Why quantize?

What is Quantization? The process of reducing precision to speed up computations and decrease size.

4x

Smaller

up to 4x

Faster

Reduced Size: Reducing model size enables better storage on limited-capacity devices.

Faster Processing: Accelerates computation, increasing model performance on low-power devices.

Energy Efficiency: Lower precision consume less power, crucial for battery-operated devices.

What is Quantization?

Floating Point (32 bits per value)

12.5	-203.4	-60.3	40.8
-6.2	-300.0	-25.6	32.6

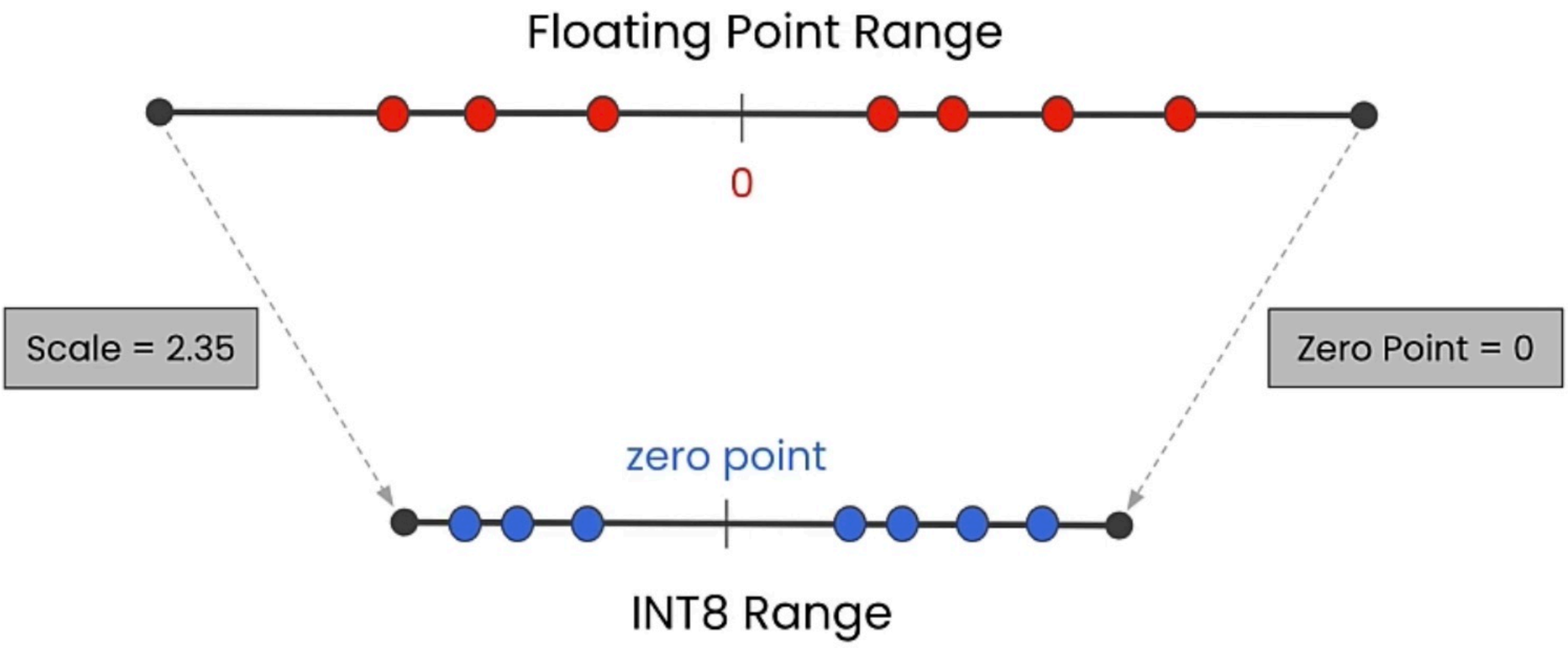
Integer (8 bits per value)

5	86	-26	17
-3	-127	11	14

Scale = 2.35

Zero Point = 0

What is Quantization?

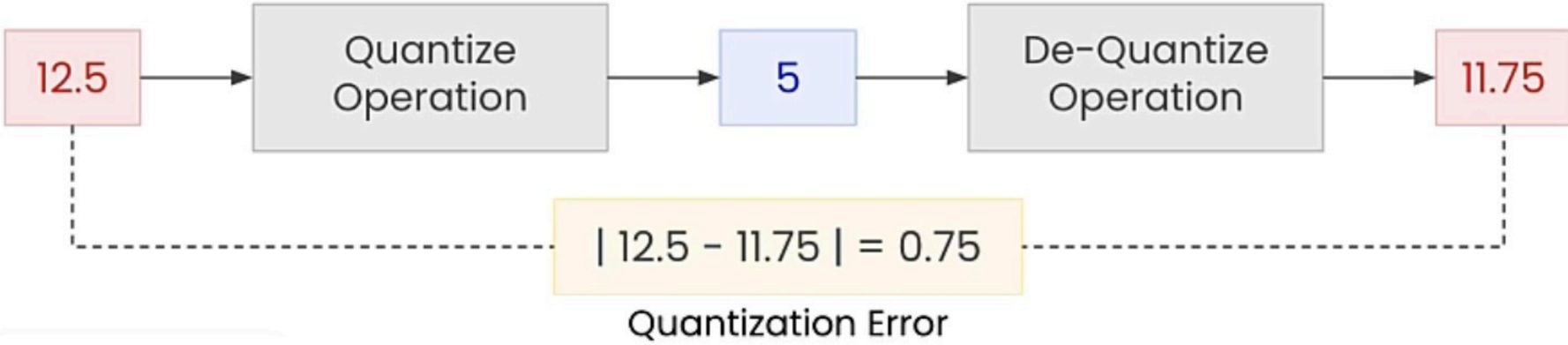


$$\text{Round}\left|\frac{12.5}{2.35}\right| - 0 = 5$$

Quantize Operation

$$(5 + 0) \times 2.35 = 11.75$$

De-Quantize Operation



Types of Quantization

Weight Quantization: Reduces the precision of model weights to optimize storage and computational speed.

