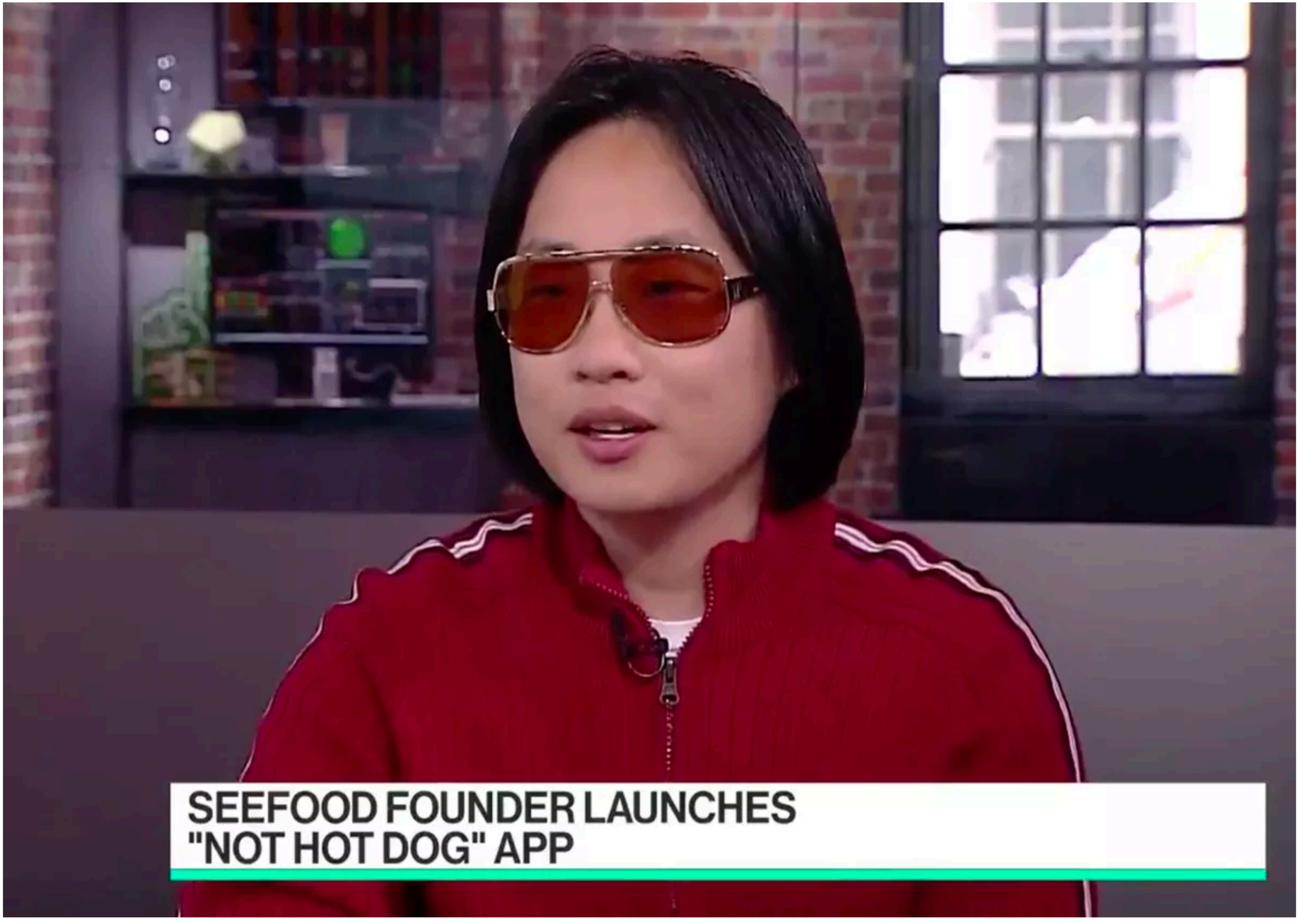
Machine Learning Inference on Mobile Phones

Anuj Dutt
Bose Corporation


SEEFOOD FOUNDER LAUNCHES
"NOT HOT DOG" APP

Changes in Mobile Phone SoC Market over Years

- **CPU:** Performs hundreds of thousands of calculations per second.

- **GPU:** Uses multi-threading for doing multiple things in parallel, more speedup for specialized operations.

- **AI Processors:** New SoC in the market. Specialized for running Neural Networks, faster at that than most GPU's.

The need for AI Processors

- Increased efficiency in three main areas:
  a) Size
  b) Computation
  c) Energy

- Examples: Apple's Bionic Neural Engine, Huawei HiSilicon Kirin 980 etc.

- Specialized SDK's: Google Neural Network API, Qualcomm Snapdragon Neural Processing Engine SDK etc.

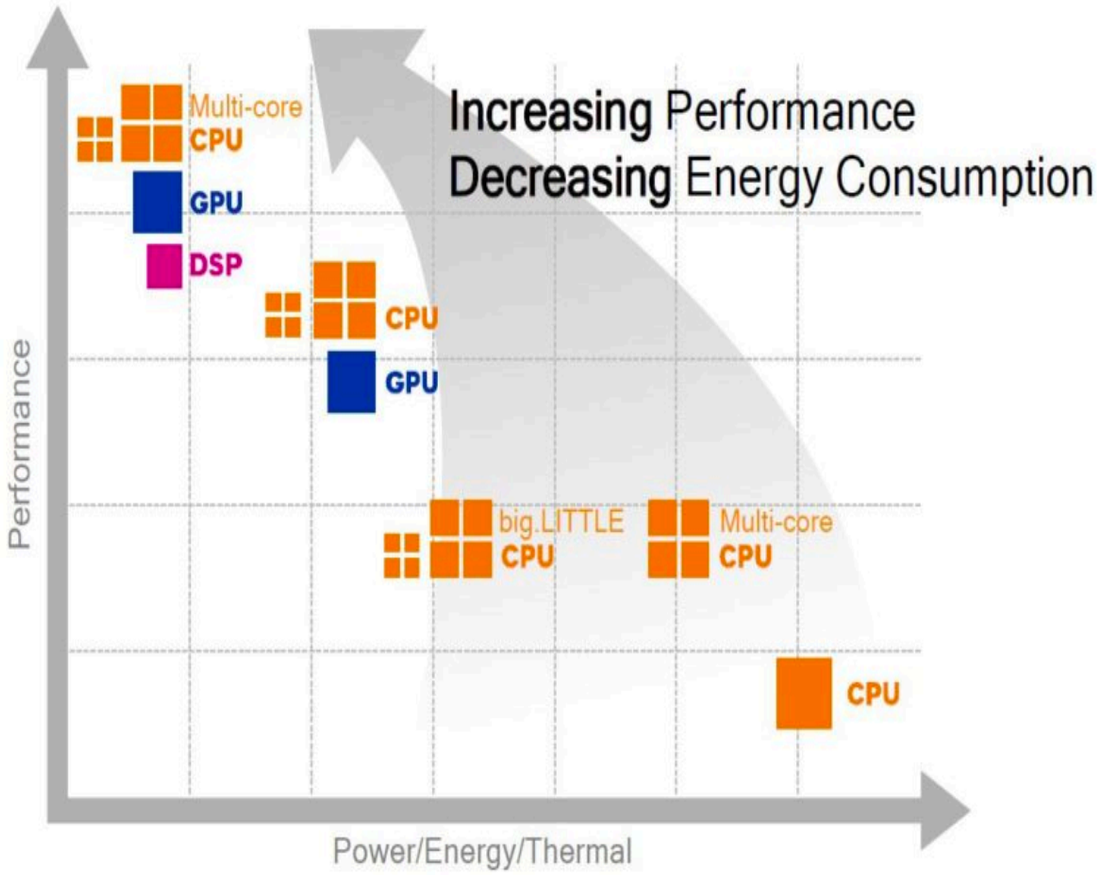


Image Source: https://www.androidauthority.com/smartphone-ai-processor-803019/

What does Machine Learning Inference on Mobile Phones Offer?

