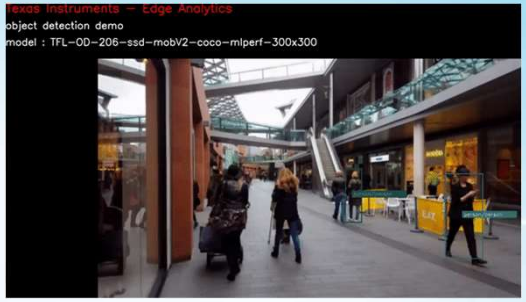
CH

Enabling High-performance Embedded Edge AI

Build your Embedded Edge AI Hello WORLD application today

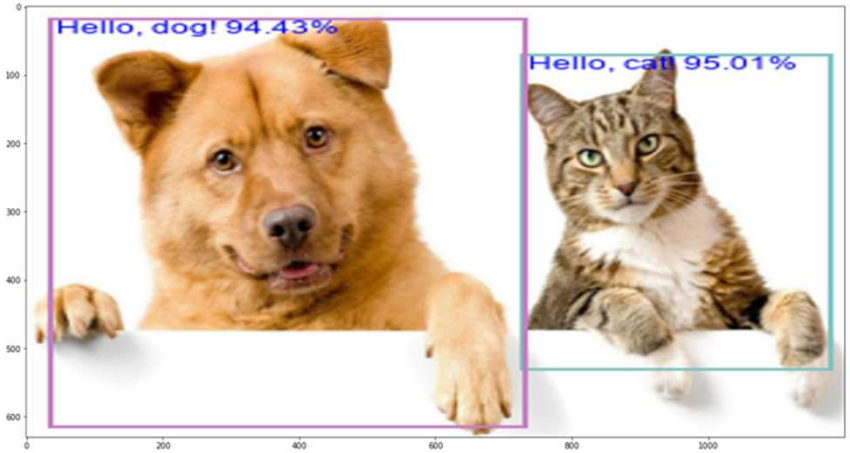


Jacinto™ AI monthly webinar series

Jun 2021

Hello world | Evolution to Hello AI

Traditional “hello world” (1978)



With AI: Determine who to say “hello”

- ✓ **Need to “read” and “preprocess” the image**
- ✓ **“Determine” multiple objects in the image**
- ✓ **Output the results.**
- ✓ **That’s a lot of complexity**
- ✓ **And, we need to “embed” that into an edge device**

We are going to make this as easy as possible

Source: https://en.wikipedia.org/wiki/%22Hello,_World!%22_program#/media/File:Hello_World_Brian_Kernighan_1978.jpg

CH

Webinar | Agenda

- Introduction to Jacinto Edge AI portfolio
- Embedded AI application development process
  - Model development, porting and optimization
- HelloWorld example (Three steps)
  - Three steps: PC, Embedded on ARM, Embedded with Deep learning acceleration
  - Demo
- Advanced development
- Call to action

Embedded Edge AI Technology | Unlimited possibilities

Smart Cameras, Autonomous Machines & Robots

Factory Automation

Retail Automation

Smart building & cities

Industrial transport

Healthcare

Agriculture

Construction

Aerospace & Defense

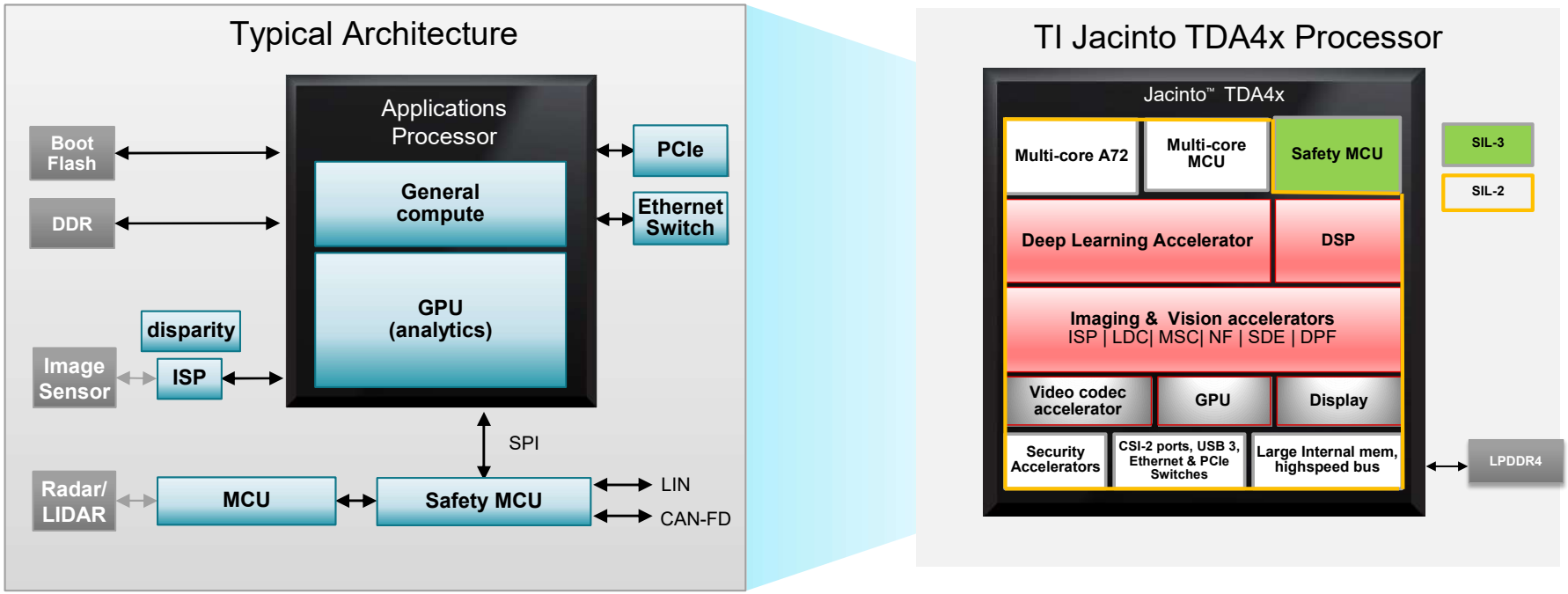
Logistics

Delivery

AI is influencing broad applications
New use cases in existing applications

CH

Enable intelligence | with purpose-built solutions



AI in low power, AI with low complexity
Simple Linux based programming using popular software frameworks