Activation Quantization: Applies lower precision to activation values to accelerate inference and reduce memory usage.

FP32 Weight

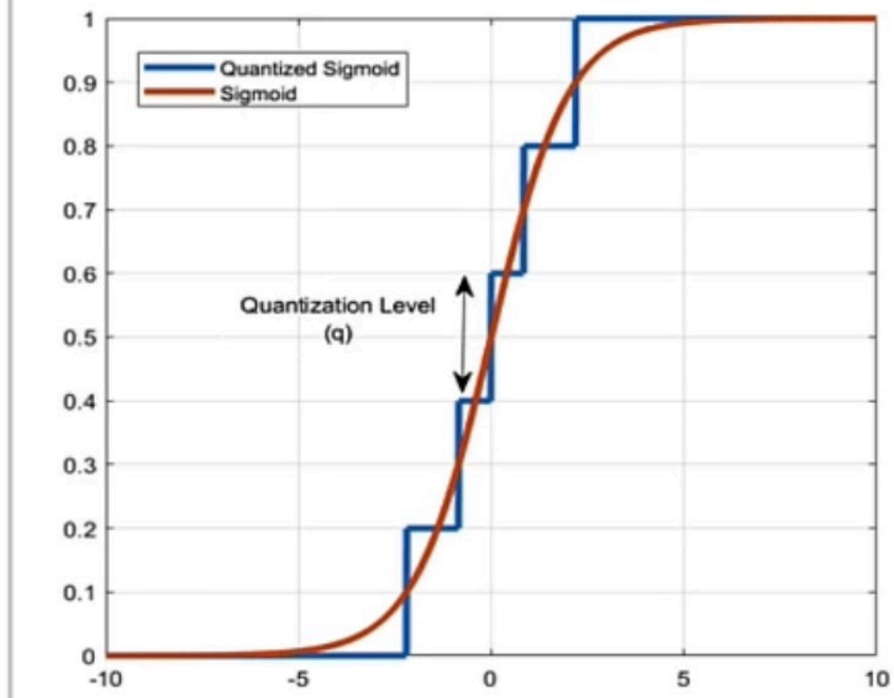
12.5	-203.4	-60.3	40.8
-6.2	-300.0	-25.6	32.6

↓

5	86	-26	17
-3	-127	11	14

INT8 Weight

Weight Quantization



Activation Quantization

W8INT8: Weights to 8-bit, activations to 8-bit integer

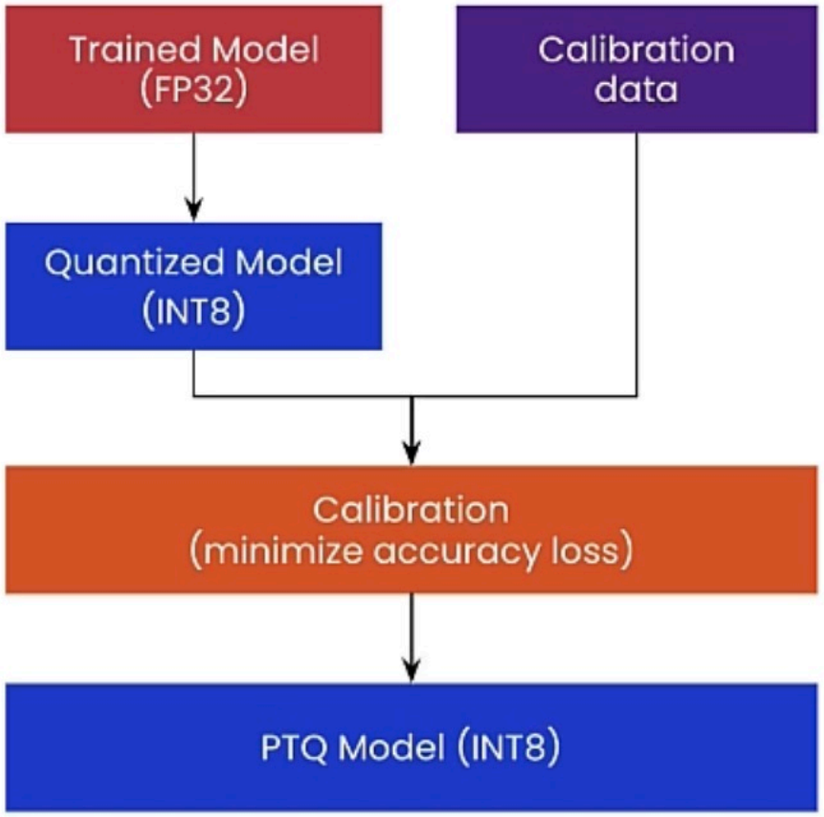
W8INT16: Weights to 8-bit, activations to 16-bit integer