Models on Device

Privacy

Speed

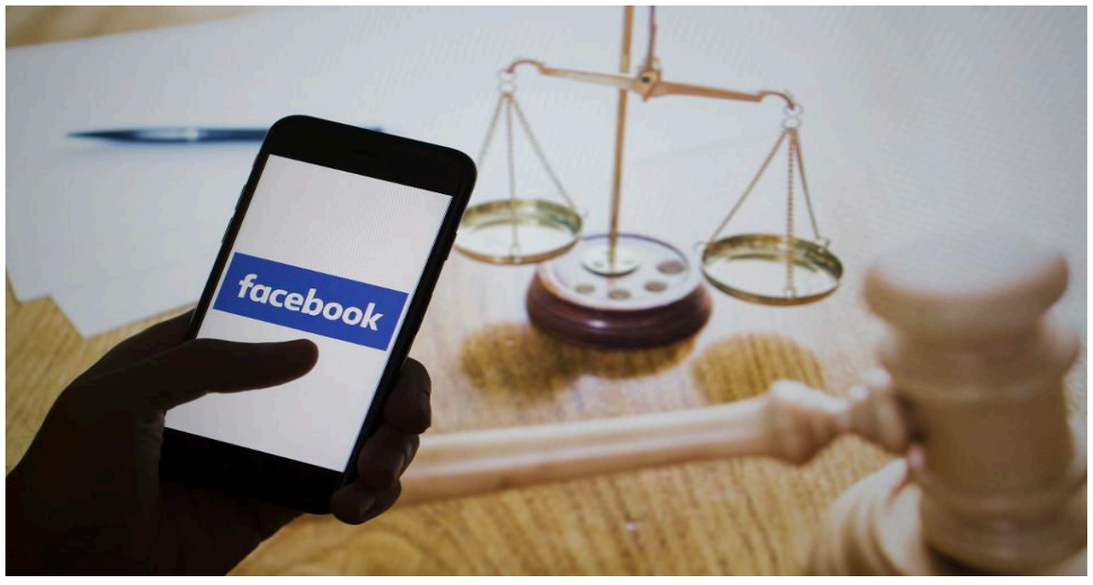
No Server

Available

Need to store machine learning models on device

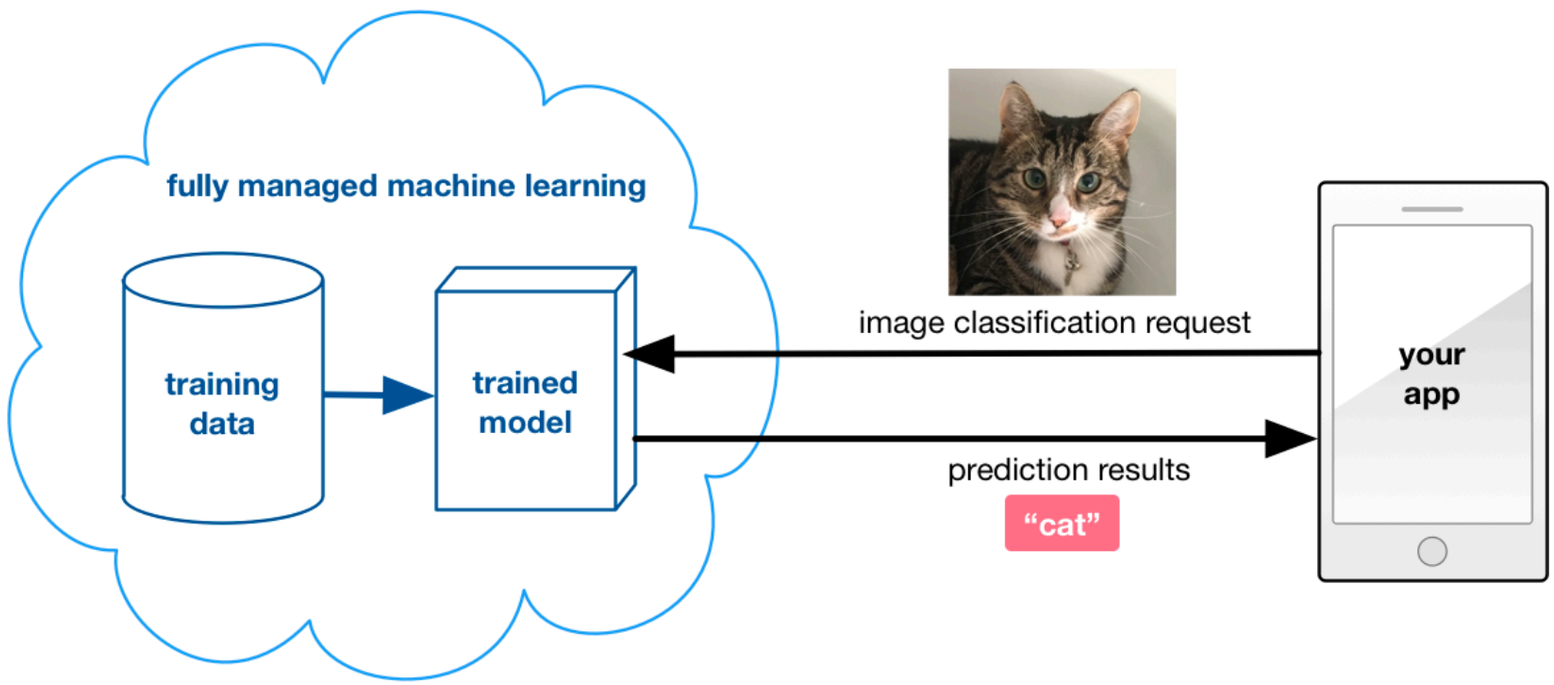
User Data Privacy

- A hard job and a big onus lies upon the companies collecting the data.
- Many instances available when privacy violations involve people who have been granted access to data.
- Regulators across the world are approaching the data privacy concerns in different ways like the introduction of GDPR in EU, but is there something more that we can do?
- Solution: Do as much processing as possible on the phone for ML applications.
- This way user's data never has to leave their mobile phone.