CH

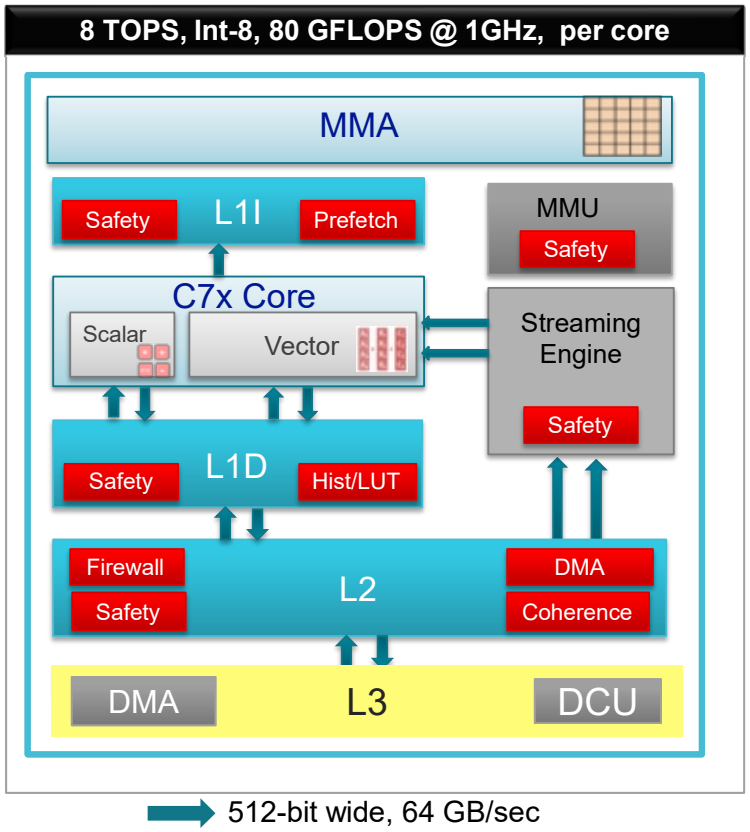
C7x + MMA | Industry's most efficient Deep Learning Accelerator

High FPS/TOPS
Designed for Lower power
Enables Fan-less design

Lowest #of DDR interfaces & bandwidth
SUPER-TILING
Using TI Proprietary Technology

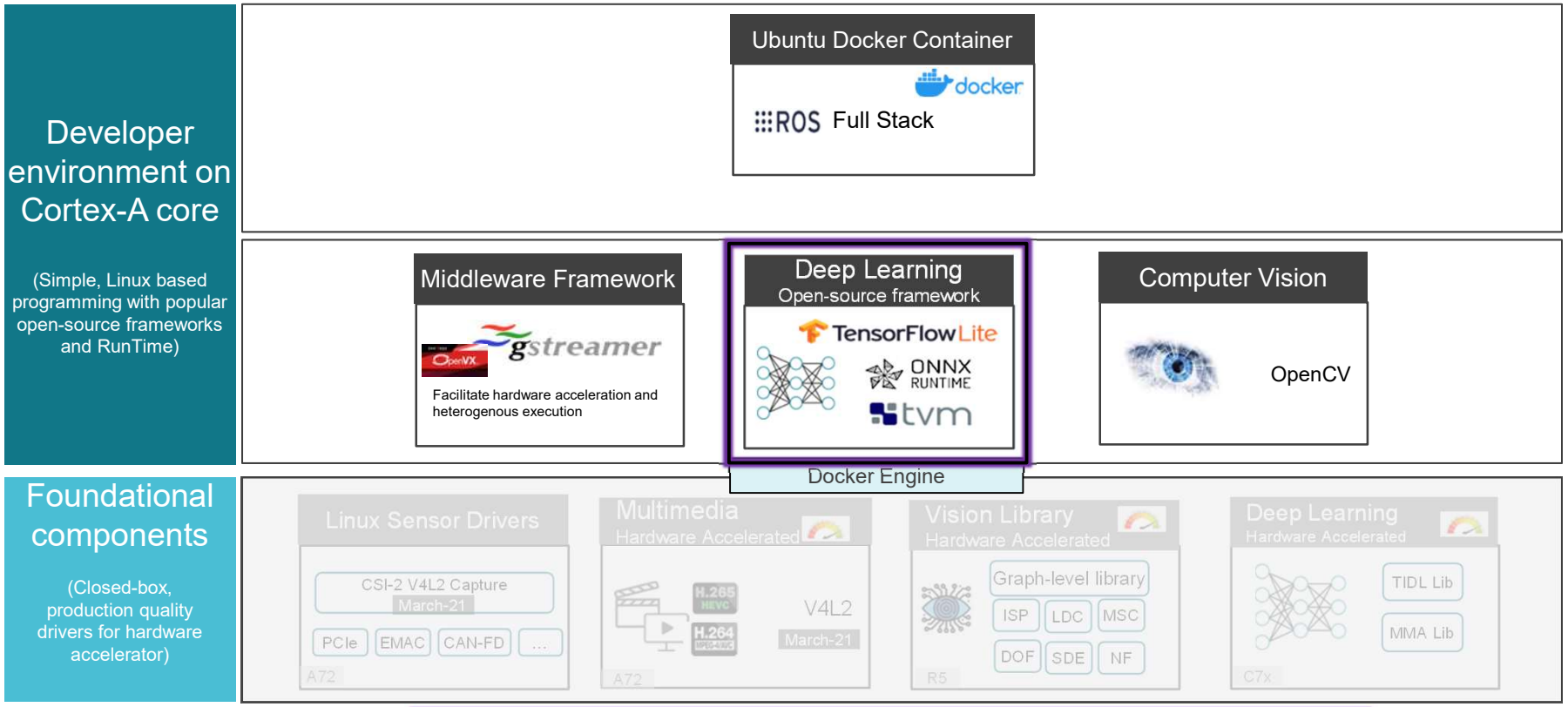
Designed for Functional Safety
SIL 2
ECC on data memory

- C7x DSP + Matrix Multiply Accelerator (MMA)
  - Programable accelerator for tensor, vector and scalar processing
- Self sustained for DL work-loads
  - No dependency on host ARM, GPU, has its own DMA engine and memory sub-system
- Smart memory architecture results in up to 90% utilization of the accelerator and DDR BW savings
  - High bandwidth interconnect, Large internal memory, 4D programmable DMA, Data forward engine