W4INT16: Weights to 4-bit, activations to 16-bit integer

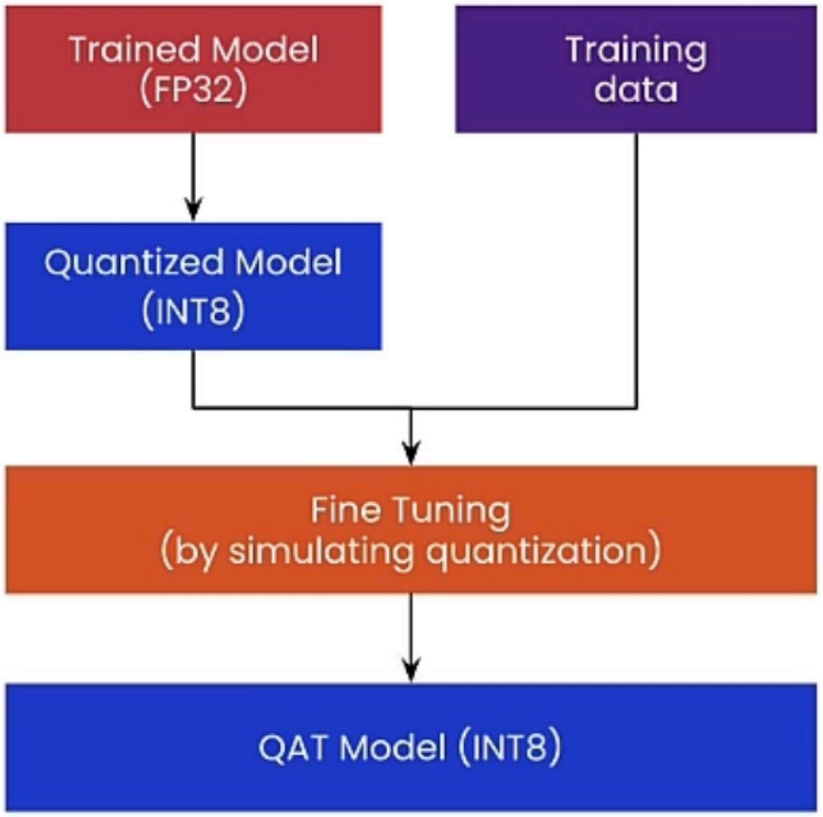
Types of Quantization

Post Training Quantization (PTQ): Quantization applied after model has been trained by calibrating the trained model using sample data.

Quantization Aware Training (QAT): Quantization applied into the training process of the model by simulating effects of quantization.



Post-Training Quantization



Quantization Aware Training

Adds scales & zero points to all weights **and converts** floating points weights to integer weights

Re-learns entirely new integer parameters in the model as quantization involves re-training.

Impact of Quantization

4x

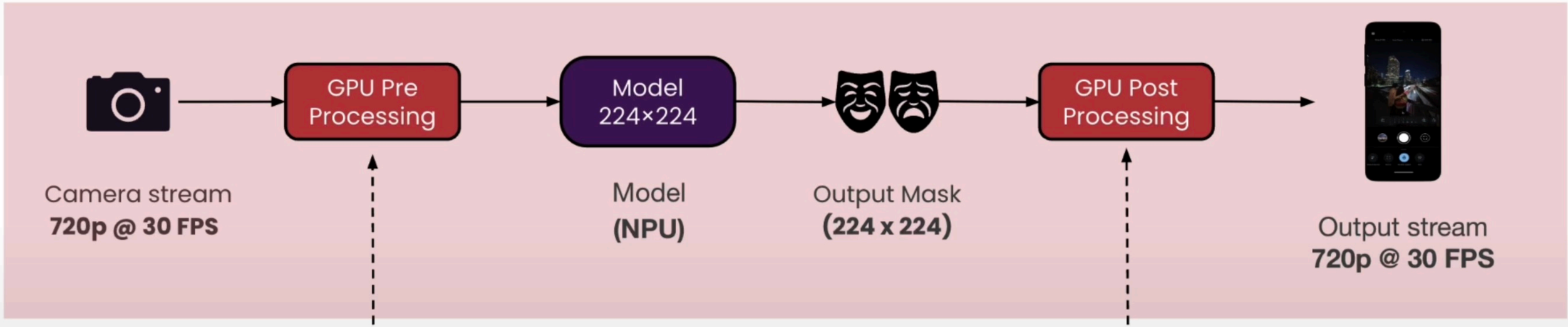
Smaller

up to 4x

Faster

Variant	Model Size (MB)		Latency (msec)		Speedup
	FP32	INT8	FP32	INT8	
FFNet 40s	55.6	13.5	16.9	4.6	**3.7**
FFNet 54s	72.2	17.5	18.4	5.1	**3.6**
FFNet 78s	109.9	26.5	21.7	5.9	**3.7**

Device Integration



- YUV to RGB conversion
- Downsample (to 224 x 224)

- Exponential smoothing
- Upsample (to input resolution)
- Threshold (to 0.55)
- Normalized box-blur

How is the application implemented?

Camera Stream: Extract RGB or YUV data form the camera stream at 30 FPS. Setup camera stream for processing.

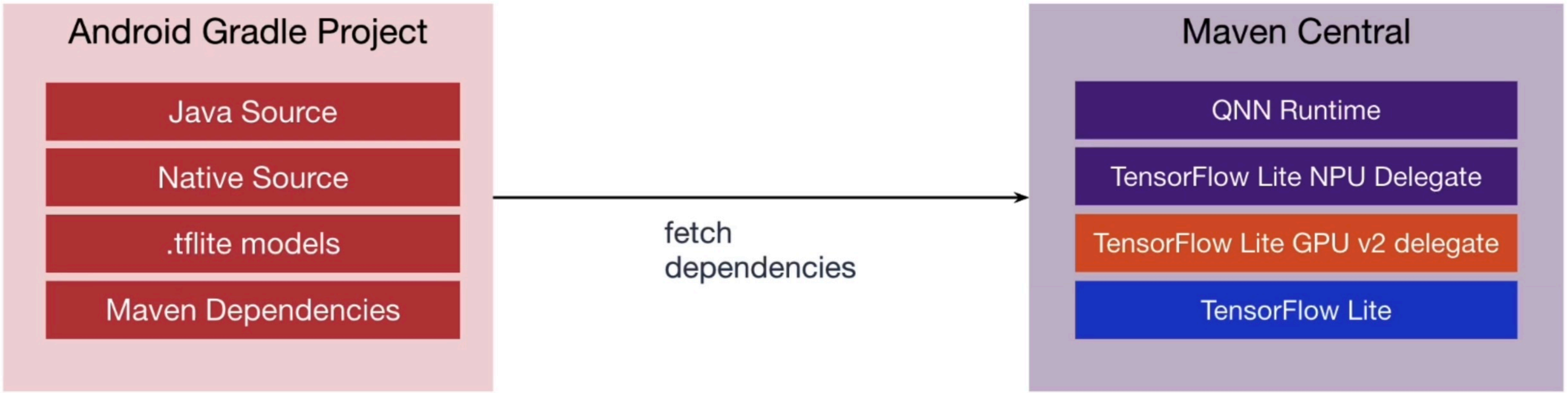
Implement Pre-processing: Use OpenCV on the GPU for faster pre-processing.

