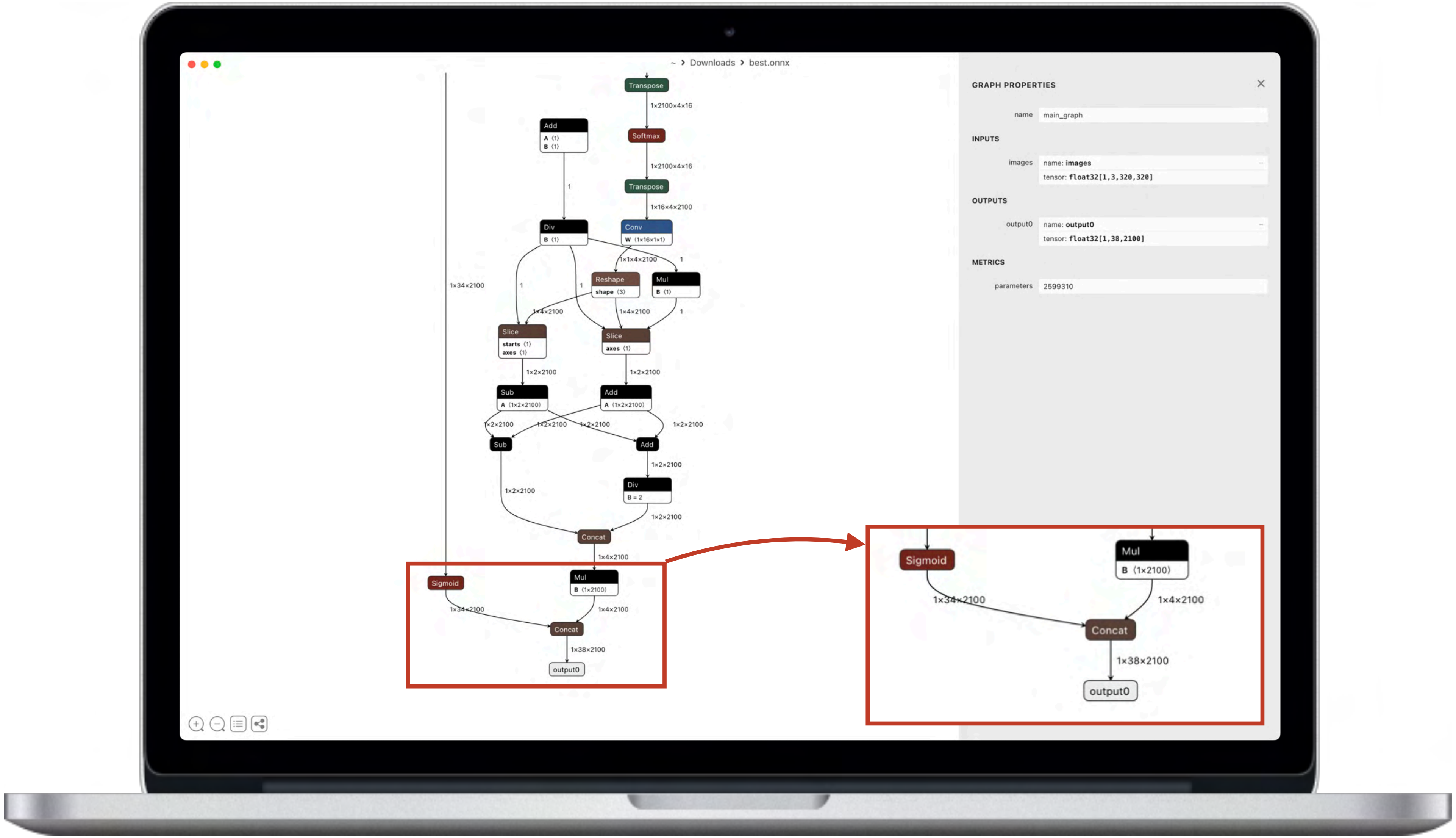
# YOLO HAGRID MODEL

## Trading Hand Gesture Detection Model using Ultralytics + Yolo11N

```python
from ultralytics import YOLO
if _name_ == "_main_":
    model = YOLO("yolo11l.pt")  # load an official model
    dataset_yaml = "/path/to/hagRIDv2_512px_10GB/yolo_format/data.yaml"
    project_dir = "/path/to/expriments"  # Directory to save training results.

    epochs = 10
    imgsz = 640
    device = "cuda"
    workers = 8
    batch = 64
    optimizer = "AdamW"
    lr0 = 0.001

    # Train the model.
    results = model.train(
        data=dataset_yaml,
        epochs=epochs,
        imgsz=imgsz,
        device=device,
        project=project_dir,
        workers=workers,
        batch=batch,
        optimizer=optimizer,
        lr0=lr0)
```