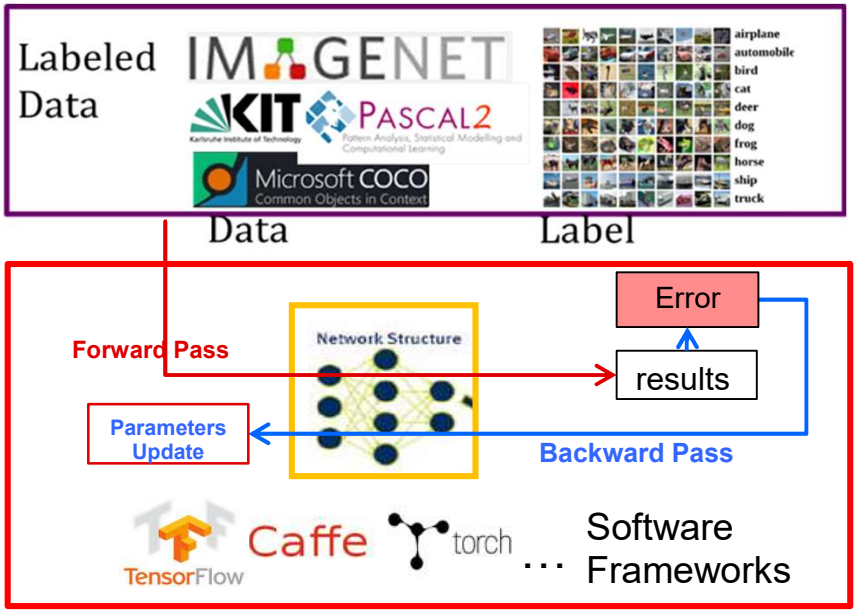
Embedded AI SW development flow

Comprehensive Software | offering from TI

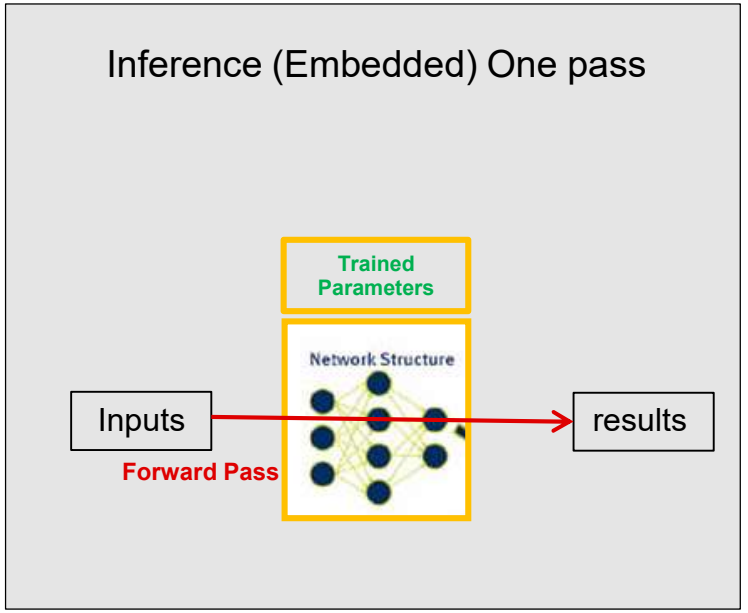


Focus for today. Other topics for future sessions

Deep learning | Introduction



Computing Platform



Embedded Hardware (TDA platform)

Adding AI in your stems | Three steps

DL Tools & software to reduces model development time

Accelerated inference using open-source industry standard RunTime Engines
Out of box optimized inference support for 60+ models

Train anywhere, Develop anywhere

1

PYTORCH mxnet Keras Caffe TensorFlowLite

1. **TI Model Zoo (60+ models)**
   - Model Selection tool
   - New weights
2. **Own model**

Optional QAT (Quantization artifacts tool) from TI

https://github.com/TexasInstruments/jacinto-ai-devkit

Compile & Optimize for TI SoC
Using industry standard Compilers/RTs

2

TVM Compiler/ TFLite / ONNX-RT

- Common representation
- Post Training Quantization
- Calibration
- Optimization
- Compilation

Deploy on TI SoC
Using industry standard APIs

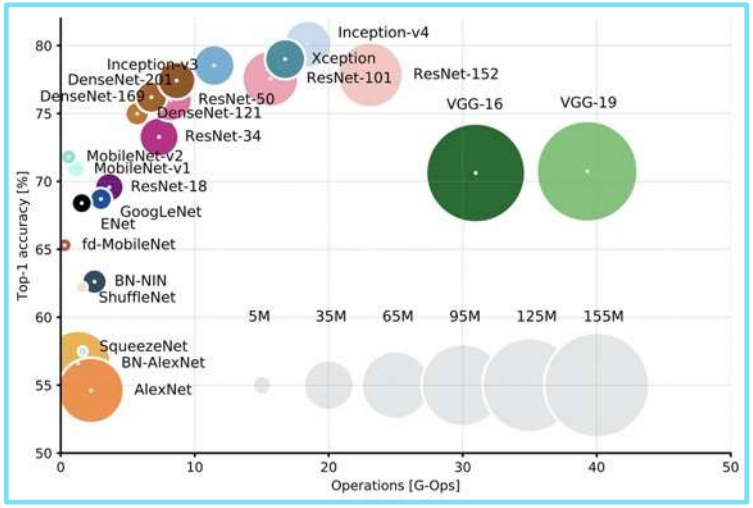
3

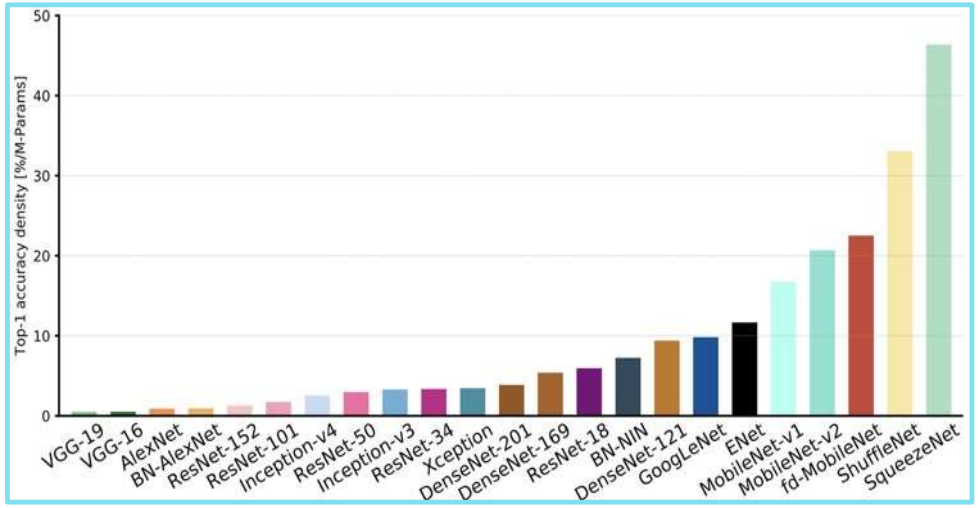
TFLite RT /ONNX-RT/Neo-AI-DLR

TI Edge AI Processor

TIDL RunTime	TIDL Library
Linux	RTOS
Cortex-A	DLA

Model section | Accuracy and performance trade-off





Accuracy vs Computing need

Top-1 Accuracy density (% per M-parameters)

Source: Analysis of deep neural networks. By Alfredo Canziani, Thomas Molnar… | by Eugenio Culurciello | Medium

CH

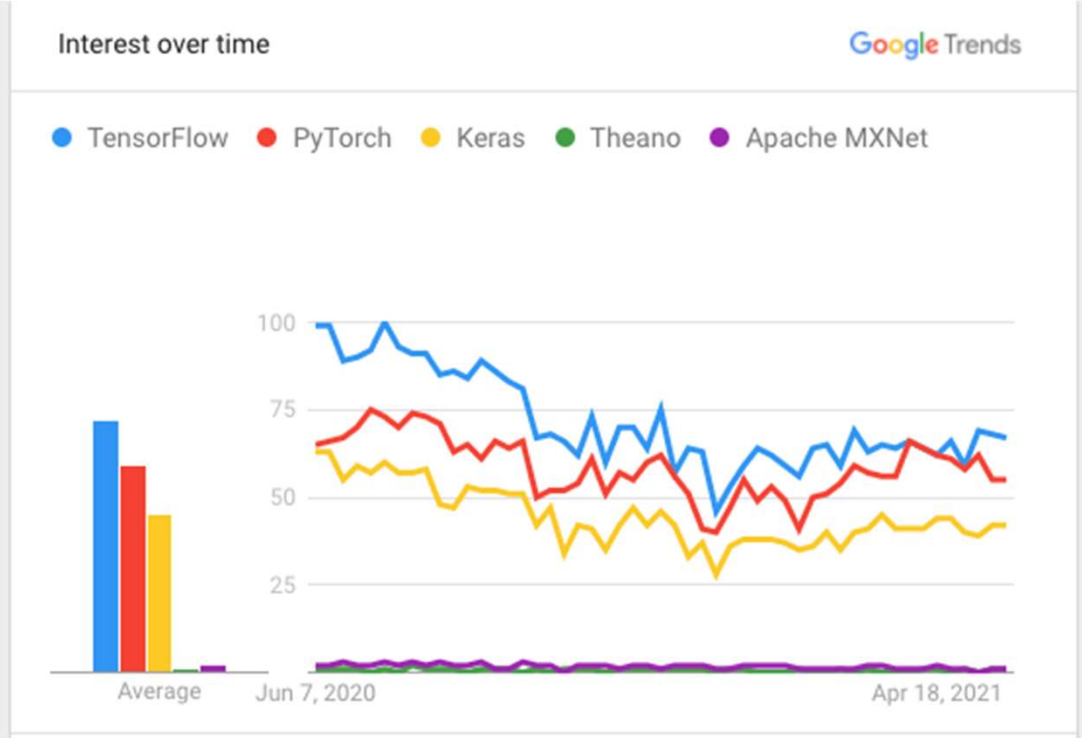
Extensive and Pre-trained | Models available ready to use