Model Inference: Use the runtime APIs (C++, Java) for model inference.

Implement Post-processing: Use OpenCV on the GPU for faster post-processing.

Packaging Runtime: Make sure to package all the runtime dependencies for hardware acceleration.

Runtime Dependencies



Models : Package models with application

Libraries: Break down library dependencies for CPU, GPU, NPU for optimal delivery

Bundle vs Over-the-air: Both libraries & models can be bundled or delivered over-the-air

Demo: Real Time Segmentation

Frames Per Second (FPS): Real time semantic segmentation running at ~30 FPS. Detecting outdoor segments.

Compute Unit Utilization: Uses both GPU, and NPU

- Fully quantized model running entirely on NPU
- Pre/Post Processing running on GPU

Compatibility: Across all Android phones. Runs on NPU on Qualcomm powered Android phones released post 2019. GPU on lower end phones.

Did you know?

Smartphone Camera: 20+ models run with each picture taken

Industrial IOT: Estimated economic impact of on-device AI is $3.1T

ADAS: Advanced driver assistance is entirely on-device AI based

On-device AI is Everywhere

Write

Speak

Deliver

Edit

Drive

See

Assemble

Applicable Use-Cases

Audio & Speech : Text-to-speech, Speech Recognition, Machine Translation, Noise Removal

Images & Video: Photo classification, QR code detection, Virtual Background Segmentation

Sensors: Keyboards, Physical Activity Detection, Digital handwriting recognition

Audio & Speech

Image & Video

Sensors & Text

Applicable Industries

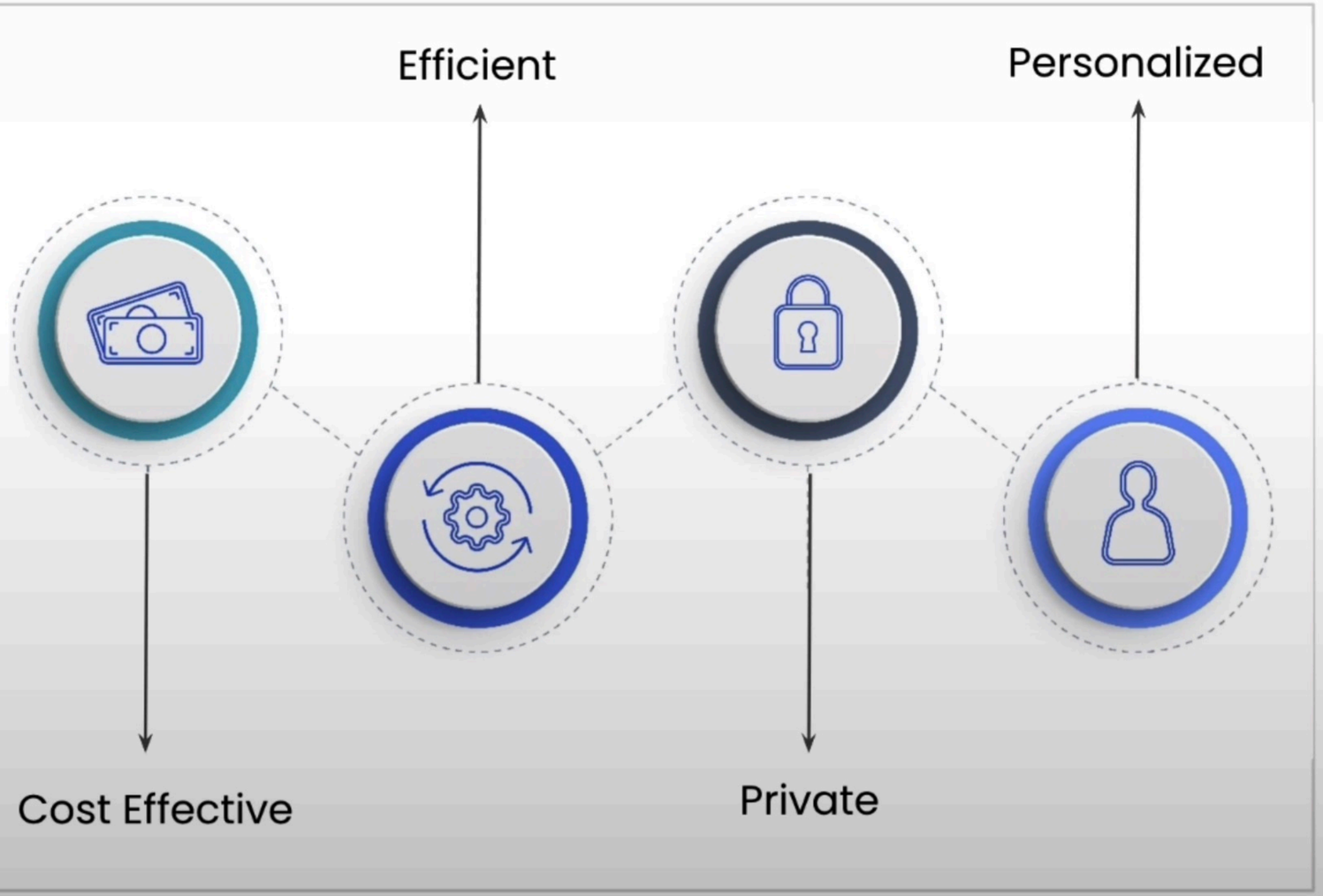
Mobile

PC

Industrial IOT

Automotive

Why on-device?