# OUTPUT NORMALIZATION ISSUE

## Concatenation



# OUTPUT NORMALIZATION ISSUE

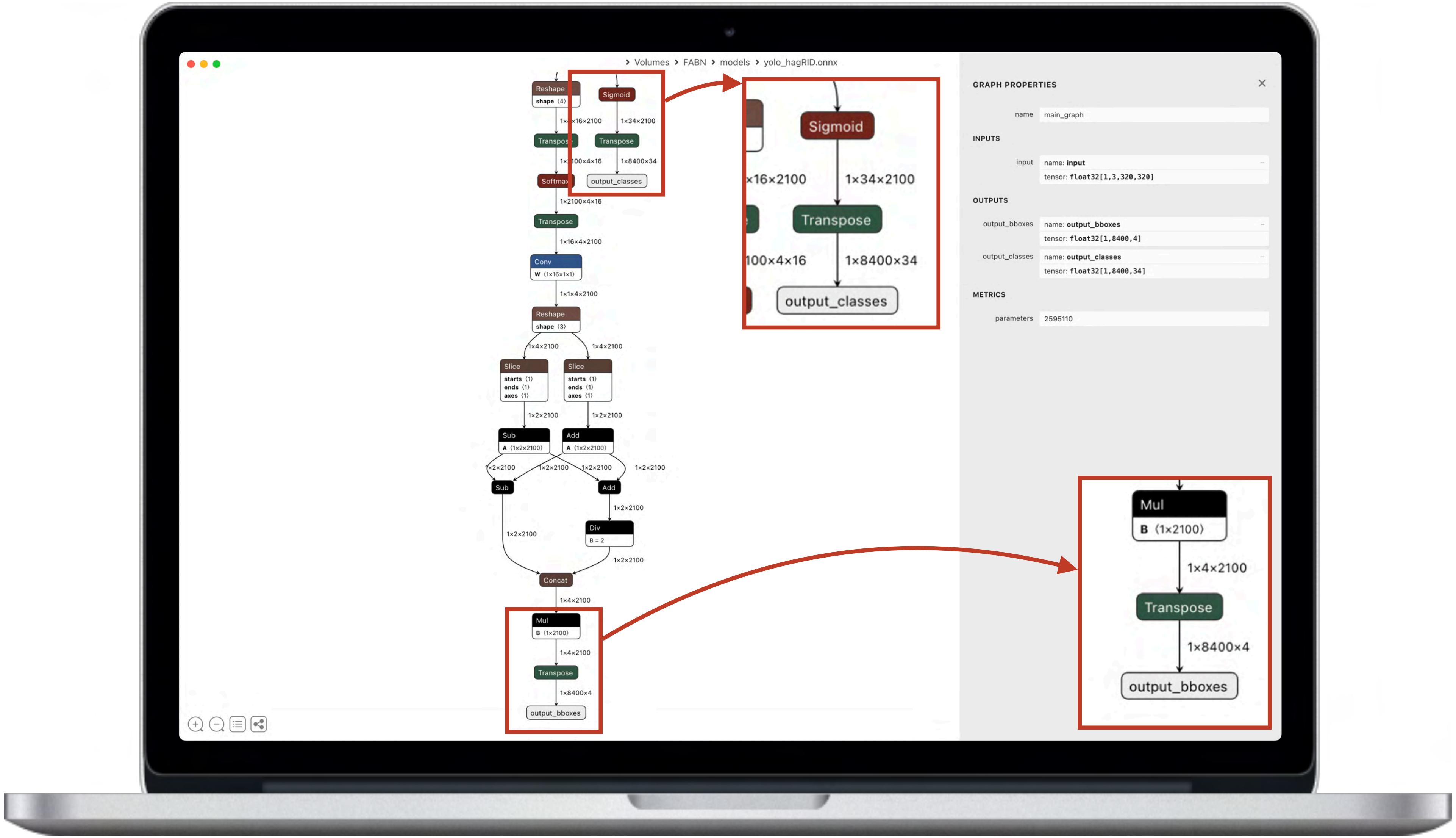
Slicing the output into output_bboxes and output_classes

```
def transform_io_and_prune_node(
  model_path: str, output_path: str,
  new_input_name: str,
  new_output_names: List,
  internal_tensor_names: List,
  remove_node_name: str
):
  model = onnx.load(model_path)

  # Step 1: Rename input.
  old_input_name = model.graph.input[0].name
  model.graph.input[0].name = new_input_name
  for node in model.graph.node:
    node.input[:] = [
      new_input_name if i == old_input_name else i for i in node.input
    ]
  ...
  # Step 5: Replace node list with cleaned + new output nodes.
  model.graph.ClearField("node")
  model.graph.node.extend(original_nodes + new_nodes)

  # Save model.
  onnx.save(model, output_path)
```

GN

# OUTPUT NORMALIZATION ISSUE

## YOLO-hagRID Model with Two Outputs



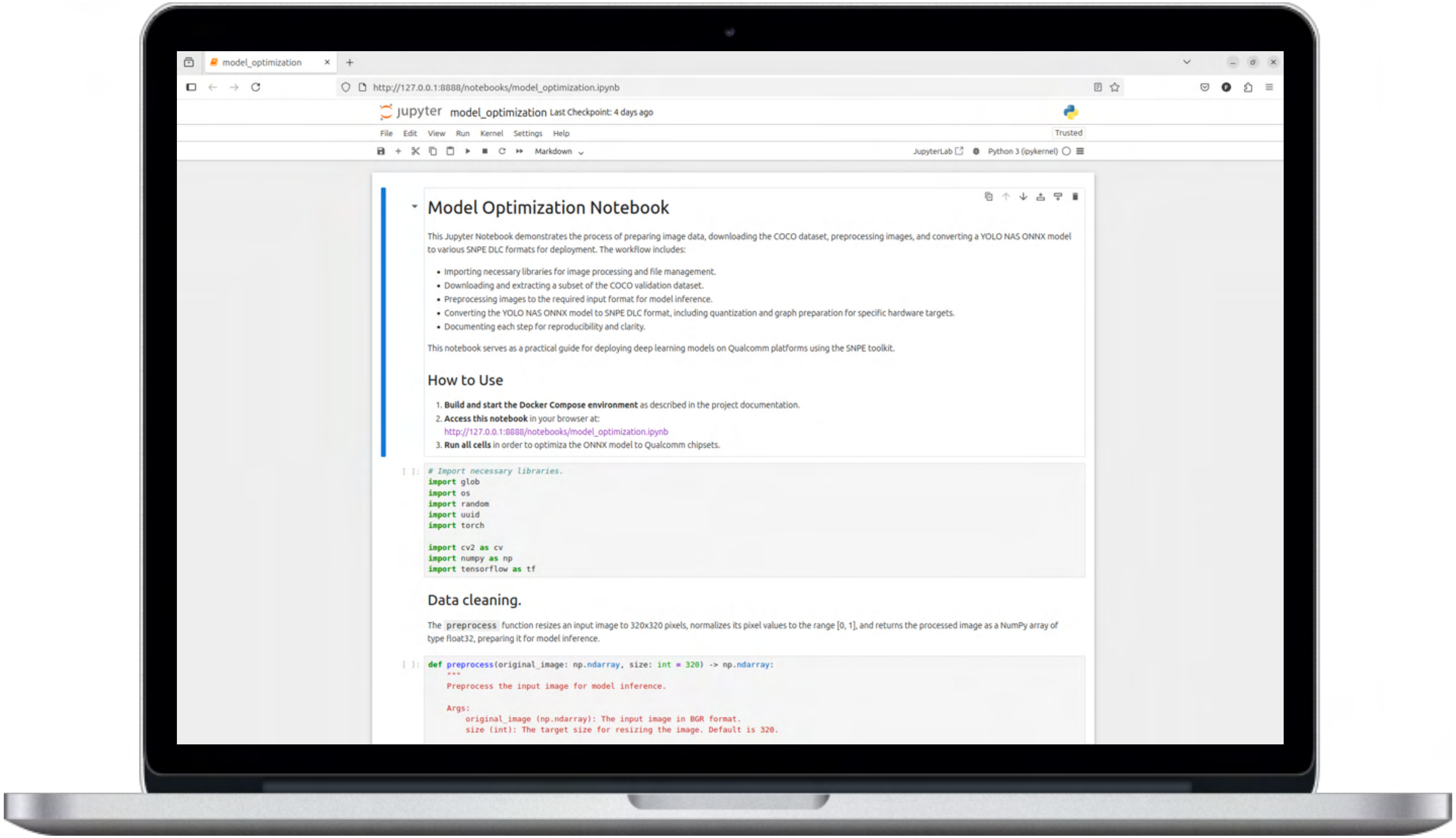
# EXPORTING YOLO-HAGRID TO ONNX

http://127.0.0.1:8889/notebooks/model_zoo.ipynb

```python
# Load the model.
model = YOLO("/models/best.pt")

# Export to ONNX.
model.export(format="onnx", opset=11, imgsz=320)

# Transform the model by renaming inputs, outputs, and pruning a node.
transform_io_and_prune_node(
    model_path="/models/best.onnx",
    output_path="/models/yolo_hagRID.onnx",
    new_input_name="input",
    new_output_names=[
        "output_bboxes",
        "output_classes"
    ],
    internal_tensor_names=[
        "/model.23/Mul_2_output_0",
        "/model.23/Sigmoid_output_0"
    ],
    remove_node_name="/model.23/Concat_5"
)
```