TI Model Selection Tool

Select Task (●) Classification ( ) SemanticSegmentation ( ) ObjectDetect	Select Device (●) TDA4VM
Select Performance Metric (●) fps ( ) Latency ( ) GMAC	Select Bubble Size (●) ddr-bw-mbps ( ) GMAC
Select Mode For Accuracy (●) int8 ( ) float	Select Resolution for Classification Task (●) 224x224
Select Accuracy Metric For Object Detect Task ( ) map50-95	Select SW Version (●) Current Performance ( ) Future Performance
Select RunTime Type ( ) All (●) TFLITE ( ) TVM ( ) ONNX-RT	Select Model Zoo Type (●) All ( ) Recommended

- TI's Model zoo: 60 plus models to choose from
- Select type of function: Classification, Detection or Segmentation
- Select the runtime: Tflite or TVM or ONNX-RT

Development flow | Popular DL frameworks



- TI supports popular frameworks
- Will focus on TensorFlow in this webinar
- Will use Python

Embedded AI “Hello World“ in 3 steps

1. Run the program first on the PC
2. Port the ”same program” to embedded platform: J7 EVM device
3. Run with deep learning acceleration

Deep learning | Model development

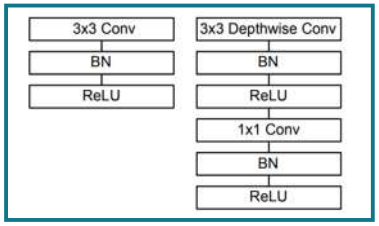
- How do we develop a model?
  1. Create a custom model from grounds up
  2. Use a pre-trained model such as MobileNet, Resnet, InceptionNet, …etc
  3. Apply Transfer learning on a pre-trained model

- In this webinar, we will use option 2)

- Model location: Tensorflow.org

  http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_2018_01_28.tar.gz

Example Network | MobileNet v1



Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Input

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Output

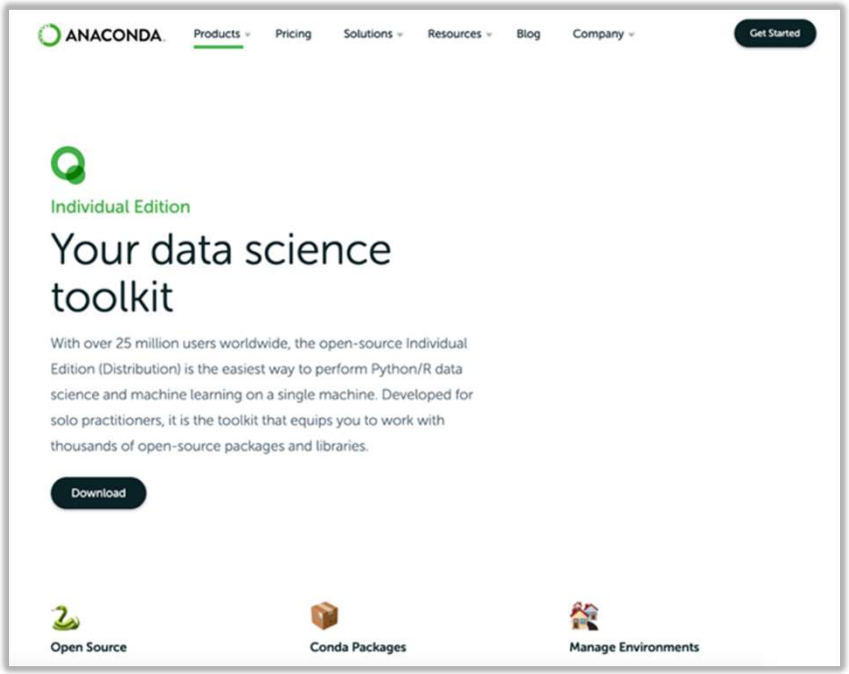
Labels (numbers associated to dog, cat…etc)

Year	Month	Net Name	# Prms	MMACs	Top 1 %	Top 5 %
1998	NA	Lenet (mnist)	0.431	2.3	95%	NA
2012	NA	AlexNet	60.9	726	57.2	80.3
2013	NA	Zfnet	62.3	1170	64	85.3
2014	NA	VGG16	138	15300	72	90.6
2015	CVPR	GoogleNet	7	1500	69.8	90.8
2015	Feb	BN-Inception	7	1500	69.8	89.6
2015	Dec	ResNet(50)	25.5	3860	75.2	92.2
2016	Feb	SqueezeNet	1.2	833	57.5	80.3
2016	Oct	Xception	22.8	NA	79	94.5
2016	Aug	DenseNet(121)	7.98	3080	75	92.2
2016	Nov	ReNext	25.5	4200	77.8	93.4
2017	April	MobileNets V1	4.3	569	70.9	89.9
2017	Jul	ShuffleNet V1	2	524	73.7	92
2017	Sep	SE Net (resnet50	30	3870	77.7	93.4
2018	Jan	MobileNetV2	3.4	300	72	92.5
2018	Jul	ShuffleNet V2	2	299	72.5	92.8
2019	May	MobileNetV3	5.4	219	75.2	93.4

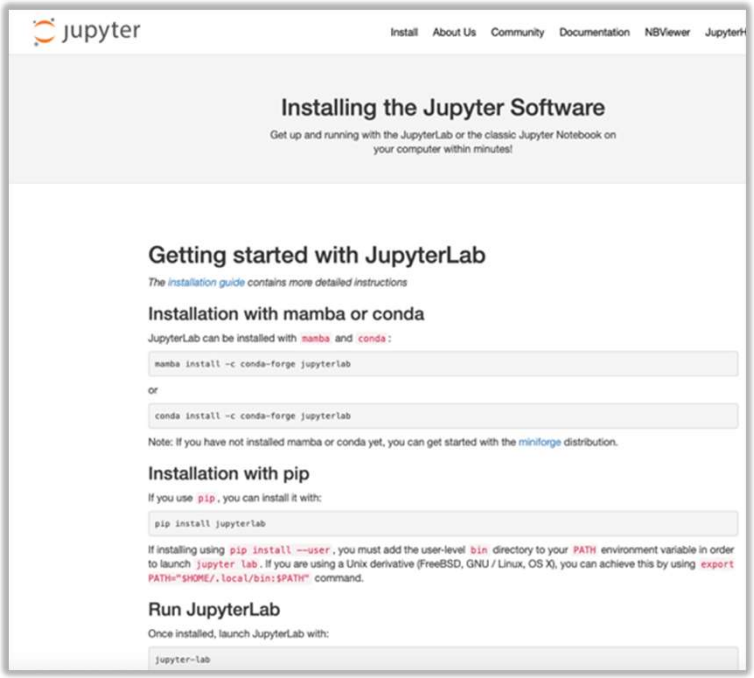
Now, Let's see what SW we need to develop our 1st application!