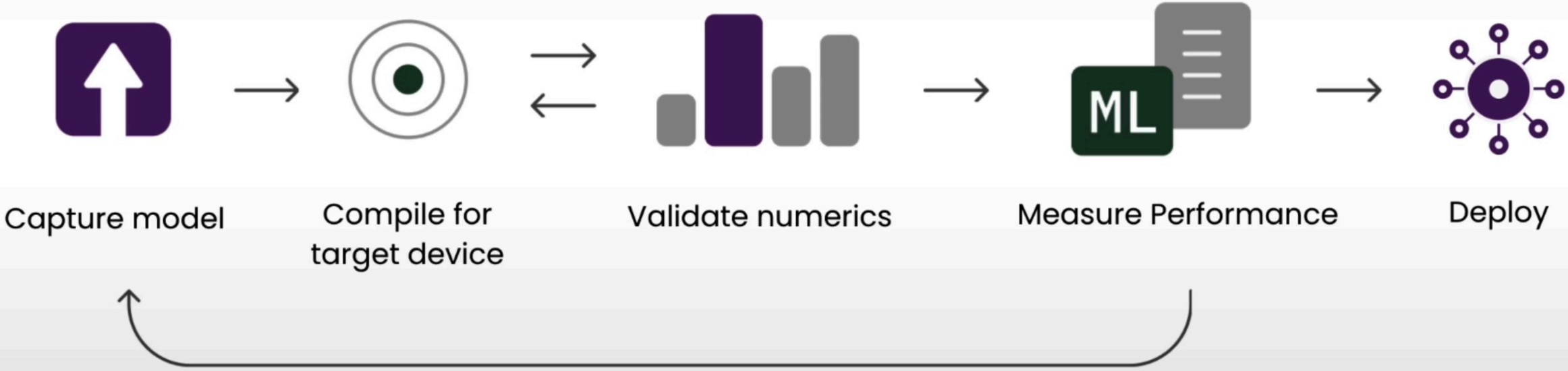
Cost Effective: Reduces recurring costs by minimizing dependency on cloud computing resources.

Efficient: Faster processing speed and power efficiency by leveraging local computation power.

Private: Keeps data on the device, enhancing security and protecting user privacy.

Personalized: Allows for continuous model customization without external data transfer or updates.

Device In-the-loop Deployment



Device in the loop

Qualcomm® AI Hub

On-Device Generative AI

Applications

- Live translation
- Live transcription
- Photo portrait generation
- Photo AI editing
- Semantic search
- Text summarization
- Virtual assistants
- Writing assistance
- Image generation

aotgan convnext_tiny ddrnet23_slim deeplabv3_resnet50 densenet121 detr_resnet101

detr_resnet101_dc5 detr_resnet50 detr_resnet50_dc5 **Whisper** esrgan facebook_denoiser

fcn_resnet50 **Baichuan** huggingface_wavlm_base_plus inception_v3 lama_dilated litehrnet

mediapipe_face mediapipe_hand googlenet Resnet18 efficientnet_b0 **Mediapipe_pose**

mnasnet05 mobiledet

100+ Models

mobilenet_v3_small resnet_mixed

openpose QuickSRNet_Large real_esrgan_general_x4v3

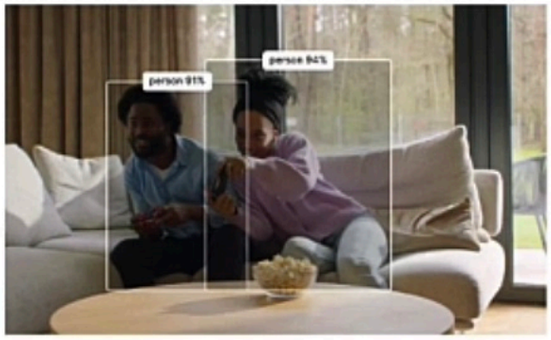
resnet_2plus1d resnet_3d

real_esrgan_x4plus **Stable Diffusion** **Control Net** ResNeXt50

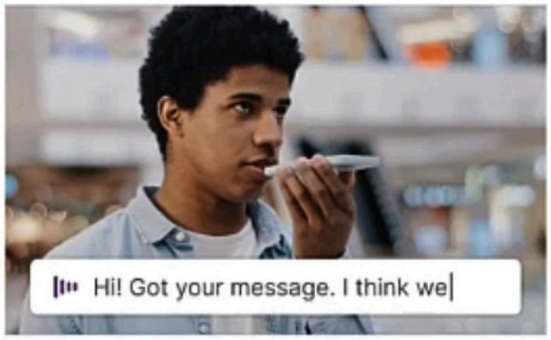
sam Sesr_m3 shufflenet_v2 sinet squeezenet1_1 stylegan2 trocr unet_segmentation

Vit whisper_asr wideresnet50 xlsr yolov6 yolov7 yolov8_det yolov8_seg

On-Device AI Applications



Object Detection



Speech Recognition

Image Segmentation

Pose Estimation

Super Resolution

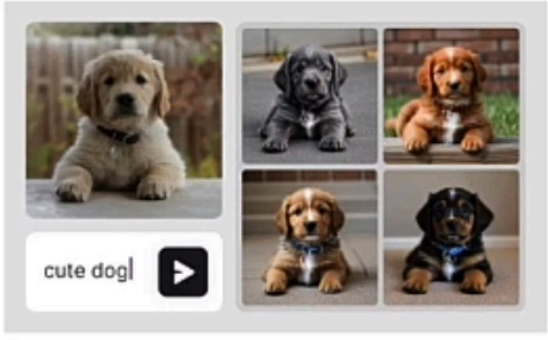


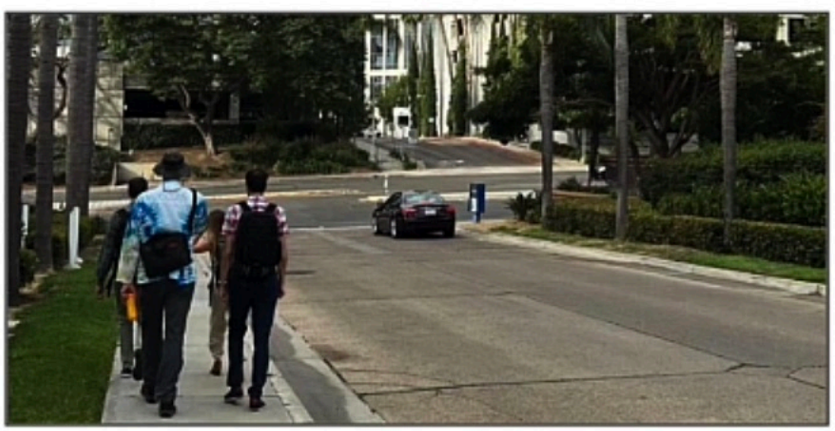
Image Generation

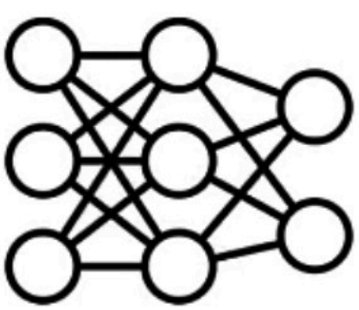
Explore more at https://aihub.qualcomm.com/

Qualcomm

What is Image Segmentation?