# MODEL OPTIMIZATION NOTEBOOK

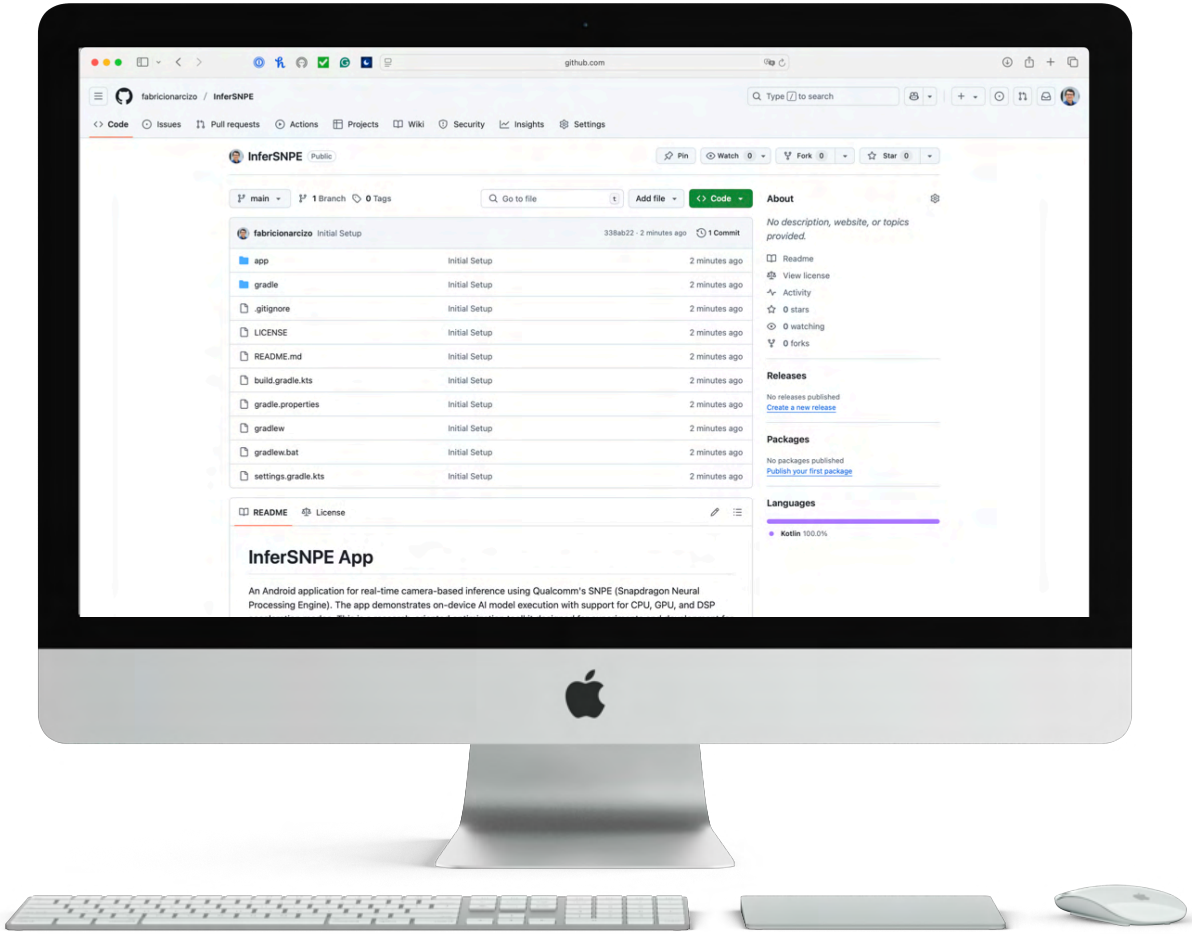
http://127.0.0.1:8888/notebooks/model_optimization.ipynb