Reference: https://arxiv.org/pdf/1704.04861.pdf

Hello world AI | development environment



https://www.anaconda.com/products/individual

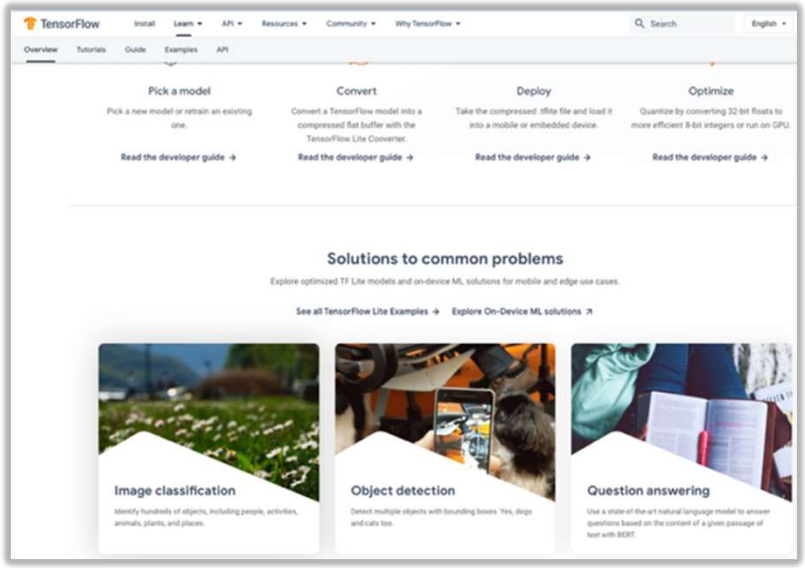


https://jupyter.org/install

Many other frameworks available for development

Example development flow | Tensorflow lite

- Tensorflow is open-source deep learning runtime framework
- Tflite is ”light-weight” version of Tensorflow targeting edge devices
  - Smaller memory and efficient for accuracy
- Tflite achieves this using
  - Quantization: weights, biases, activation functions and operations (J7 platform can support both 32-bit floating point and fixed point)
  - Weight pruning: Trimming parameters in a model that have little impact on the performance
- https://www.tensorflow.org/lite



Install from the Anaconda Powershell

```
$ pip3 install tensorflow
$ pip3 install --index-url https://google-coral.github.io/py-repo/ tflite_runtime
```

OpenCV | Computer vision SW library

- OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.
  - https://opencv.org/
- Provides a common infrastructure for computer vision applications
- Has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.
- Has C++, Python, Java and MATLAB interfaces

```
$ pip install opencv-python --user
```

Numpy | Numerical Computing in Python

- Open-source fundamental package for computing in Python
  - https://numpy.org/

- At the core of the NumPy package, is the ndarray object.
  - This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

- Vectorized operations: concise, intuitive and faster!
  - Broadcasting with other scalars or other matrices
  - Many intuitive operations are implemented in this library

```
$ pip install numpy
```

We installed everything now… Let's jump to the the code…

Hello world | Three main code segments

```
import numpy as np
import tflite_runtime.interpreter as tflite
import cv2
```

1

Import three main packages

- OpenCV, Numpuy and Tensotflow

```
# Load TFLite model and allocate tensors.
label_dict = make_label_dict('TFL-OD-200-ssd-mobV1-coco-mlperf-
300x300/labels.txt')

# Use a pre-trained model compiled using TIDL
tflite_model_path = 'TFL-OD-200-ssd-mobV1-coco-mlperf-
300x300/ssd_mobilenet_v1_coco_2018_01_28.tflite'

interpreter = tflite.Interpreter(model_path=tflite_model_path)
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

2

Select/Run the model

- Initialize and load the tflite model
- Allocate the tensor, get input and output tensors