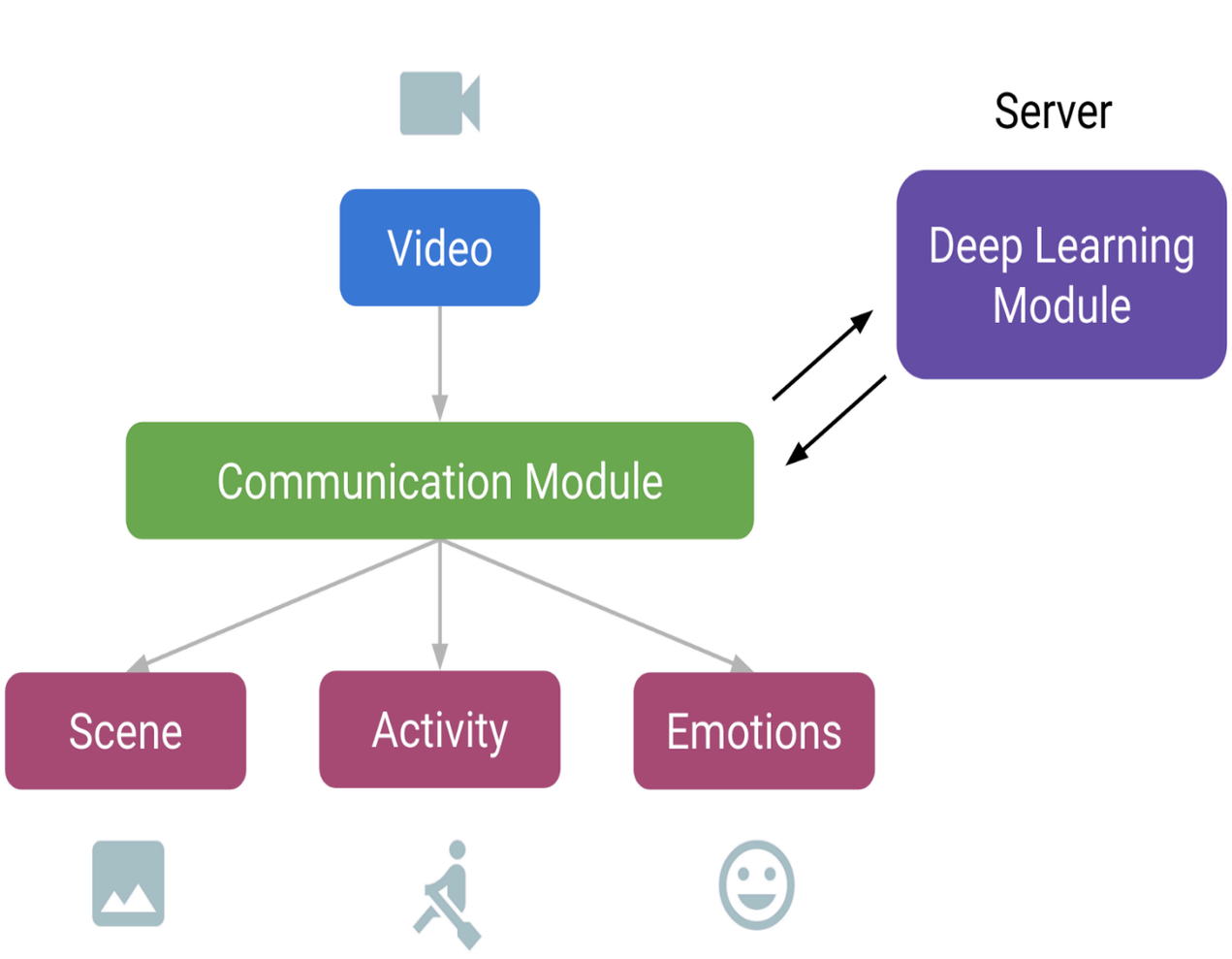
No Extra Data Cost

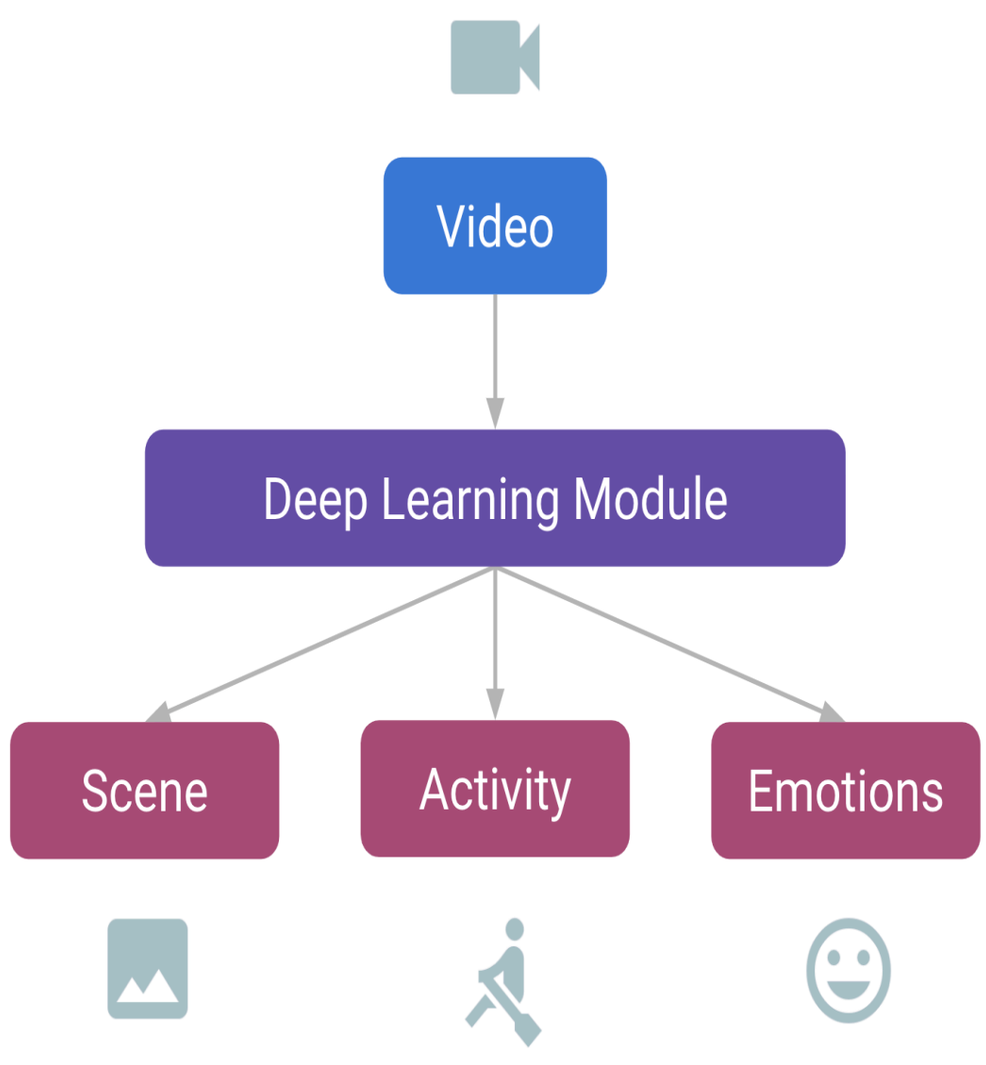
- With every app solution on a mobile device there is an internet data cost associated with it.
- **First Data Cost:** To download the application, specially the big ones.
- **Second Data Cost:** App to Cloud API communication in the cloud.
- **Third Data Cost:** Continuous streaming of data or uploading bulk data to cloud for processing or storage.



No Server Maintenance Cost

- Machine learning model runs locally within the application on the mobile device, hence:
  1. No server maintenance cost.
  2. No compute cost [EC2 etc.]
  3. No storage cost [S3]



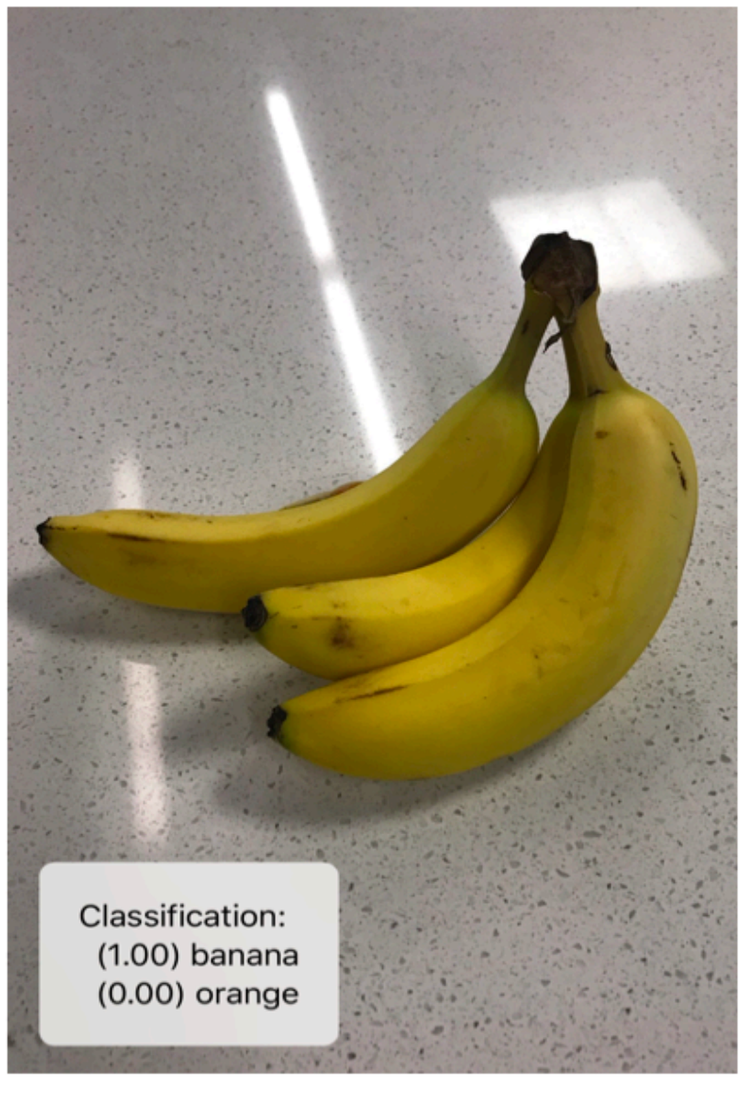


Always Available Applications

- Doing all the processing on the mobile phone means that your application will work always irrespective of whether you have an internet connection or not.
- Hence, the users never have to see any downtime or app unavailability.

Real Time Inference

- Running the model locally on-device means:
  1. No Internet Latency.
  2. No Bad Connection = Bad App Experience
  3. No Application Downtime.



Mobile Machine Learning Recipe

ML Application = Efficient Pre-Trained Models + Efficient Mobile Inference Engine

Tools for Deploying ML Solutions on the Phone

- Apple’s CoreML/CoreML 2 [iOS]
- Google’s TFLite [Android/iOS]
- Facebook’s Caffe2Go [Android/iOS]

Apple’s ML Ecosystem

- **Metal:** low level and low overhead hardware accelerated 3D graphic and computer shader API.