Image segmentation **divides an image** into **meaningful segments** for easier object identification and analysis.





Predicted Segments

Types of Image Segmentation

Semantic Segmentation: Assigns a label to every pixel in an image for classifying entire objects and shapes.

Instance Segmentation: Distinguishes individual objects within the same category by labeling each instance separately.

Semantic Segmentation

Segment image only by class (e.g. person, table)

Instance Segmentation

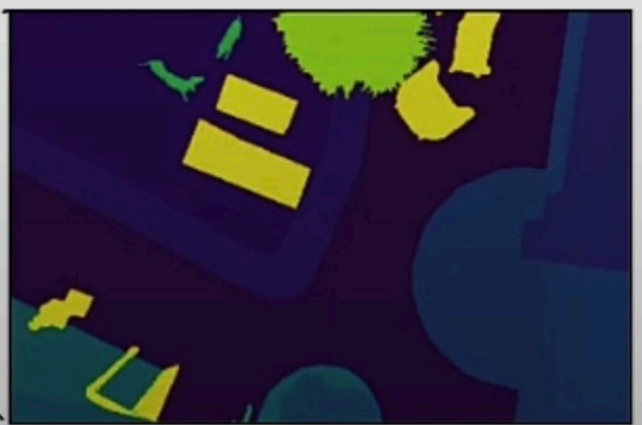
Each instance labelled separately (e.g. person 1, person 2)

Applications of Image Segmentation

Driving Assistance

Image Editing

Landscape Mapping



Real-Time Segmentation

Processing and segmenting **images instantly** in real-time for immediate analysis or action.

Apply Image Segmentation frame-by-frame

Semantic Segmentation Models/Algorithms

ResNet (Residual Network): Uses residual connections to enable training of very deep networks by allowing gradients to flow through layers without diminishing.

HRNet (High-Resolution Network): Maintains high-res representations through the network, enabling it to capture fine details for accurate segmentation.

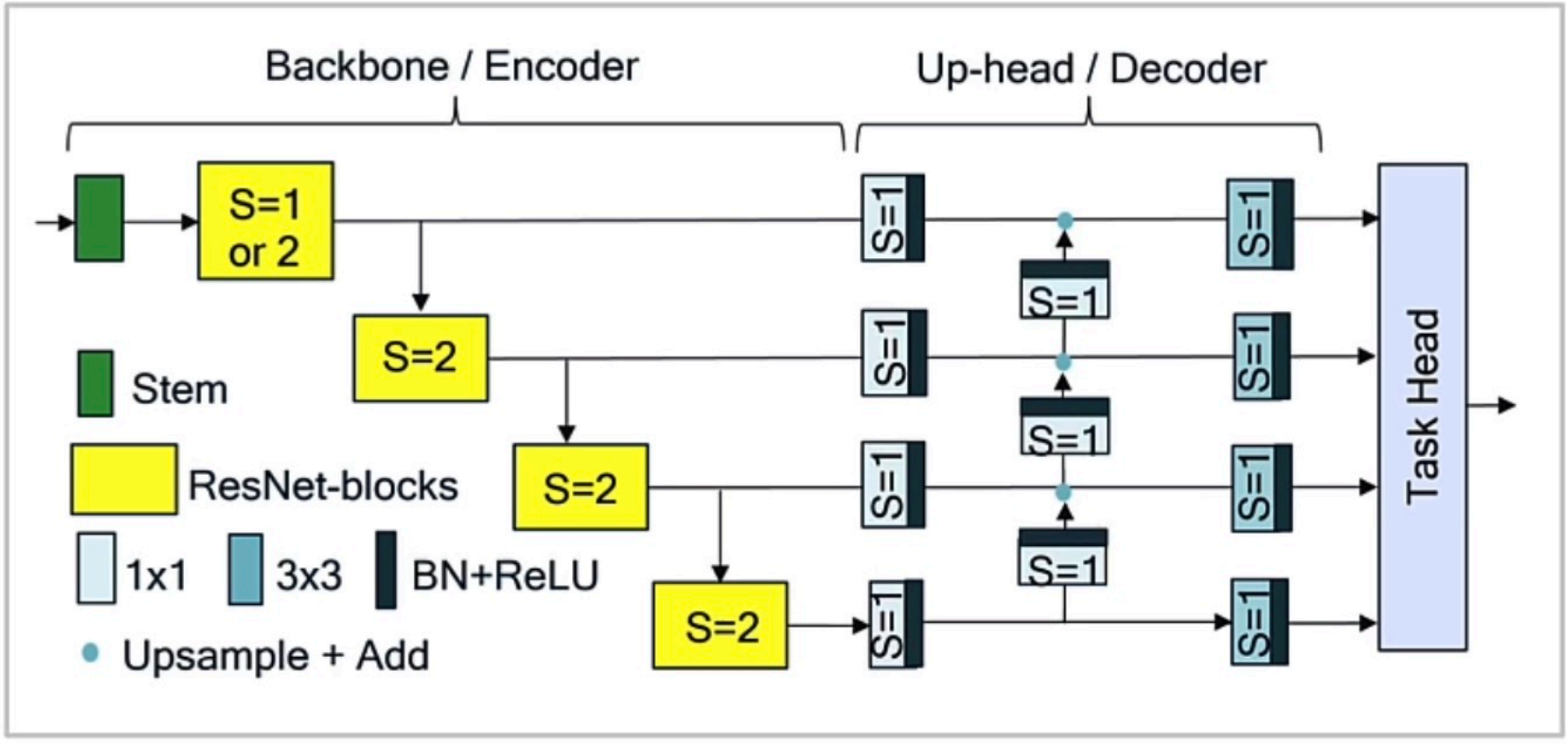
FANet (Feature Agglomeration Network): Focuses on agglomerating features from different scales, enhancing the model's ability to discern finer details.