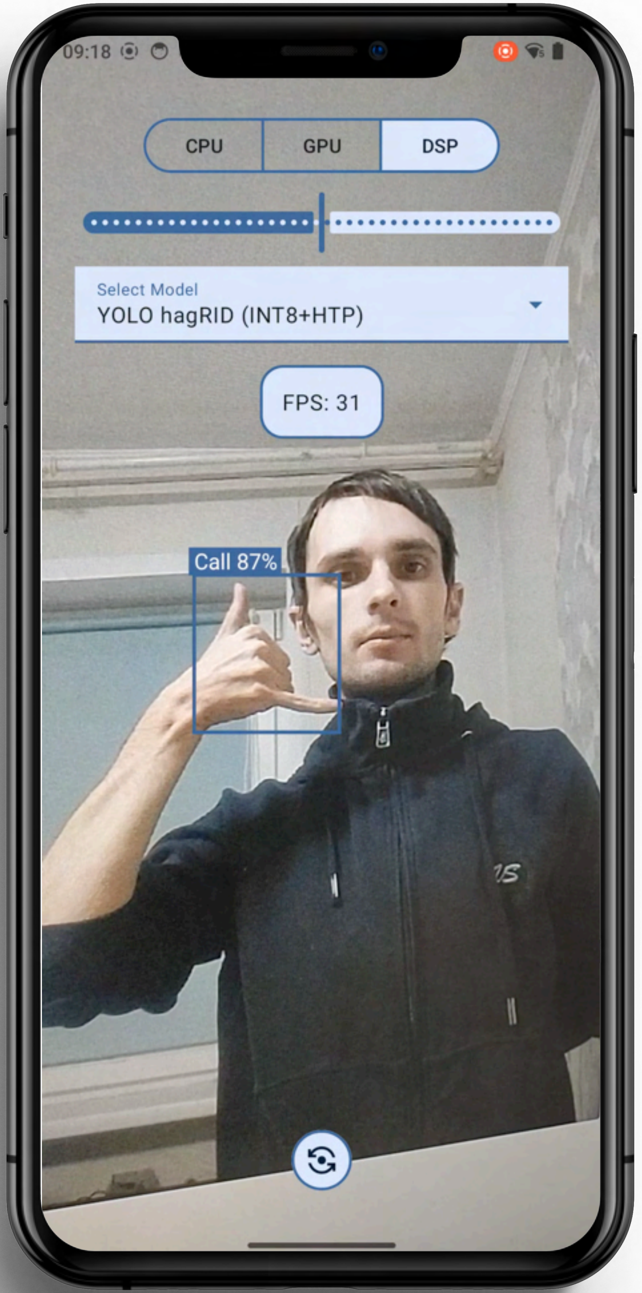


# INFERSNPE APP

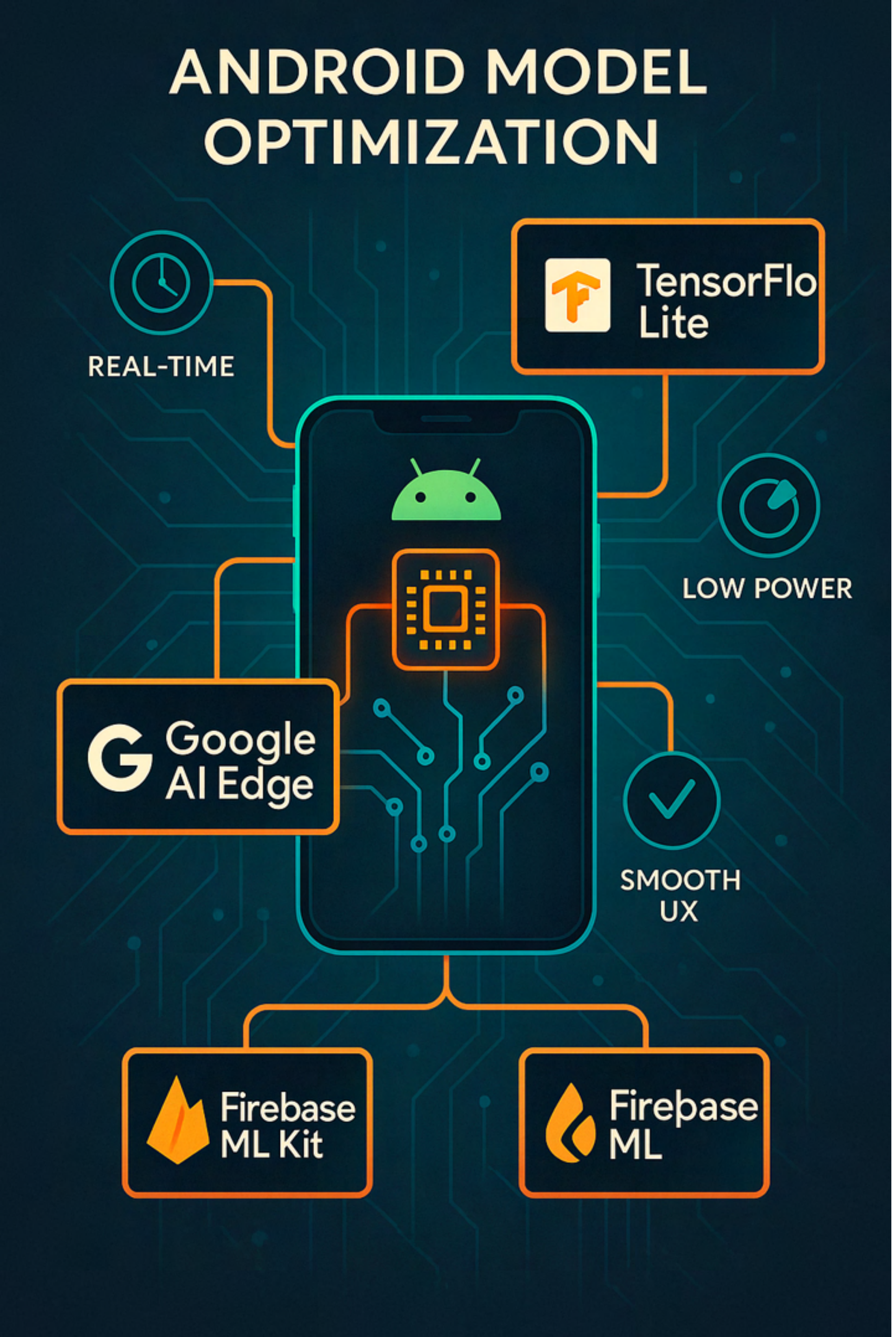
https://github.com/fabricionarcizo/InferSNPE

FAIRPHONE



# OPTIMIZATION FOR MOBILE PHONES




# OPTIMIZATION FOR ANDROID

## Overview

Optimizing models for Android devices is crucial to ensure real-time performance, low power consumption, and smooth user experiences. Several tools and platforms support this goal:

### TensorFlow Lite

A lightweight inference engine designed for mobile and embedded devices; supports quantization, GPU acceleration, and hardware delegation.

### Google AI Edge

Framework for deploying models on supported Android devices with AI chips (e.g., Edge TPU, NPUs), offering hardware acceleration and optimized runtimes.

### Firebase ML Kit

Provides pre-trained and custom model support with simplified APIs for tasks like image labeling, object detection, and translation.

# ONNX TO TENSORFLOW

onnx2tf

The *onnx2tf* tool enables the conversion of ONNX models to TensorFlow-compatible formats, such as *.pb* or *.tflite*. This is particularly useful when deploying models trained in PyTorch or exported to ONNX into Android apps.

### Cross-Framework

Converts models from ONNX to TensorFlow, making them usable in TFLite and Android.

### onnx2tf Toolchain

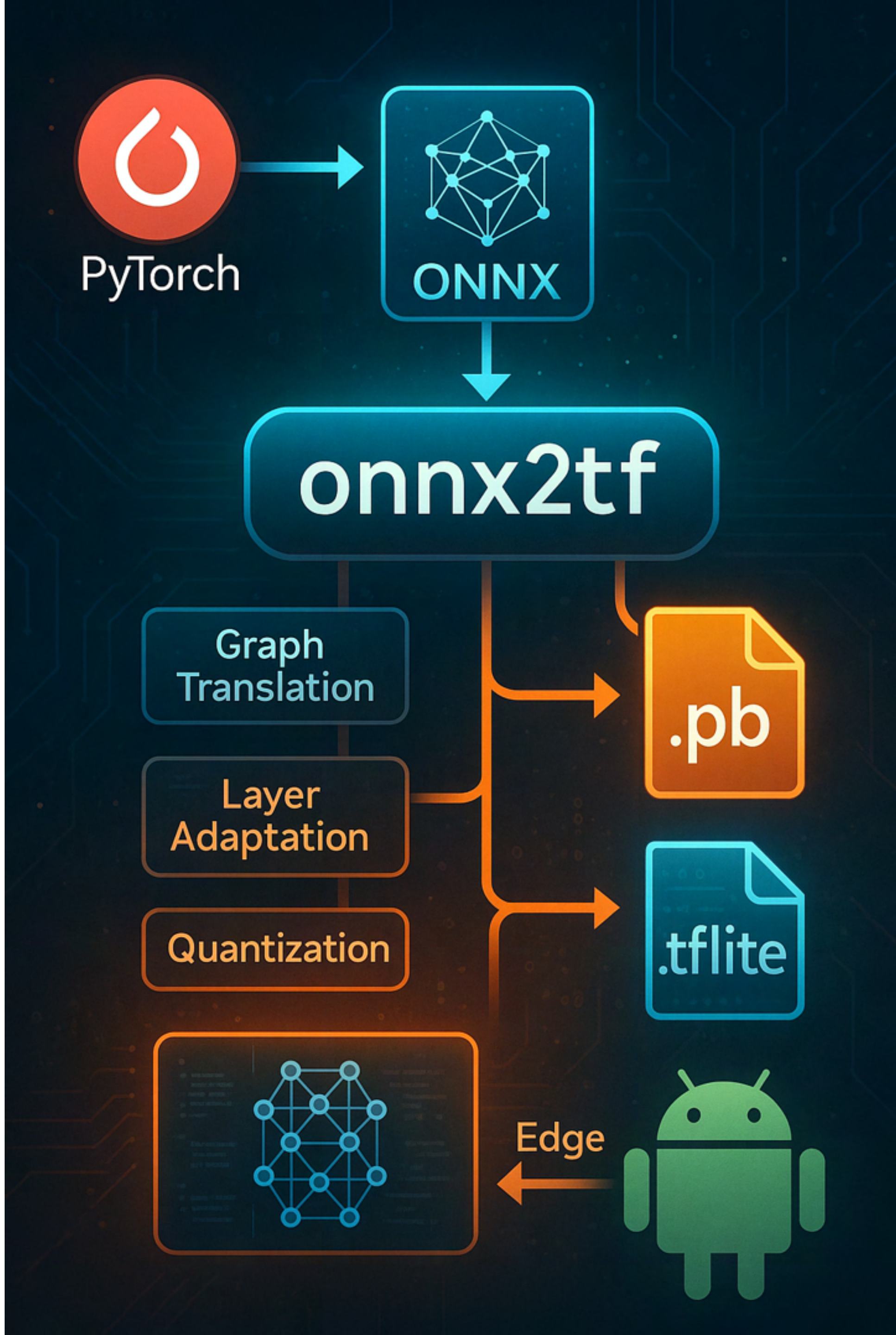
Open-source converter that maps ONNX operators to equivalent TensorFlow operations.