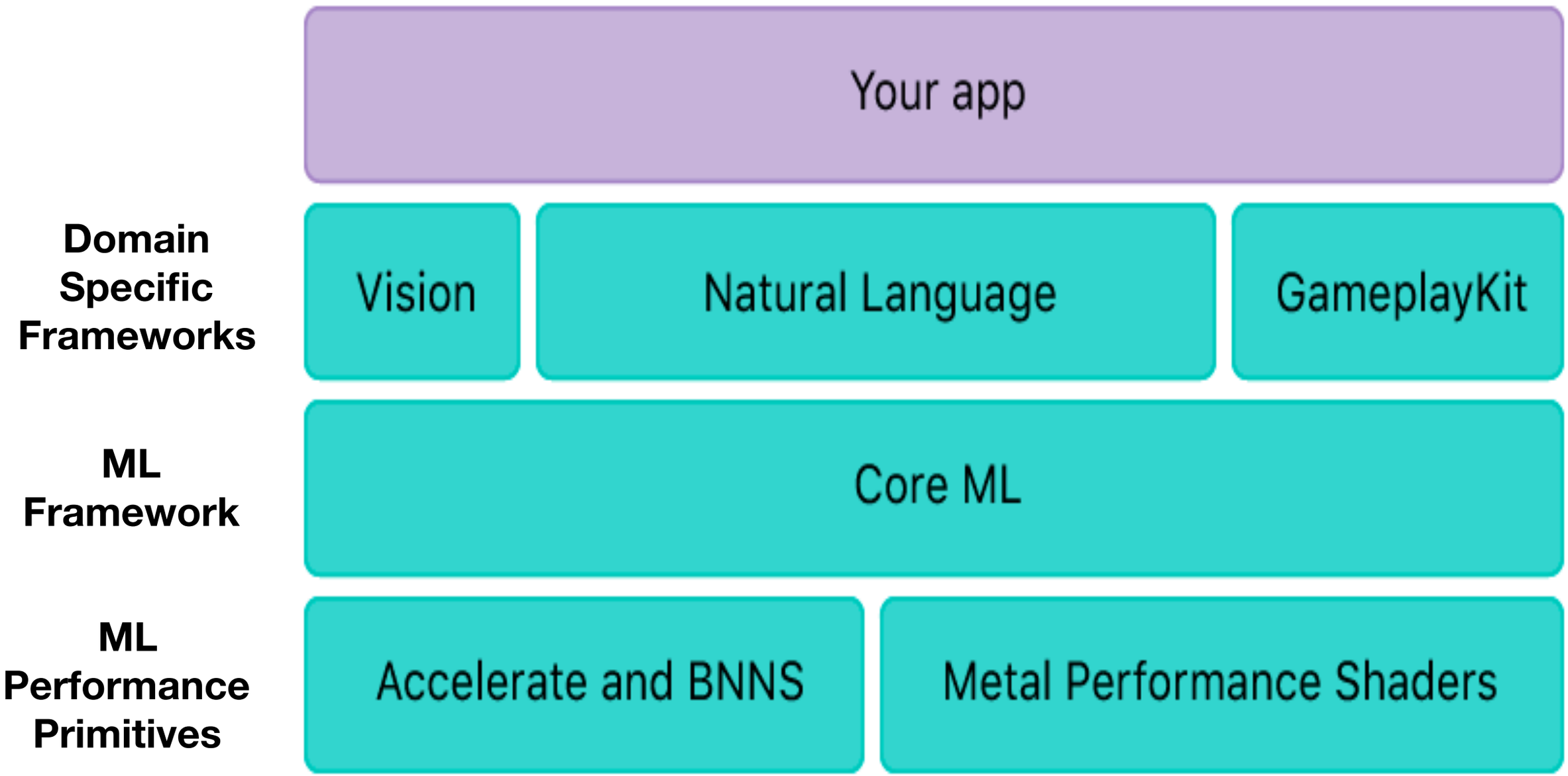
- **BNNS:** Basic Neural Network Subroutine ideal for Fully Connected Neural Networks

- **MPS:** Metal Performance shaders, ideal for Convolution operations.

- **CoreML:** Convert Pre-Trained or custom machine learning models in frameworks like TensorFlow, Keras, Scikit-Learn, PyTorch etc. to a format suitable for running in iOS applications.

- **CoreML 2:** Convert and optimize models on the go for on-device application integration.

CoreML Architecture



CoreML in Depth

- CoreML supports Feed Forward NN, CNN's, RNN's, Tree Ensembles, SVM and Generalized linear Models.

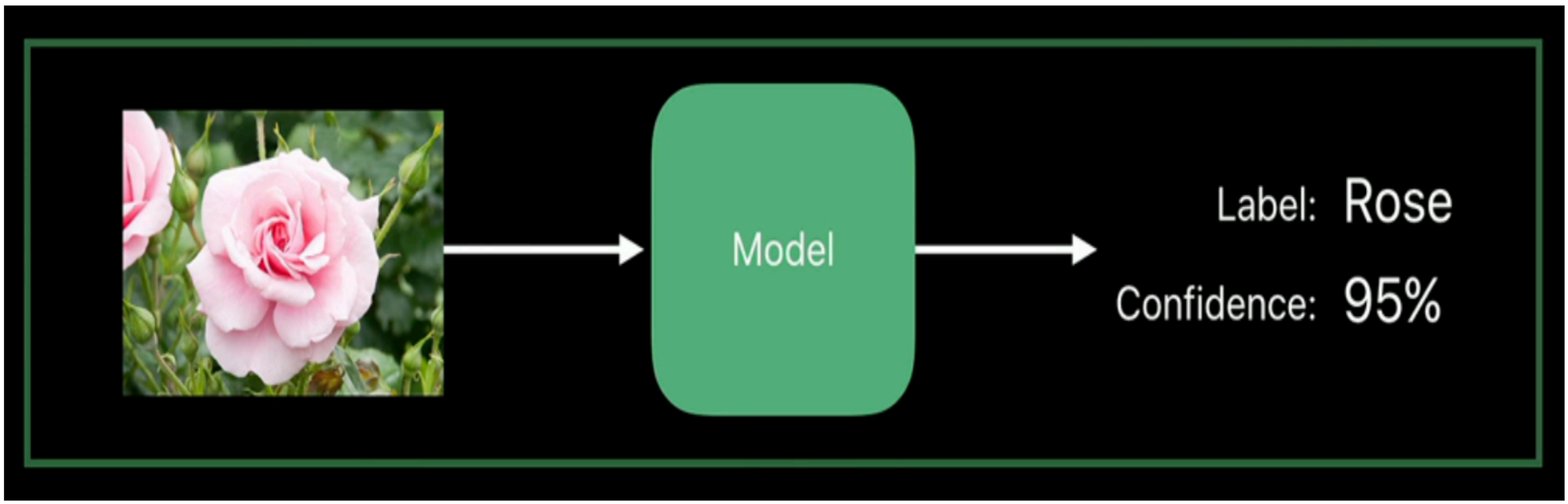
- The CoreML model contains a high level description of the machine learning model like the model weights, Input/Output types and structure of the network.

- A CoreML model supports following input types:
  1. Numerics, Categorical [Double, Int etc.]
  2. Images as CVPixelBuffer
  3. Arrays as MLMultiArray
  4. Dictionaries

- CoreML automatically takes care of hardware optimizations:
  A. Compute heavy tasks are scheduled to run on the GPU or Neural Processing Engine.
  B. Memory heavy tasks are scheduled to be run on Accelerate.
  C. Auto-context switching from CPU to GPU and vice-versa for parts of the model.

CoreML Vision Framework

- CoreML's Vision framework is a specialized framework for applying computer vision algorithms to a variety of tasks involving image and video data.
- Examples: Face recognition, face landmark detection, text detection, barcode recognition, image registration, and general feature tracking.
- Vision also allows the use of custom Core ML models for tasks like classification or object detection.



Using API's from Vision Framework

```swift
// Function to process image using CoreML
func processImage(image: UIImage) {
    // Create a Vision Model
    guard let visionModel = try? VNCoreMLModel(for: self.model.model) else {
        fatalError("Unable to create Vision Model !!")
    }

    // Create Vision Request
    let visionRequest = VNCoreMLRequest(model: visionModel) { request, error in
        if (error != nil){
            return
        }

        guard let results = request.results as? [VNClassificationObservation] else {
            return
        }

        // Get Predictions from CoreML Model
        let predictions = results.map { observation in
            "\(observation.identifier): \(observation.confidence * 100)"
        }

        // Show Predictions in Text Box
        DispatchQueue.main.async {
            self.imageLabel.text = predictions.joined(separator: "\n")
        }
    }

    // Vision Request Handler
    let visionRequestHandler = VNImageRequestHandler(ciImage: CIImage(image: image)!, orientation: .up, options: [:])

    // Run the Vision Request on Main Thread
    DispatchQueue.global(qos: .userInteractive).async {
        try! visionRequestHandler.perform([visionRequest])
    }
}
```

CoreML Natural Language Framework

Language Recognition

```
import NaturalLanguage

let recognizer = NLLanguageRecognizer()

recognizer.processString("困死啦睡觉去了")

let lang = recognizer.dominantLanguage

let hypotheses  = recognizer.languageHypotheses(withMaximum:2)
```

Word Tagging

```
import NaturalLanguage

let tagger = NLTagger(tagSchemes: [.nameType])

let str = "Prince Harry and Meghan Markle had their wedding ceremony in Windsor"

let strRange =  str.startIndex ..< str.endIndex

tagger.string = str

tagger.setLanguage(.english, range: strRange)

let tags = tagger.tags(in: strRange, unit: .word,

            scheme: .nameType, options: .omitWhitespace)
```

CoreML Features

- **Weight Quantization:** Reduced model and application size.