```
img  = Image.open(image_file).convert('RGB')
img_in = preprocess(image_file , size, mean, scale, layout,
reverse_channels)
             img = cv2.resize(img, (size[1], size[0]),
interpolation=cv2.INTER_CUBIC)
interpreter.set_tensor(input_details[0]['index'], img_in)
interpreter.invoke()
res = [interpreter.get_tensor(output_detail['index']) for
output_detail in output_details]
```

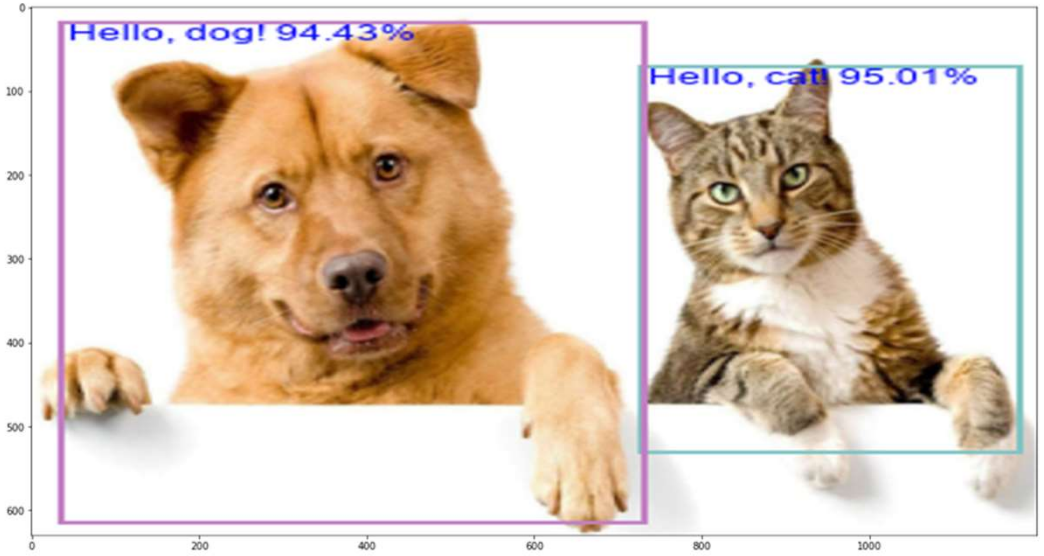
3

Run the model on the input

- Process the input to **match** with the model
- **Invoke** the Interpreter with the input
- Analyze the output

Good reference: https://github.com/tensorflow/models/tree/master/research/object_detection/utils

Hello world | Tensorflow lite



Elapsed time: 0.45 seconds

System: Windows PC i7 CPU
Tensorflow runtime engine without GPU

Embedded AI “Hello World“ in 3 steps

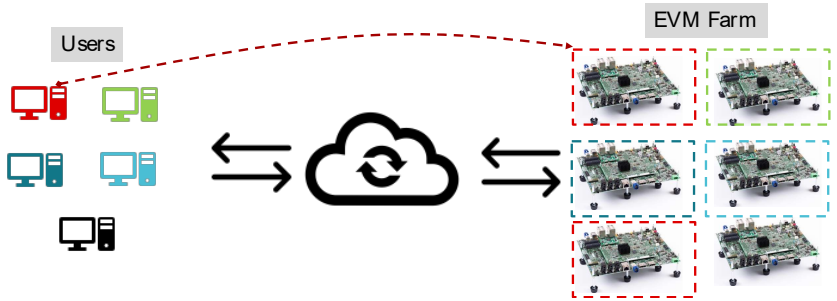
1. Run the program first on the PC
2. Port the ”same program” to embedded platform: J7 EVM device
3. Run with deep learning acceleration

Same Python program on to the Jacinto evaluation module

You don’t have an EVM? No Problem!

No need to purchase an EVM to get started | TI Cloud Tools

Free on-line service, enable deep learning evaluation **instantly**



No EVM needed

Evaluate TI SoC DL capabilities in remote EVM farm using web browser, Jupyter Notebook

Benchmarking

Collect latency, FPS, accuracy, DDR BW and Power benchmarks in minutes

TensorFlow Lite ONNX RUNTIME tvm

- < 1 min to explore & compere performance : Model Selection tool
- < 5 min to experience SW & evaluate HW : TI Model Zoo examples
- < 30 min to evaluate custom models : Custom model examples
- 1 hr+ to benchmark performance : TI Model Zoo examples

Available now at **https://dev.ti.com/edgeai**

Edge AI In the Cloud | Faster evaluation

Find your model

Learn performance statistics

< 1 min

Compare model performance

Find the model that best meet your performance and accuracy goals on TI Processor from TI Model Zoo. Learn current performance statistics of models such as FPS, Latency, Accuracy & DDR bandwidth.

TI Model Zoo

Model selection tool

Get model benchmarks

The following Notebooks let you access benchmarks for pre-compiled and custom models

< 5 min

Model performance

Using a pre-compiled model from TI Model Zoo, this example notebook lets you run inference on a TI Edge AI processor to get **latency, FPS, DDR bandwidth and power benchmarks**

Select task:

- (•) Classification
- ( ) Detection
- ( ) Segmentation

Select runtime engine:

- (•) TensorFlow lite
- ( ) ONNX runtime
- ( ) TVM

Open notebook

1 hr+

Model accuracy

Using a pre-compiled model from TI Model Zoo, this example notebook lets you run inference on a TI Edge AI processor to get **accuracy benchmarks**.

Open notebook

< 30 min

Custom models

This notebook lets you compile your own model and run inference on a TI Edge AI processor to get **latency, FPS, DDR bandwidth, power and accuracy benchmarks**.

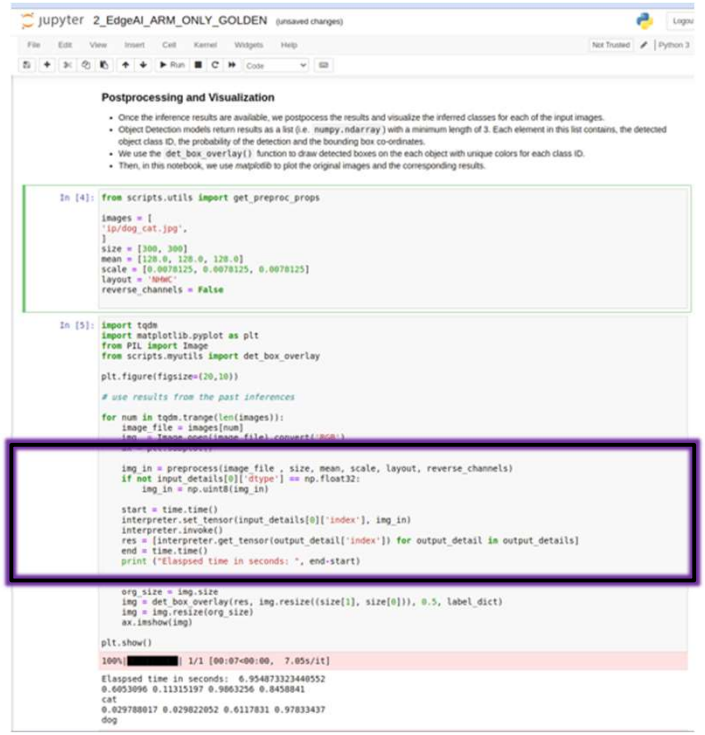
Select runtime engine:

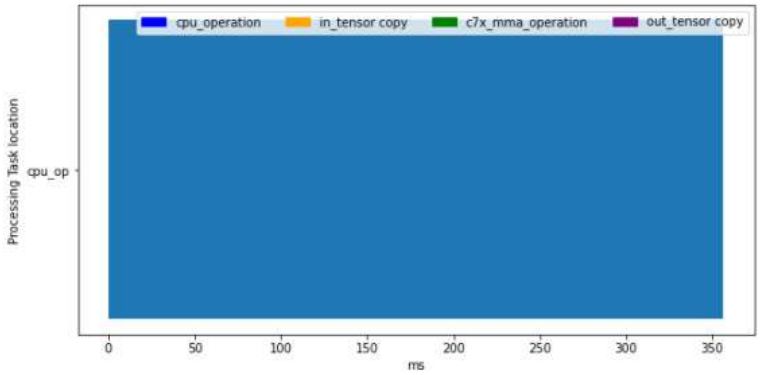
- (•) TensorFlow lite
- ( ) ONNX runtime
- ( ) TVM

Open notebook

Use custom model and open the same program that ran on the PC

Hello world | on Jacinto EVM in the cloud

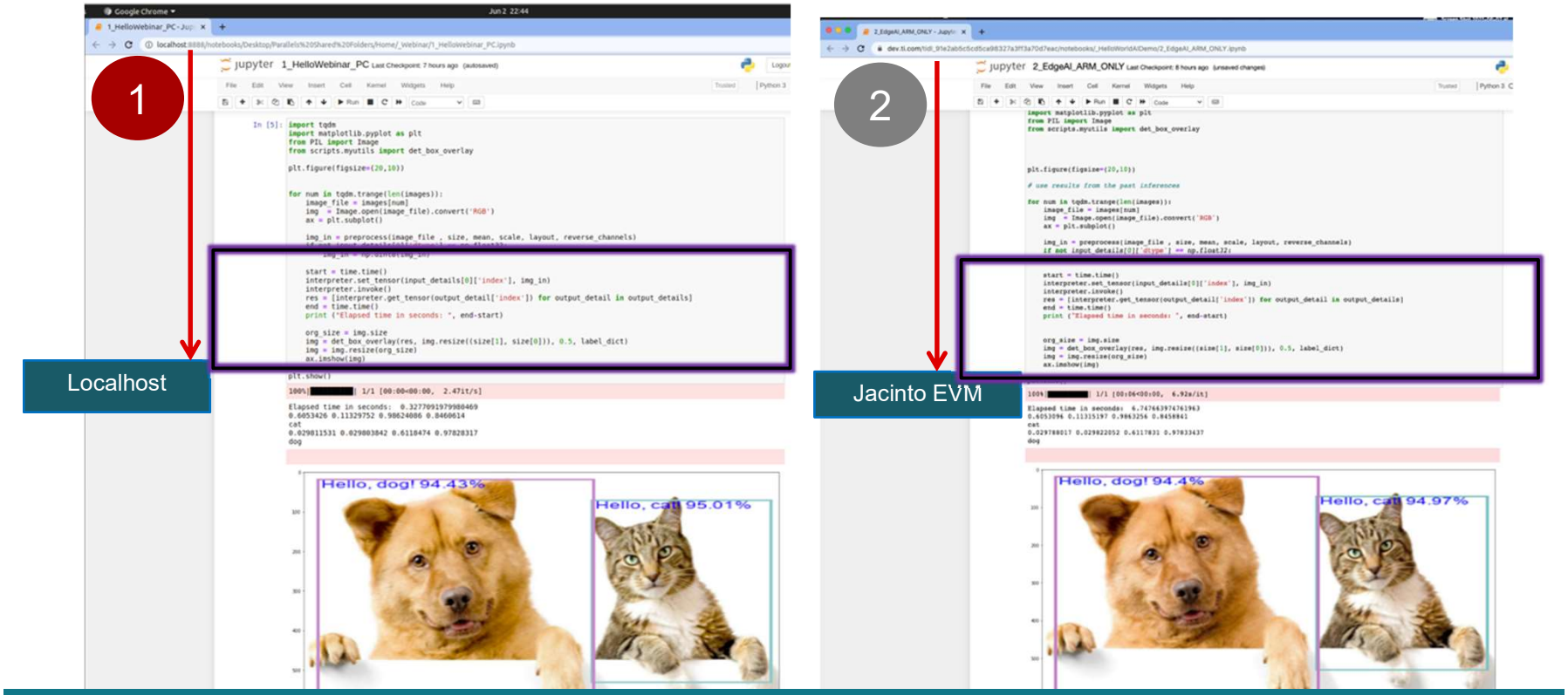