### Quantized Output

Facilitates post-training quantization to produce TFLite models suitable for low-power edge deployment.

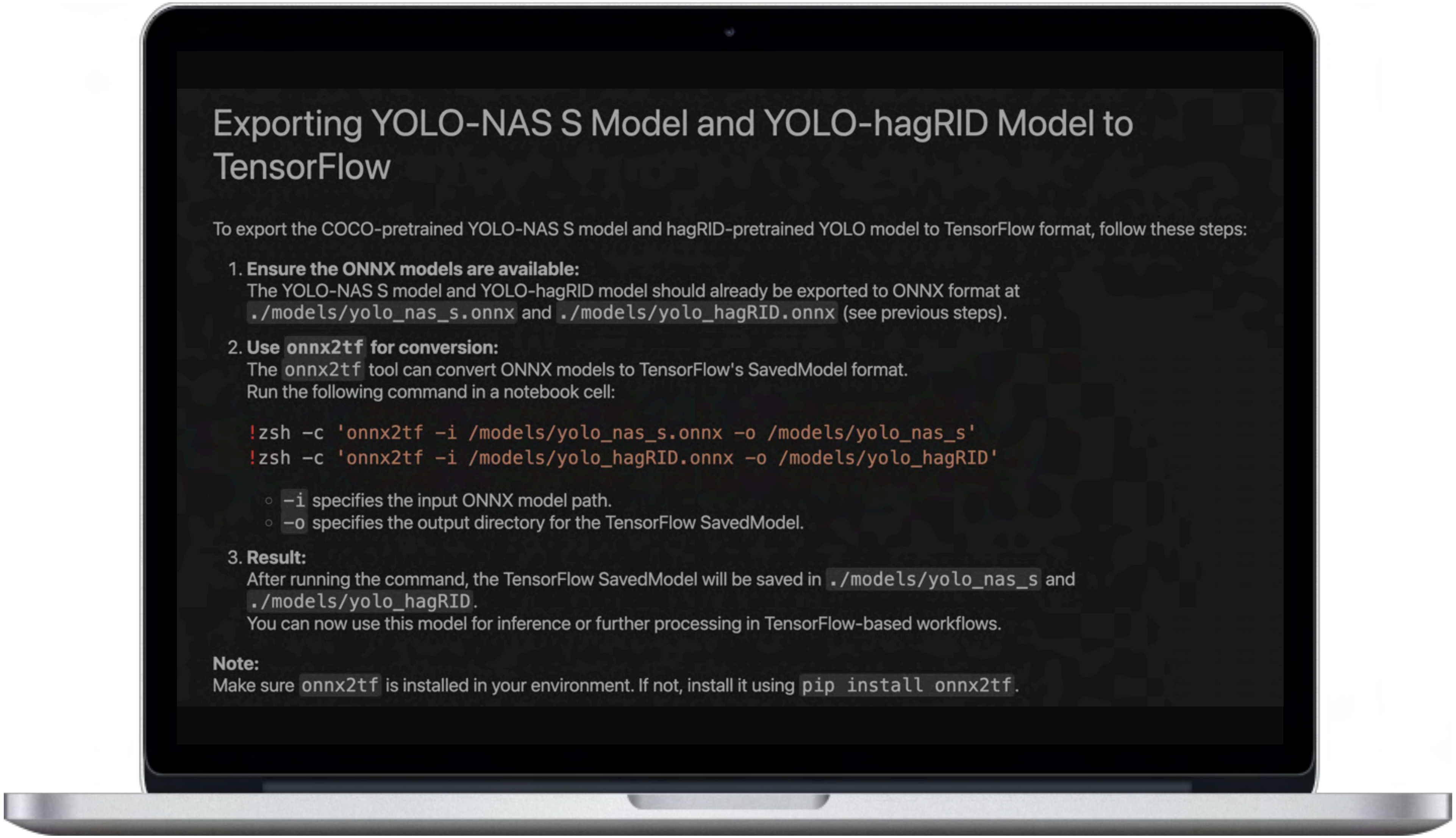
### Android Deployment

Enables seamless migration of PyTorch-trained models to TensorFlow-based mobile inference environments.



# EXPORTING ONNX TO TENSORFLOW

http://127.0.0.1:8889/notebooks/model_zoo.ipynb



## Exporting YOLO-NAS S Model and YOLO-hagRID Model to TensorFlow

To export the COCO-pretrained YOLO-NAS S model and hagRID-pretrained YOLO model to TensorFlow format, follow these steps:

1. **Ensure the ONNX models are available:**
   The YOLO-NAS S model and YOLO-hagRID model should already be exported to ONNX format at `./models/yolo_nas_s.onnx` and `./models/yolo_hagRID.onnx` (see previous steps).
2. **Use `onnx2tf` for conversion:**
   The `onnx2tf` tool can convert ONNX models to TensorFlow's SavedModel format.
   Run the following command in a notebook cell:

   ```
   !zsh -c 'onnx2tf -i /models/yolo_nas_s.onnx -o /models/yolo_nas_s'
   !zsh -c 'onnx2tf -i /models/yolo_hagRID.onnx -o /models/yolo_hagRID'
   ```

   - `-i` specifies the input ONNX model path.
   - `-o` specifies the output directory for the TensorFlow SavedModel.
3. **Result:**
   After running the command, the TensorFlow SavedModel will be saved in `./models/yolo_nas_s` and `./models/yolo_hagRID`.
   You can now use this model for inference or further processing in TensorFlow-based workflows.