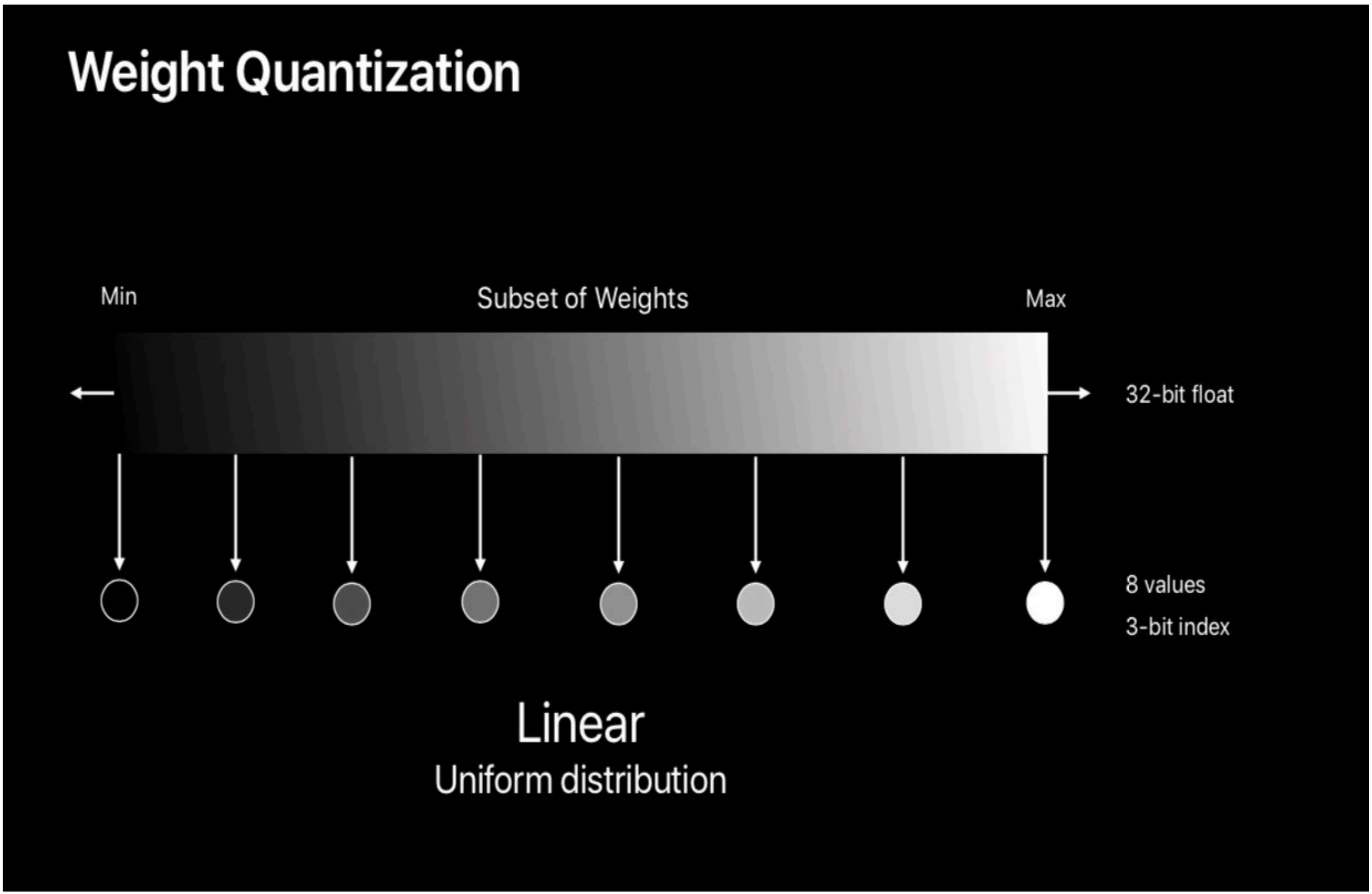
- iOS 12 supports 8-bit, 4-bit unto 1-bit quantizations which reduces the model size drastically.

- **Note:** Quantization might have an effect on your model accuracy. So, test for model accuracy after quantization and then decide if that decrease is acceptable for you or not, depending on your applications.

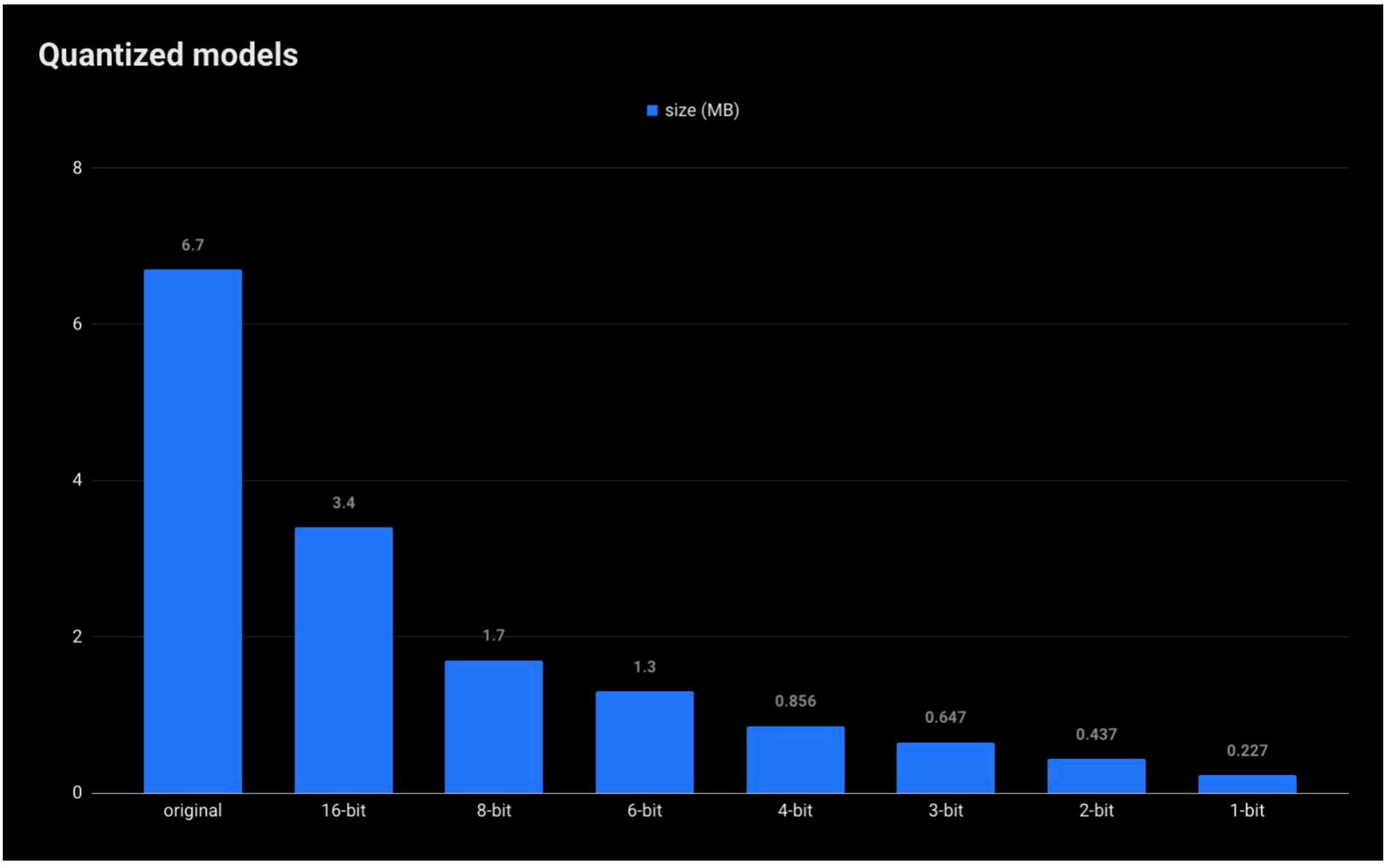
- **Batch Processing:** Efficient processing of data in batches rather than looping through them.

- **Flexible Shapes:** Pass in range of image or frame input size rather than defining a constant value.