```
SoC: J721E/DRA829/TDA4VM
 OPP:
  Cortex-A72 @2GHZ
  DSP C7x-MMA @1GHZ
  DDR @4266 MT/s

 Inferences Per Second    :    2.81 fps
 Inference Time Per Image :  356.39 ms
 DDR usage Per Image      :    0.00 MB
```

Utility to measure performance

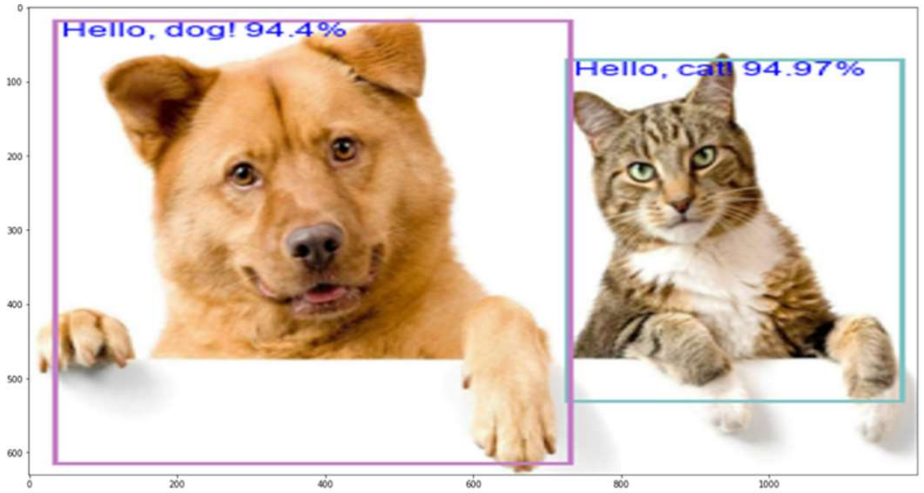
Exact same program | on PC and the Jacinto hardware

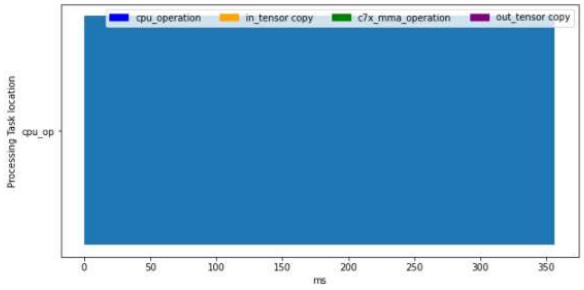


Customers do not need to learn any new language. Same Python program can be run on Jacinto hardware!

CH

Hello world | Now on the embedded device!






```
SoC: J721E/DRA829/TDA4VM
 OPP:
  Cortex-A72 @2GHZ
  DSP C7x-MMA @1GHZ
  DDR @4266 MT/s

 Inferences Per Second    :    2.81 fps
 Inference Time Per Image :  356.39 ms
 DDR usage Per Image      :    0.00 MB
```

Inference Time Per Image : 0.3 seconds

It works. But, how can we use this for real-time video?

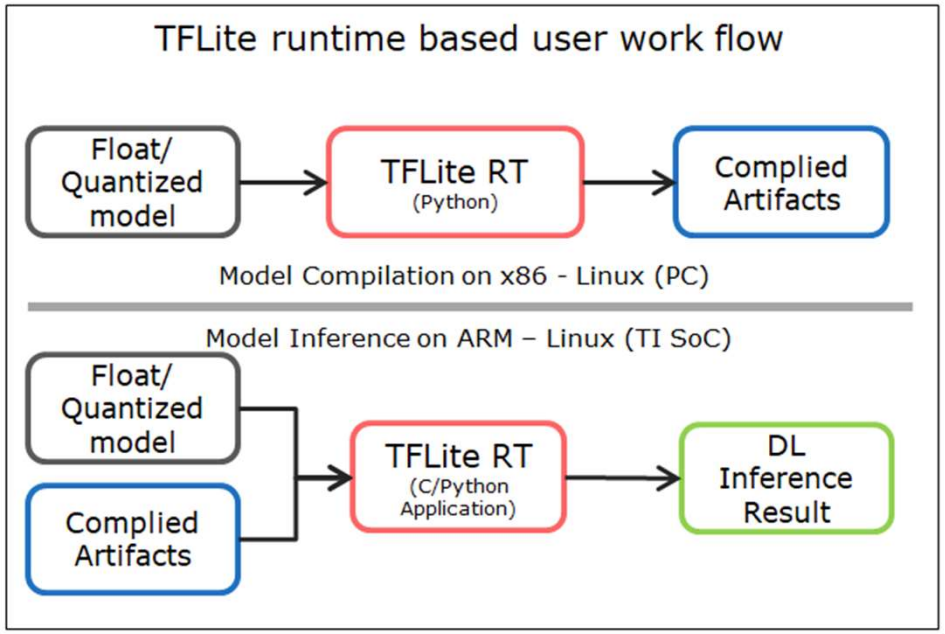
Now, it is running only on the ARM cores

Let’s bring in Deep learning hardware accelerators!

Embedded AI “Hello World“ in 3 steps

1. Run the program first on the PC
2. Port the ”same program” to embedded platform: J7 EVM device
3. Run with deep learning acceleration

Model porting | PC to embedded device



TFLite runtime based user work flow

http://software-dl.ti.com/jacinto7/esd/processor-sdk-rtos-jacinto7/07_03_00_07/exports/docs/tidl_j7_02_00_00_07/ti_dl/docs/user_guide_html/index.html

Hello world | on Jacinto EVM in the cloud (DL acceleration)

```
[4]: calib_images = [
'../sample-images/elephant.bmp',
'../sample-images/bus.bmp',
'../sample-images/bicycle.bmp',
'../sample-images/zebra.bmp',
]