**Note:**
Make sure `onnx2tf` is installed in your environment. If not, install it using `pip install onnx2tf`.

# CONVERTING MODELS TO FLOAT32

http://127.0.0.1:8888/notebooks/model_optimization.ipynb

```
for model in ["yolo_nas_s", "yolo_hagRID"]:

    # Step 1: Load the SavedModel.
    saved_model_dir = f"/models/{model}"
    converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)

    # Step 2: Restrict to float32 operations only (for max delegate
    # compatibility).
    converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS]

    # Step 3: Convert the model.
    tflite_model = converter.convert()

    # Step 4: Save the model.
    output_path = f"/models/{model}_float32.tflite"
    with open(output_path, "wb") as f:
        f.write(tflite_model)
```

# CONVERTING MODELS TO FLOAT16

http://127.0.0.1:8888/notebooks/model_optimization.ipynb

```python
for model in ["yolo_nas_s", "yolo_hagRID"]:

    # Step 1: Load the SavedModel.
    saved_model_dir = f"/models/{model}"
    converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)

    # Step 2: Enable optimization.
    converter.optimizations = [tf.lite.Optimize.DEFAULT]

    # Step 3: Set float16 as the target precision.
    converter.target_spec.supported_types = [tf.float16]

    # Step 4: Use only float ops (TFLITE_BUILTINS).
    converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS]

    # Step 5: Convert the model.
    tflite_model = converter.convert()

    # Step 6: Save the converted model.
    output_path = f"/models/{model}_float16.tflite"
    with open(output_path, "wb") as f:
        f.write(tflite_model)
```

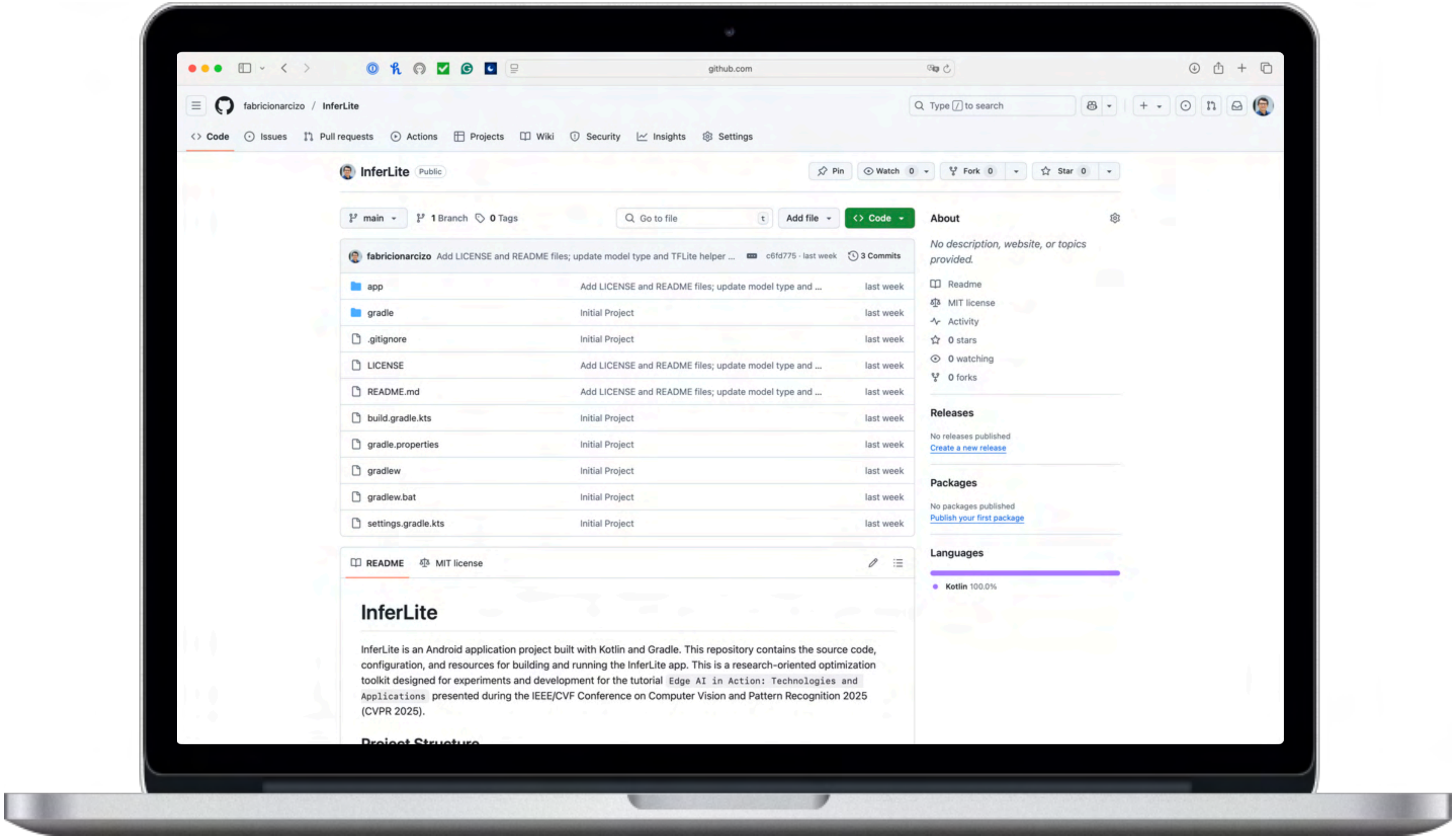
# CONVERTING MODELS TO INT8

http://127.0.0.1:8888/notebooks/model_optimization.ipynb

```python
for model in ["yolo_nas_s", "yolo_hagRID"]:
    # Step 1: Load the SavedModel.
    saved_model_dir = f"/models/{model}"
    converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
    # Step 2: Enable optimizations.
    converter.optimizations = [tf.lite.Optimize.DEFAULT]
    # Step 3: Define representative dataset generator.
    def representative_data_gen():
        for _ in range(100):
            dummy_input = np.random.rand(1,320,320,3).astype(np.float32)
            yield [dummy_input]
    converter.representative_dataset = representative_data_gen
    # Step 4: Set supported operations and data types for full Int8.
    converter.target_spec.supported_ops = [
        tf.lite.OpsSet.TFLITE_BUILTINS,      # Float32
        tf.lite.OpsSet.TFLITE_BUILTINS_INT8  # Allow fallback ops
    ]
    # Step 5: Convert the model.
    tflite_model = converter.convert()
    # Step 6: Save the optimized model to a file
    output_path = f"/models/{model}_int8.tflite"
    with open(output_path, "wb") as f:
        f.write(tflite_model)
```