DDRNet (Dual Dynamic Resolution Network): Employs dual-path architectures to balance efficiency and accuracy, facilitating real-time semantic segmentation.

Fuss Free Network (FFNET)

Fuss-Free Network (FFNet): A simple encoder-decoder architecture with a ResNet-like backbone and small multi-scale head.

Performance: Performs on-par or better than complex semantic segmentation architectures such as HRNet, FANet and DDRNets



Configurable Encoder + Decoder network

FFNET Variants

Variant	Backbone Encoder	Resolution	Model Size (MB)	params (M)	ops (GFlops)
FFNet 40s	Resnet 40S	1024 x 2048	55.6	13.9	62.3
FFNet 54s	Resnet 54s	1024 x 2048	72.2	18.0	75.7
FFNet 78s	Resnet 78S	1024 x 2048	109.9	27.4	96.0
FFNet 78s (Low Res)	Resnet 78S	512 x 1024	107.3	26.8	19.9
FFNet 122ns (Low Res)	Resnet 122N	512 x 1024	128.6	32.1	127.4

On-Device Deployment Key Concepts

"Capture"
Trained Model

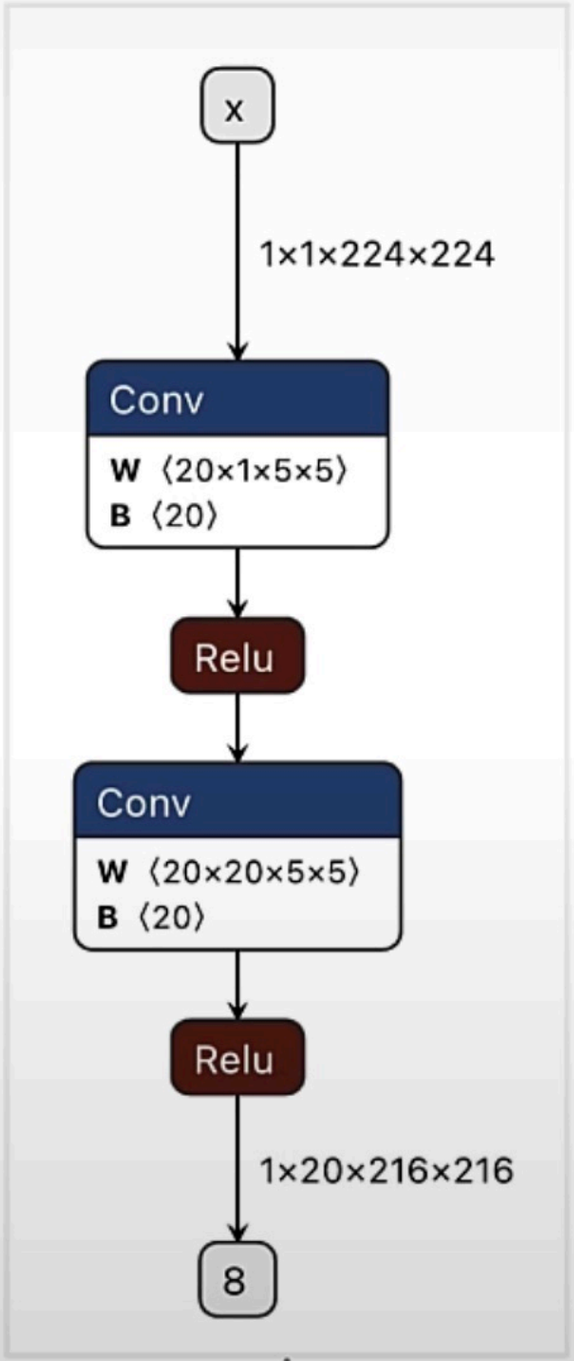
Performance profile
on-device

Compile Model
for target

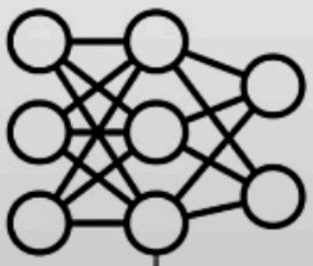
Validate
On-Target numerics

What is graph capture?