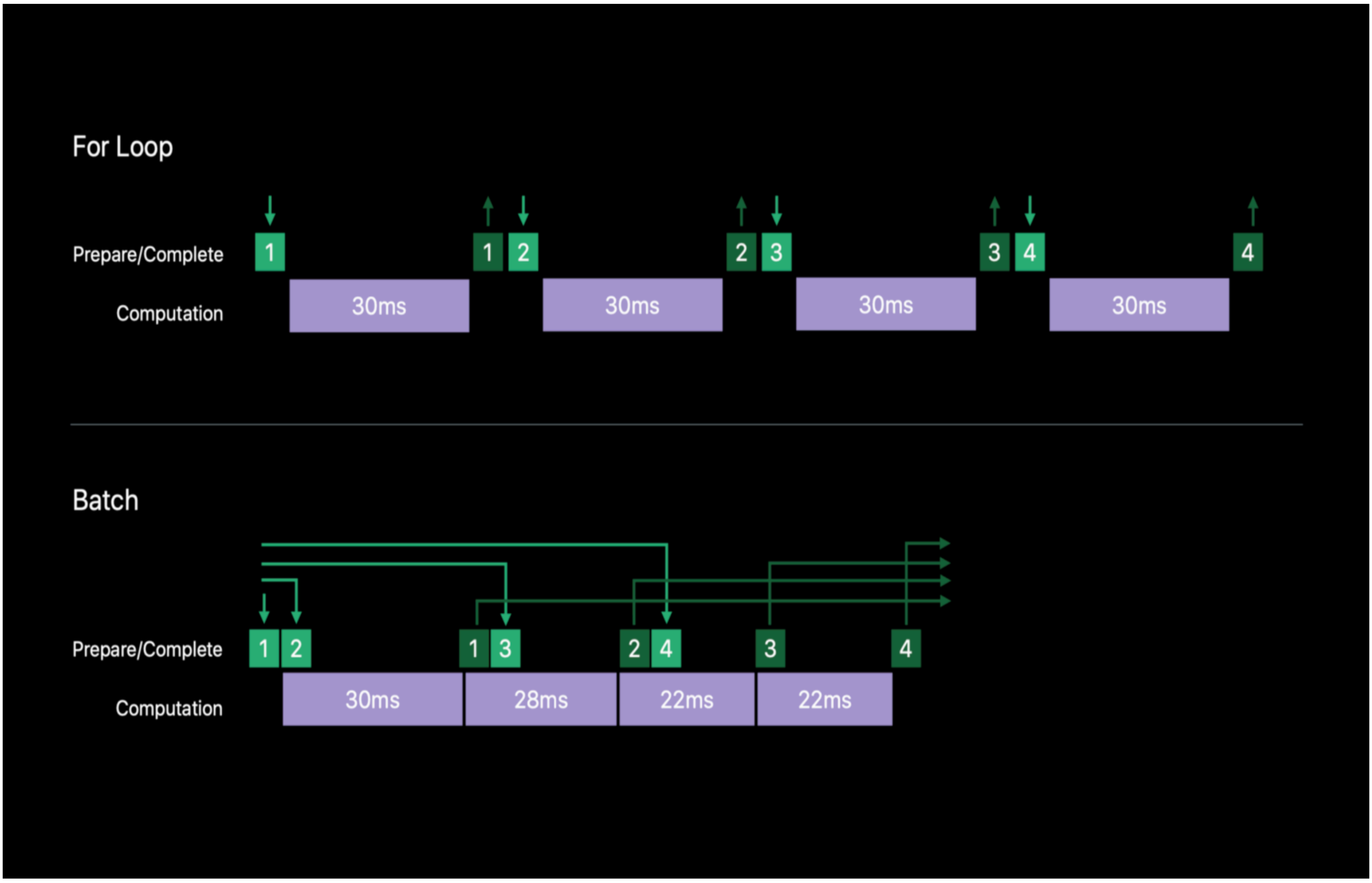
CoreML 2: Weight Quantization



CoreML 2: Reduced Model Size



CoreML 2: Batch Processing



CoreML 2: Flexible Shapes

Machine Learning Model

Name	myModel2
Type	Neural Network
Size	441 KB
Author	Jeshua Lacock
Description	Feedforward style transfer https://3DTOPO.com
License	(C) 3DTOPO Inc. All Rights Reserved.

Model Class

myModel2

Automatically generated Swift model class

Model Evaluation Parameters

Name	Type	Flexibility	Description
Inputs			
inputImage	Image (Color 1536 x 1536)	640...2048 x 640...2048	Image to stylize
Outputs			
outputImage	Image (Color 1536 x 1536)	640...2048 x 640...2048	Stylized image

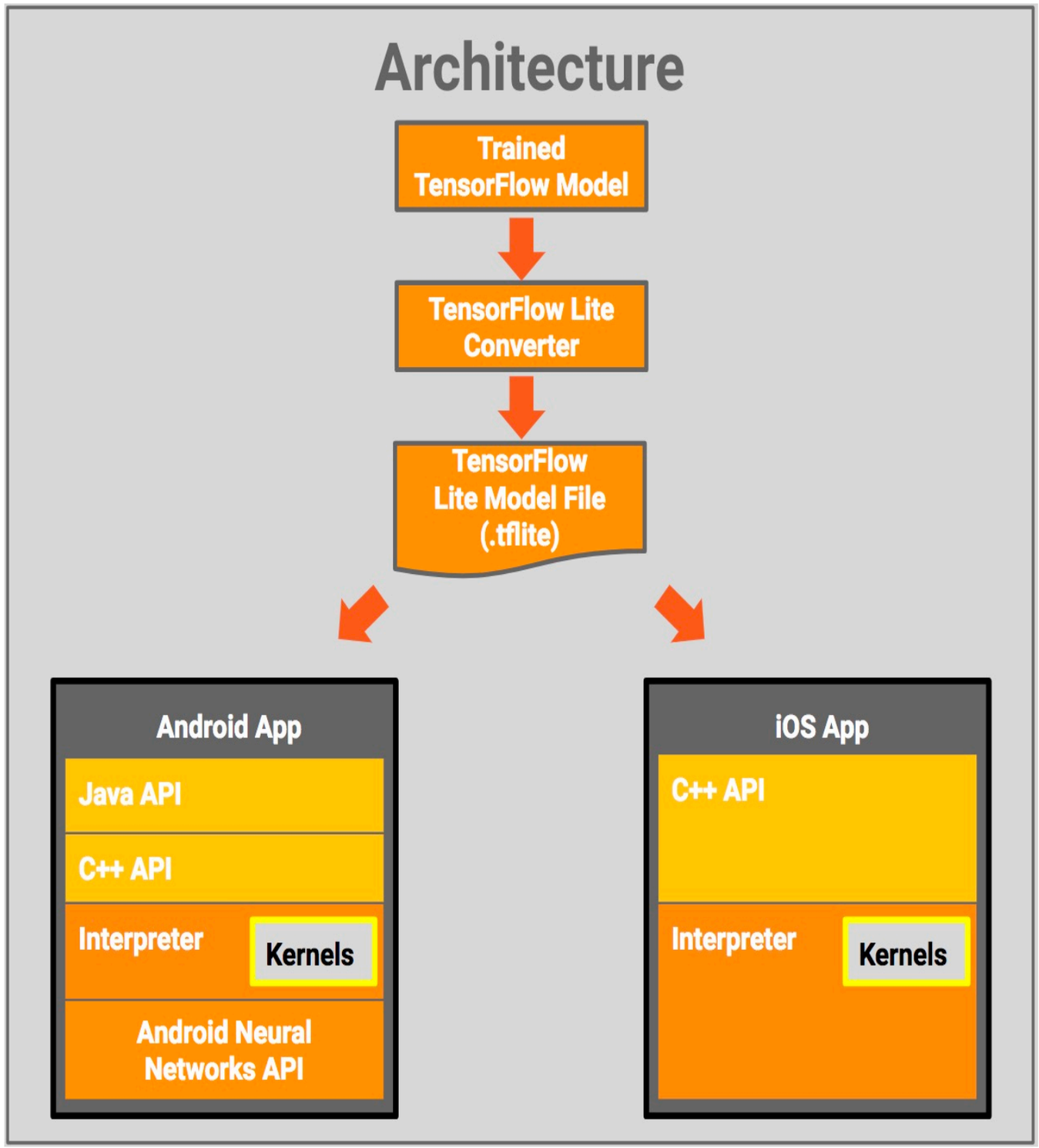
TFLite

- **Model's Protobuf + Checkpoint to TFLite:** Use existing TensorFlow API's (toco scripts) to convert saved model files i.e. protobuf and checkpoint files to a TFLite model.

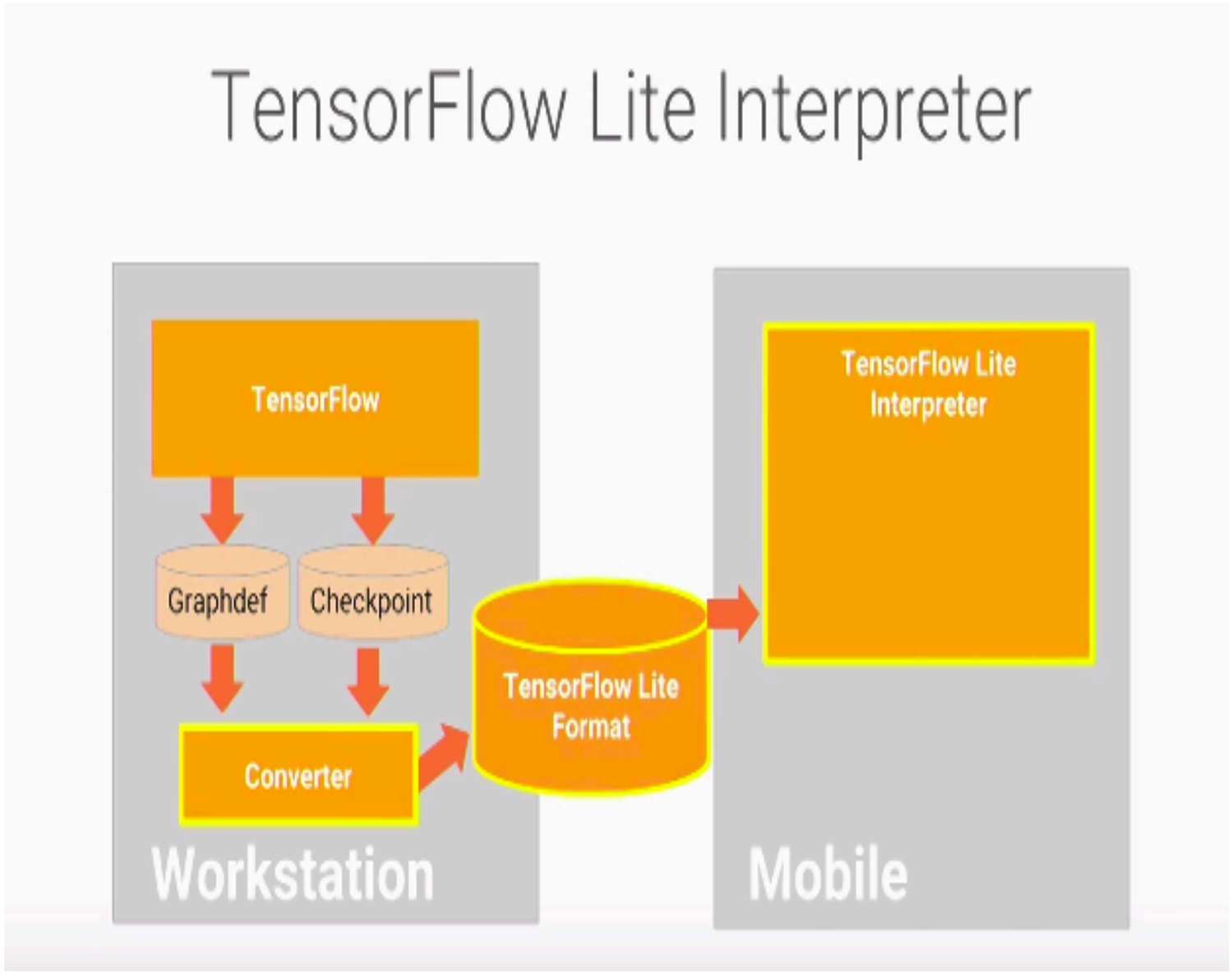
- **TFLite** is a model file that contains the structure of the model as well as the weights and biases.