# INFERLITE APP

https://github.com/fabricionarcizo/InferLite



# APP GRADLE FILE

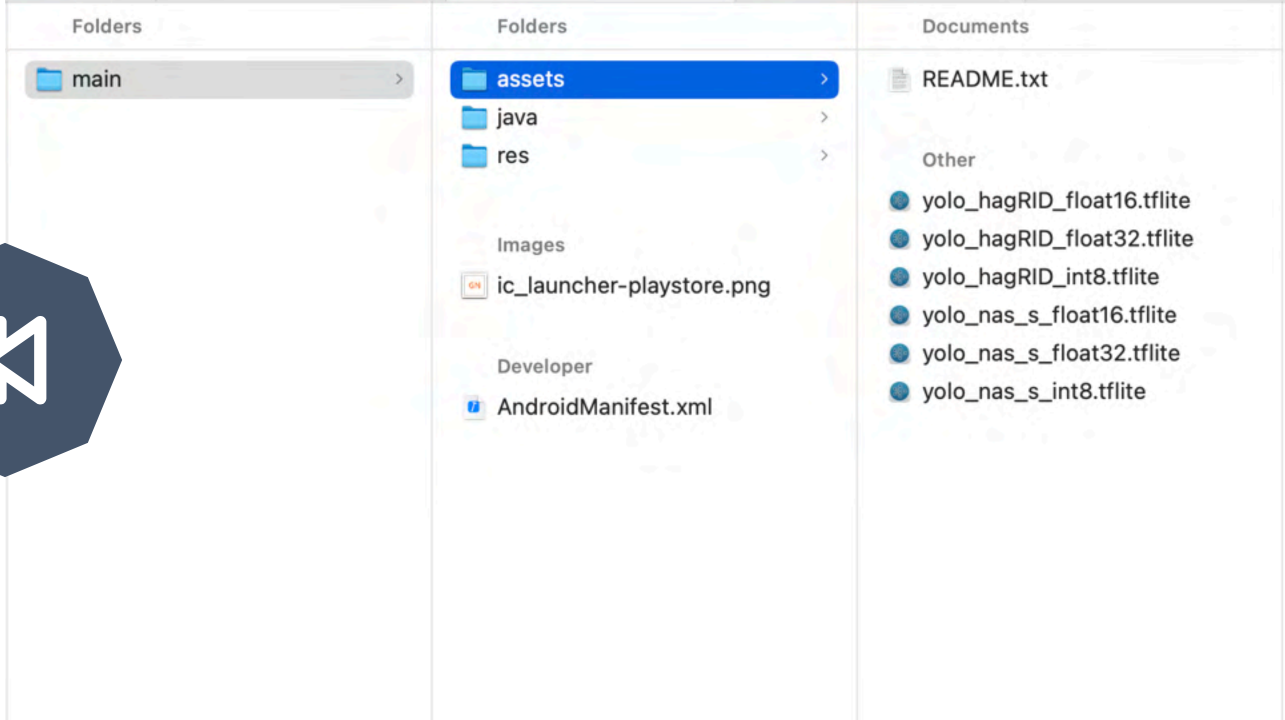
## build.gradle.kts

```
android {
    defaultConfig {
        applicationId = "com.gn.videotech.inferlite"
        minSdk = 24
        targetSdk = 35
        versionCode = 1
        versionName = "1.0"
    }
    ...
}

dependencies {
    implementation(libs.androidx.appcompat)
    implementation(libs.androidx.camera.camera2)
    implementation(libs.androidx.camera.lifecycle)
    implementation(libs.androidx.camera.view)
    ...
    implementation(libs.tensorflow.lite)
    implementation(libs.tensorflow.lite.gpu)
    implementation(libs.tensorflow.lite.gpu.api)
    implementation(libs.tensorflow.lite.gpu.delegate.plugin)
    implementation(libs.tensorflow.lite.support)
}
```

TensorFlow Lite

## TFLITE FILES (ASSETS)

Hardware-optimized model files generated by TFLiteConverter from TensorFlow sources; required for deployment on NNAPI units.



# TFLITEHELPER CLASS

## Overview

TFLiteHelper class is a utility component that simplifies the process of loading and running TensorFlow Lite models on Android. It allows developers to integrate AI functionality into mobile apps.

### Model Initialization

Loads the .tflite from assets and prepares the interpreter with optional hardware acceleration.

### Input Preprocessing

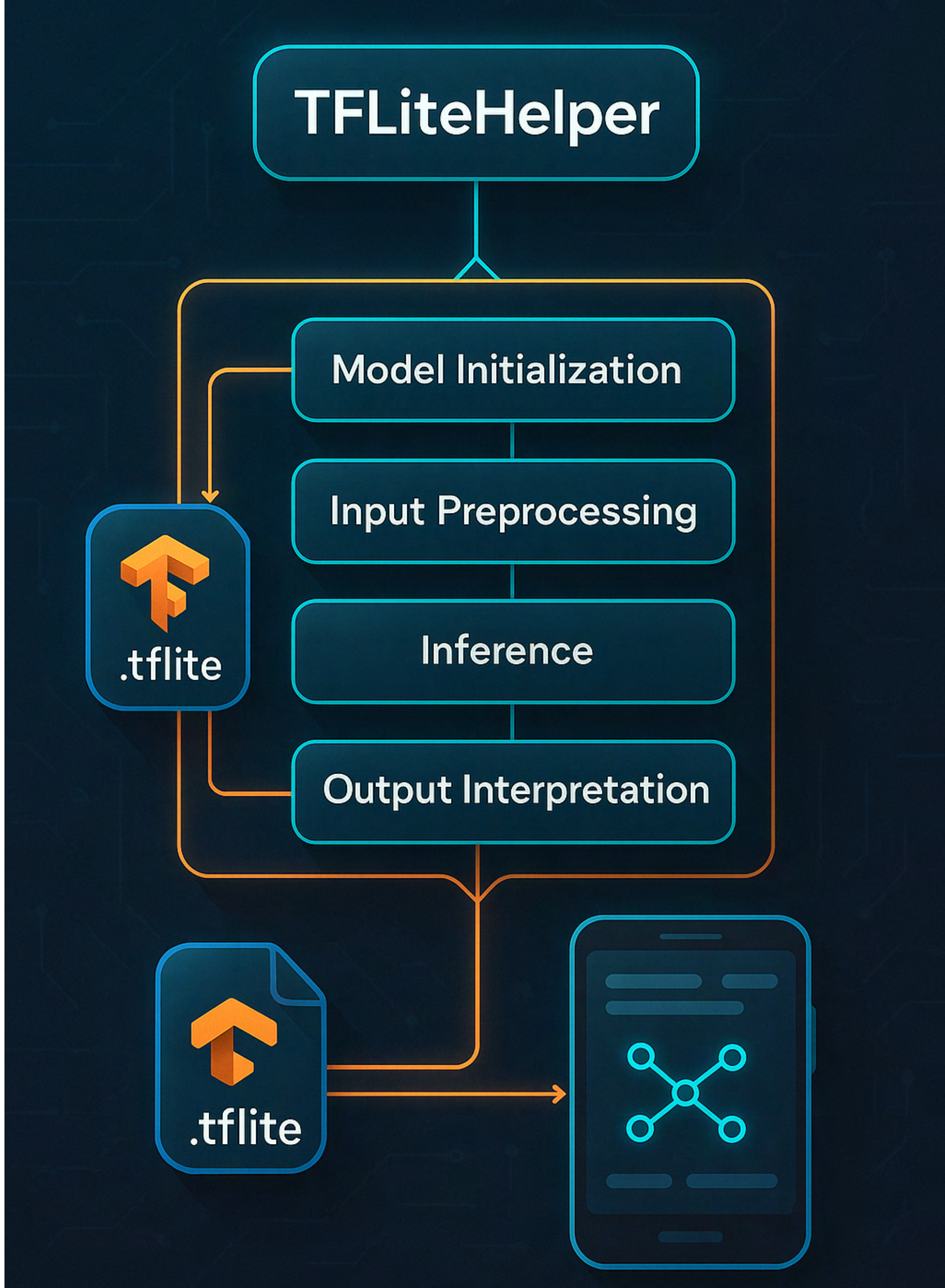
Normalizes and reshapes input images to match the model's expected input format.