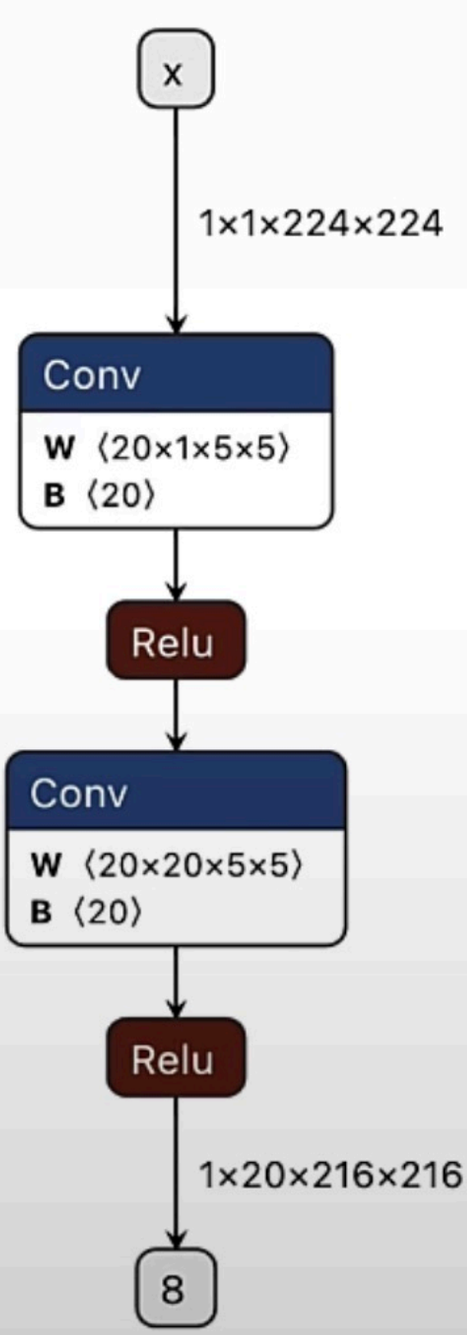
```python
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```



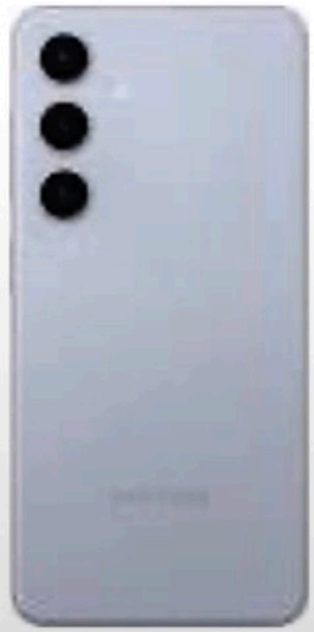
Graph captured for portability



What is compilation?



PyTorch graph captured



Model optimized for device

Target Runtime

Device

TensorFlow Lite

ONNX RUNTIME

TensorFlow Lite: Recommended for Android applications

ONNX Runtime: Recommended for Windows applications

Qualcomm AI Engine: Suitable for fully embedded applications

TensorFlow Lite

Optimized for Mobile: Specially designed for mobile and embedded devices, ensuring fast, efficient performance.

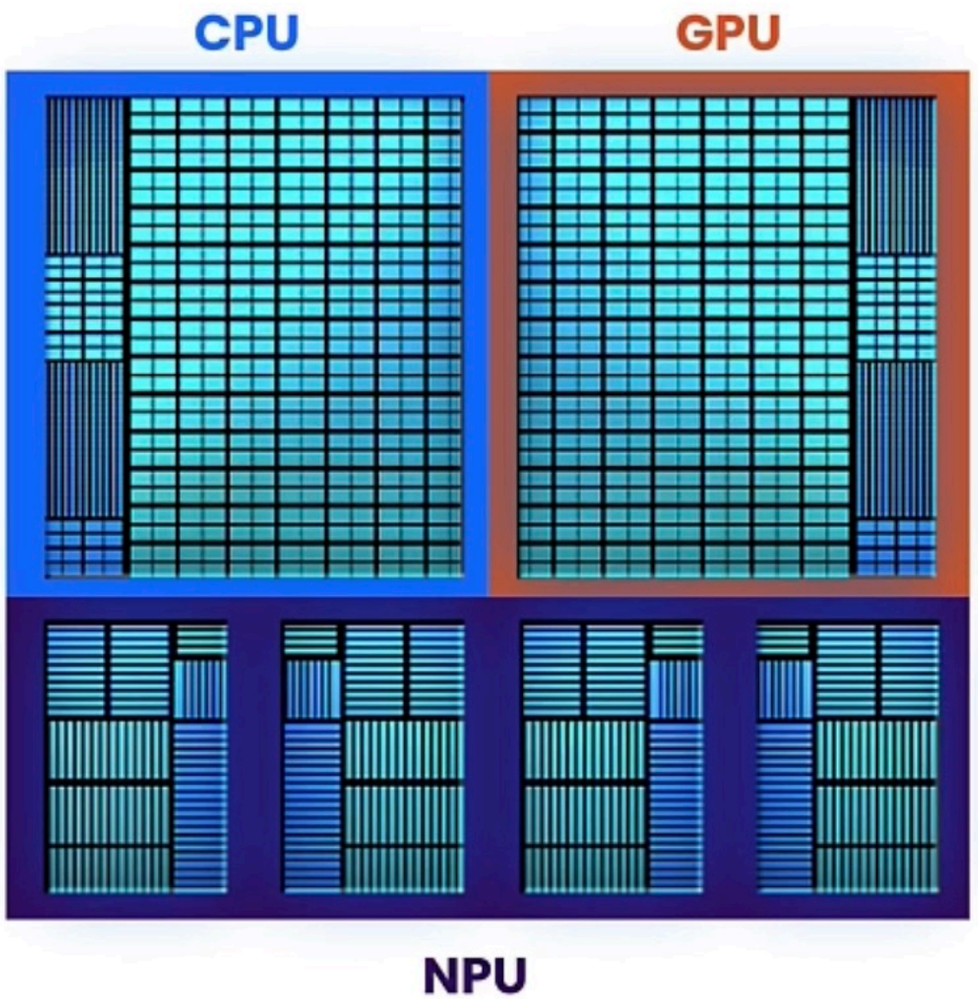
Low Latency: Provides fast response times by reducing the computational overhead of model inference.

Flexibility in Deployment: Supports a wide range of devices from smartphones to IoT gadgets, making AI ubiquitous.

Energy Efficient: Uses less power than traditional models, ideal for battery-operated devices.

Hardware Acceleration Accelerated on Qualcomm NPU with the Qualcomm AI Engine delegation

Compute Unit

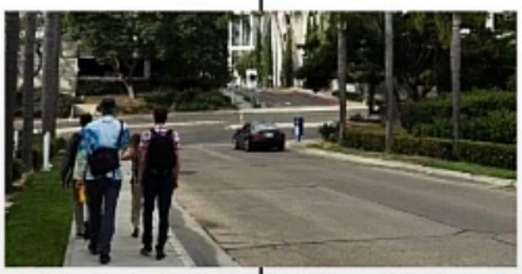


CPU: Most flexible general purpose computation engine

GPU: High-performance complex parallel computation

NPU: Extremely efficient compute block for neural networks

On-Device Accuracy



PyTorch

NPU

TensorFlow Lite

Step 1: Captured graph with PyTorch trace

Step 2: Compiled for device

Step 3: Profiled to run on NPU

Step 4: Validate on-device accuracy