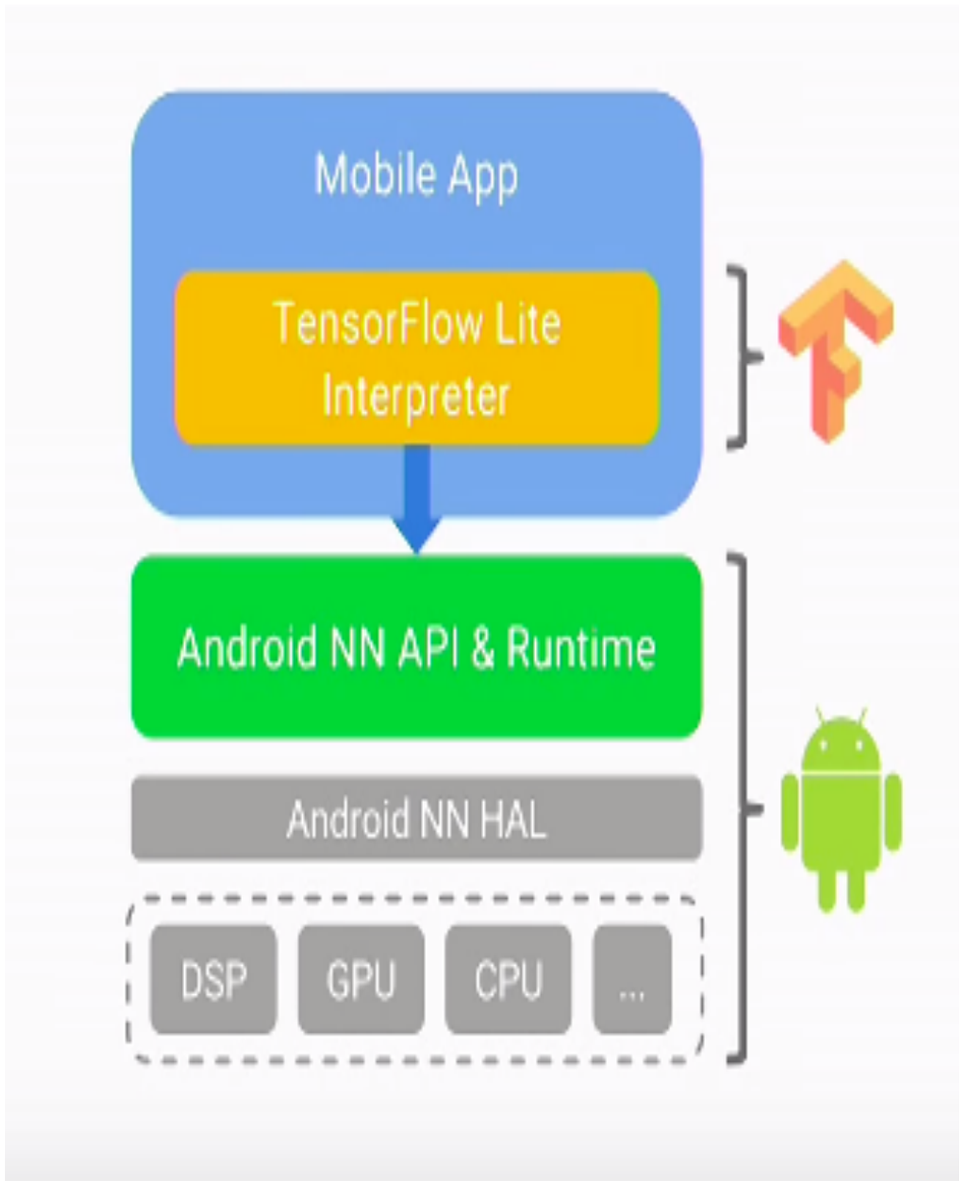
- To run a TFLite model on Android, just deploy the model in an app and it'll make use of Android's ANN API to optimize the performance of the model.

- For iOS, TFLite models use MPS to improve the performance of the model inference.



Saved ML Model to TFLite Format





Using TFLite API

Freeze Graph using saved model [Protobuf + Checkpoint] :

```
freeze_graph --input_graph=/tmp/mobilenet_v1_224.pb \
  --input_checkpoint=/tmp/checkpoints/mobilenet-10202.ckpt \
  --input_binary=true \
  --output_graph=/tmp/frozen_mobilenet_v1_224.pb \
  --output_node_names=MobileNetV1/Predictions/Reshape_1
```

Convert Frozen Graph to TFLite Model :

```
tflite_convert \
  --output_file=/tmp/mobilenet_v1_1.0_224.tflite \
  --graph_def_file=/tmp/mobilenet_v1_0.50_128/frozen_graph.pb \
  --input_arrays=input \
  --output_arrays=MobilenetV1/Predictions/Reshape_1
```

TFLite Features

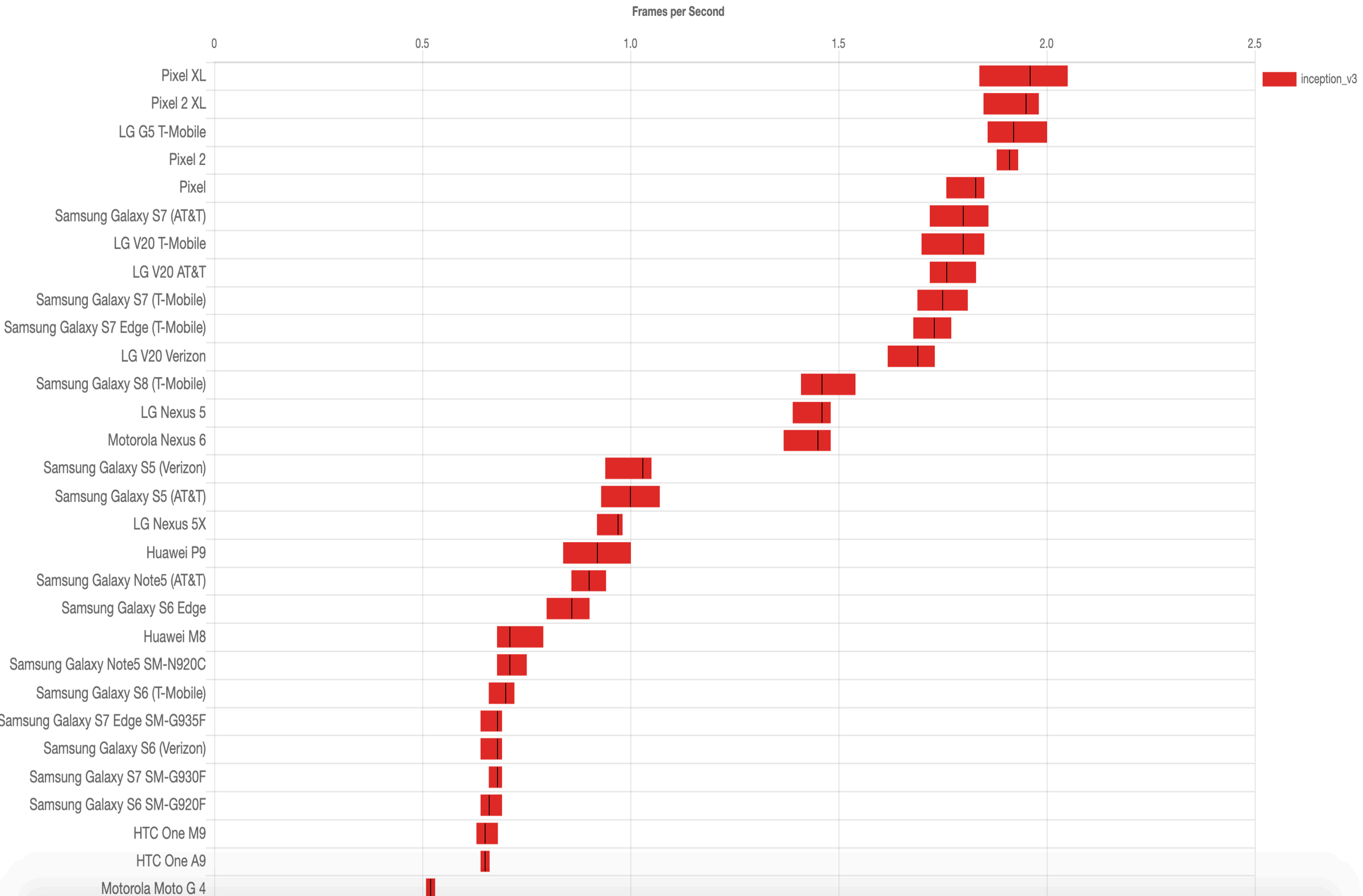
- **TFLite Models:** Smaller, faster and require minimal dependencies to run on an Android or an iOS device.