### Inference Execution

Runs inference using the TFLite interpreter and captures the raw output tensors.

### Output Parsing

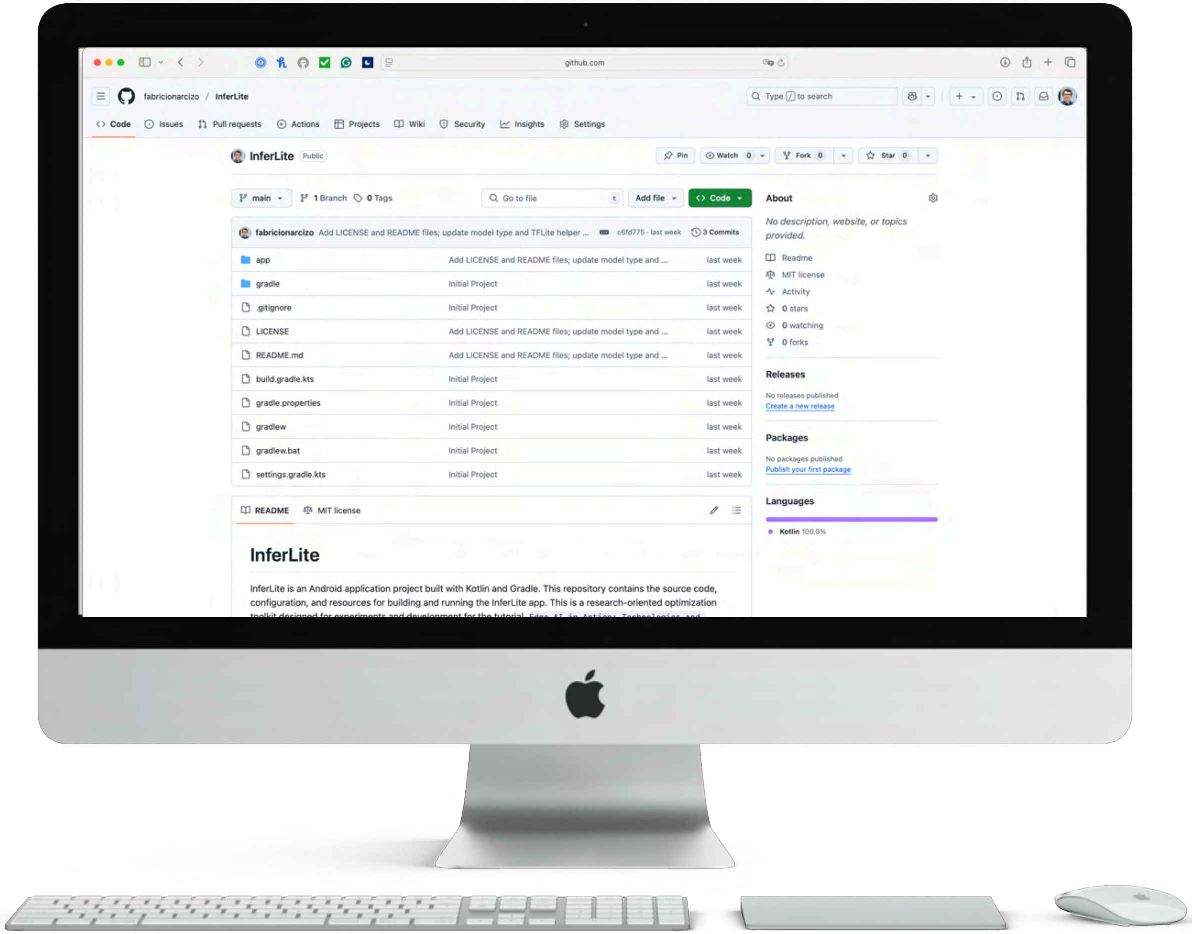
Converts tensor outputs into human-readable results such as class labels, confidence scores, or bounding boxes.

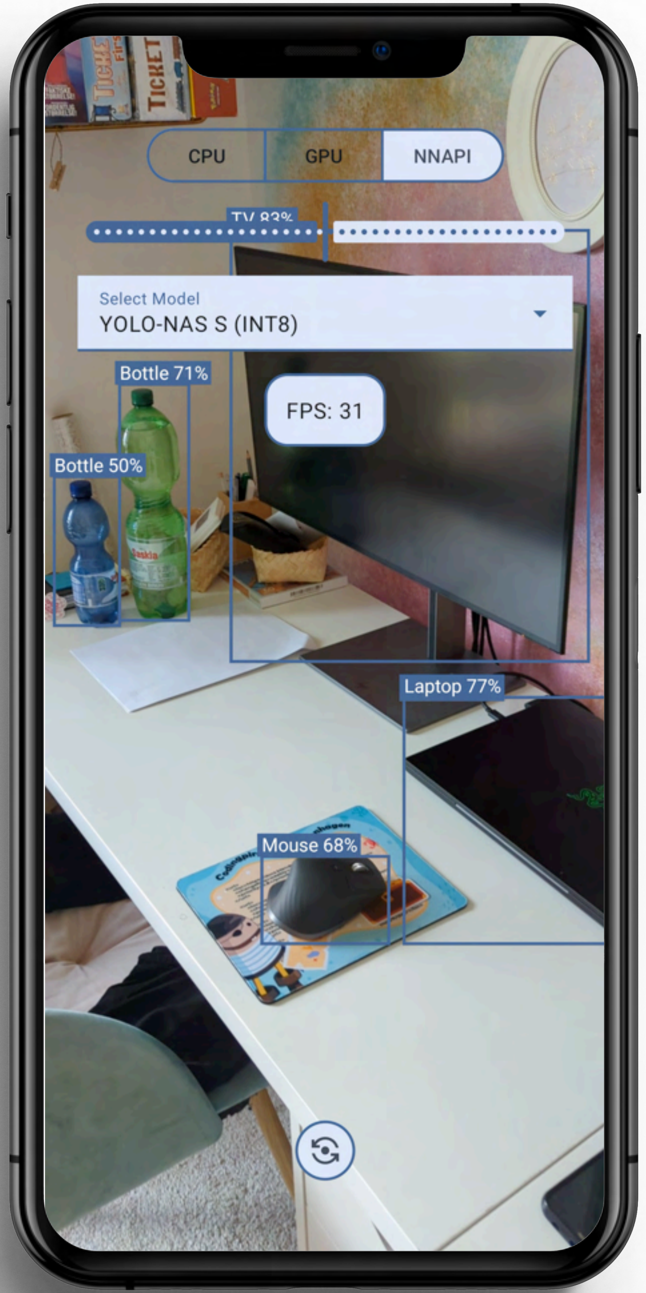


# INFERLITE APP

https://github.com/fabricionarcizo/InferLite

Pixel



# QUESTIONS & ANSWERS

# THANK YOU!