- **Low Memory Constraint:** Requires around 75 KB of memory for the core interpreter, around 400 KB when combined with the supported operations.

- TFLite takes advantage of hardware acceleration for faster inference. It makes use of Flat buffers that reduces the code footprint, memory usage, CPU cycles required for serialization and deserialization thereby improving the app startup time.

- TFLite's Interpreter uses static memory and static execution plan thereby reducing the load time.

TFLite Benchmark for Inception V3



Caffe2

- Caffe2: Deep Learning framework built by Facebook.

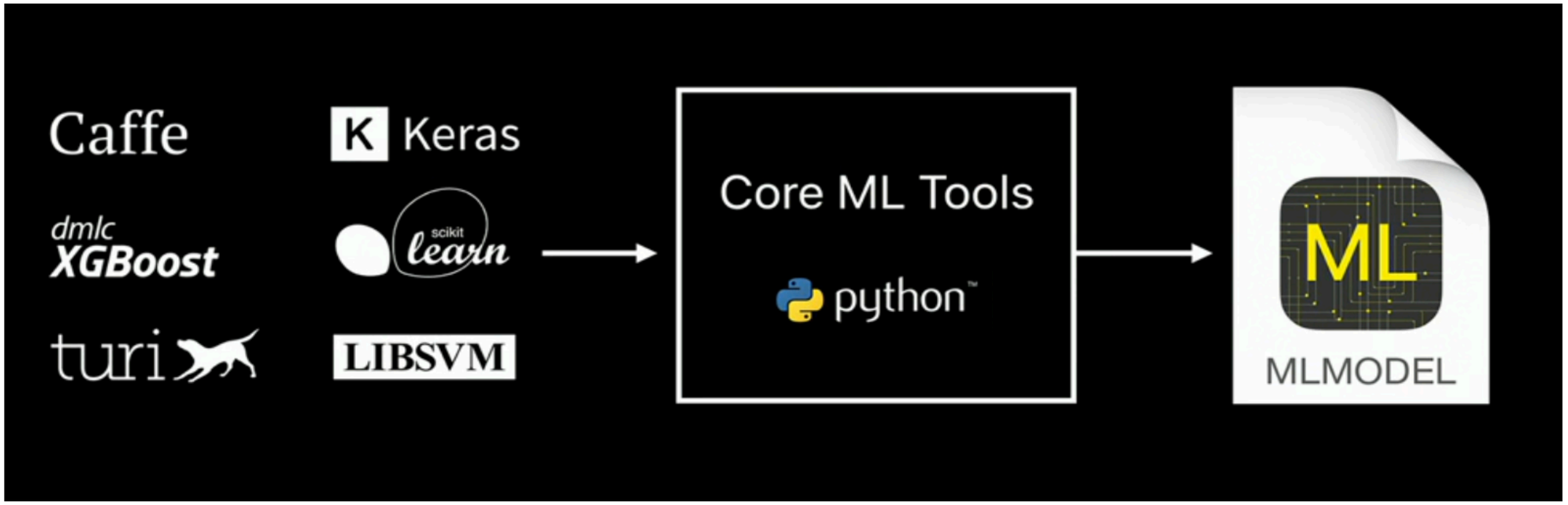
- This framework is built for speed as it has built-in optimizations for different hardware architectures like:

    1. For ARM based CPU, it makes use of NEON Kernels and NNpack which are acceleration packages for running neural networks on multi-core CPU.
    2. For iOS devices, it makes use of MPSGraph and Metal libraries to speed up performance on the GPU.
    3. For Android devices, it makes use of Qualcomm Snapdragon Neural Processing SDK that manages efficient running of machine learning models on CPU, GPU and DSP.

- It also supports the models in ONNX format as that allows interoperability between Caffe2 and PyTorch as well as other frameworks like TensorFlow, Keras etc.

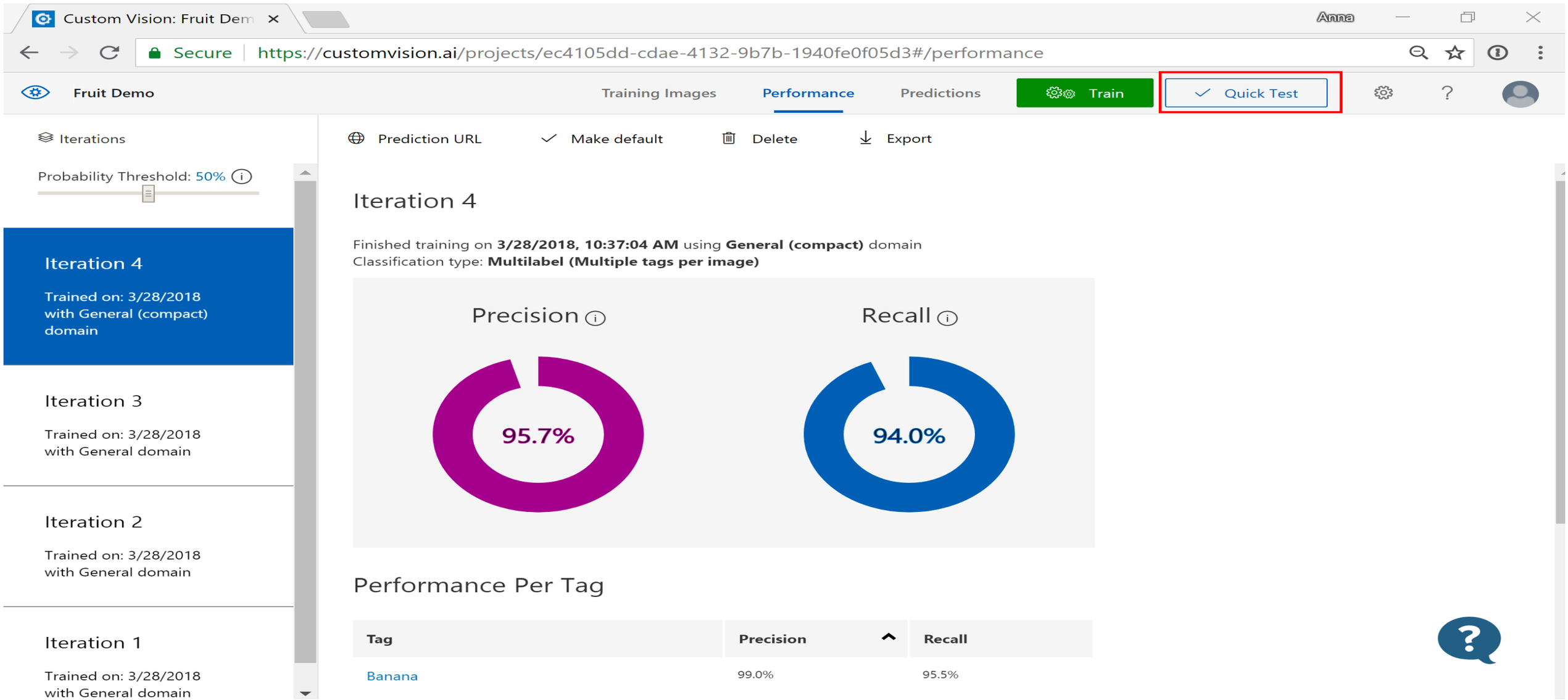
Tools to deploy ML Models on iOS

- **Microsoft Custom Vision API**
- **Turi Create**
- **CreateML**
- **Coremltools**
- **TF-CoreML**
- **ONNX-CoreML**
- **ML Kit**
- **MMDNN**



Tools to deploy a ML Model on Android

- **Microsoft Custom Vision API**
- **TensorFlow TFLite Converter**
- **ML Kit**



Applications of ML on Mobile Phones

- Apple's Face ID for unlocking the phone, authentication etc.
- Gmail and LinkedIn's Smart Reply feature using LSTM on device.
- Face Landmark detection for Facebook's on-device Augmented Reality.
- Google and Apple keyboard's predictive text for messages, emails etc.
- Activity Classification on Mobile Phones and Smart Watches.
- Shazam for music discovery.
- Image Segmentation with point clouds for Augmented Reality. ex. 6D.ai

Hands-On

Questions ??