# Make a dictionary of labels and class values
label_dict = make_label_dict('TFL-OD-200-ssd-mobV1-coco-mlperf-300x300/labels.txt')

output_dir = 'custom-artifacts'
tflite_model_path ='TFL-OD-200-ssd-mobV1-coco-mlperf-300x300/ssd_mobilenet_v1_coco_2018_01_28.tflite'


compile_options = {
    'tidl_tools_path' : os.environ['TIDL_TOOLS_PATH'],
    'artifacts_folder' : output_dir,
    'tensor_bits' : 8,
    'accuracy_level' : 1,
    'advanced_options:calibration_frames' : len(calib_images),
    'advanced_options:calibration_iterations' : 3,
}

# create the output dir if not preset
# clear the directory
if not os.path.exists(output_dir):
    os.makedirs(output_dir, exist_ok=True)

for root, dirs, files in os.walk(output_dir, topdown=False):
    [os.remove(os.path.join(root, f)) for f in files]
    [os.rmdir(os.path.join(root, d)) for d in dirs]

tidl_delegate = [tflite.load_delegate(os.path.join(os.environ['TIDL_TOOLS_PATH'], 'tidl_model_import_tflite.so'), c
interpreter = tflite.Interpreter(model_path=tflite_model_path, experimental_delegates=tidl_delegate)
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

import tqdm

mean = [128.0, 128.0, 128.0]
scale = [0.0078125, 0.0078125, 0.0078125]
layout = 'NHWC'
reverse_channels = False

for num in tqdm.trange(len(calib_images)):
    #interpreter.set_tensor(input_details[0]['index'], preprocess_for_tflite_mobilenetv1(calib_images[num]))
    interpreter.set_tensor(input_details[0]['index'], preprocess(calib_images[num] , size, mean, scale, layout, rev
    interpreter.invoke()

100%|██████████| 4/4 [01:25<00:00, 21.49s/it]
```

Two things are new in this step

1) Compile the model to the TI's DL accelerators
2) Assign the compiled output as a delegate.

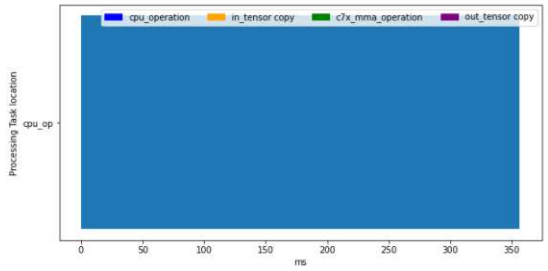
- Rest of the steps are similar to previous steps
- A delegate option in the Interpreter is a way to "**offload**" parts of the network to **hardware accelerator**
- Model compilation uses the same Tensorflowlite APIs
- Acceleration happens automatically by just adding one line of code- tflite.load_delegate (APIs coming from TFLite)

Use compiled model

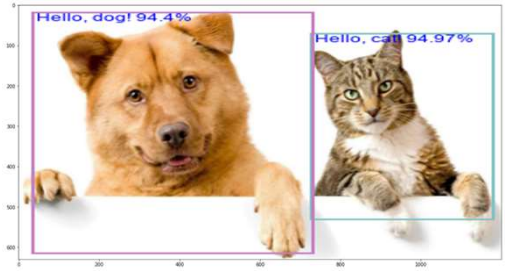
Then using ***TF Lite*** with the `libtidl_tfl_delegate` delegate library we run the model and collect benchmark data.

```
#from scripts.utils import imagenet_class_to_name
#import matplotlib.pyplot as plt

# Make a dictionary of labels and class values
label_dict = make_label_dict('TFL-OD-200-ssd-mobV1-coco-mlperf-300x300/labels.txt')

# Use the compled model
tidl_delegate = [tflite.load_delegate('libtidl_tfl_delegate.so', {'artifacts_folder': output_dir})]
interpreter = tflite.Interpreter(model_path=tflite_model_path, experimental_delegates=tidl_delegate)
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

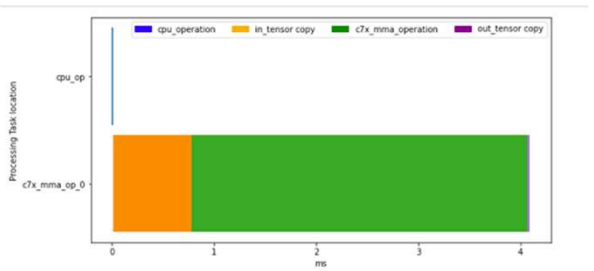
Hello world Now on the embedded device with acceleration



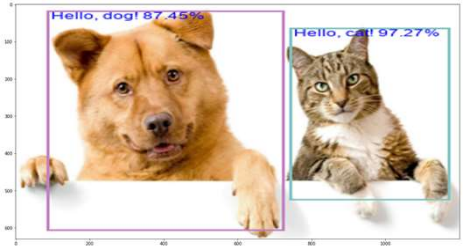

```
SoC: J721E/DRA829/TDA4VM
 OPP:
  Cortex-A72 @2GHZ
  DSP C7x-MMA @1GHZ
  DDR @4266 MT/s

 Inferences Per Second    :    2.81 fps
 Inference Time Per Image :  356.39 ms
 DDR usage Per Image      :    0.00 MB
```



❑ 2.81 fps AI processing (ONLY ARM)




```
SoC: J721E/DRA829/TDA4VM
 OPP:
  Cortex-A72 @2GHZ
  DSP C7x-MMA @1GHZ
  DDR @4266 MT/s

 Inferences Per Second    :  303.06 fps
 Inference Time Per Image :    3.30 ms
 DDR usage Per Image      :   13.51 MB
```

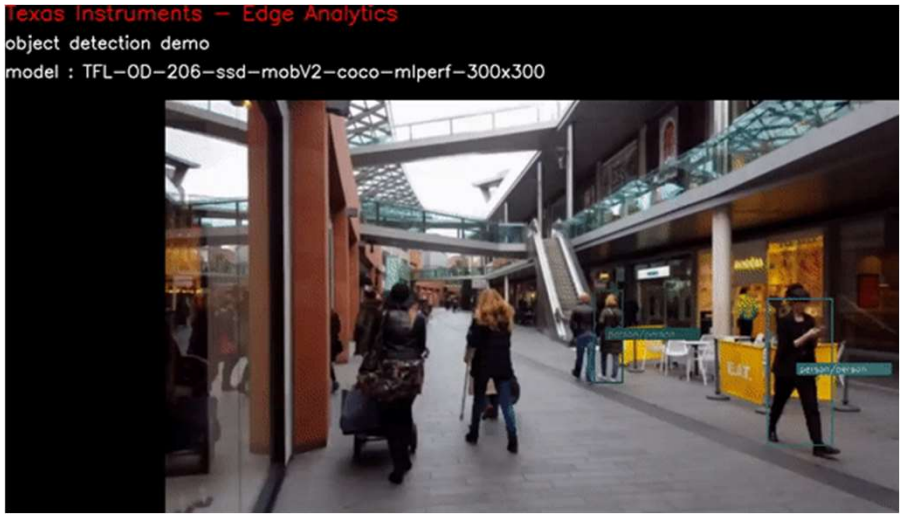


❑ 303.6 fps AI processing (TIDL acceleration)
CPU is involved only in the beginning for small amount of time (Blue)

Real world use case | AI acceleration

- Inference Time Per Image : 3.3 ms
- >300 fps object detection
- Input sequence of images or video in real-time

Video output clip from the actual Jacinto Hardware



Video input clip: https://pixabay.com/videos/shopping-mall-1887/

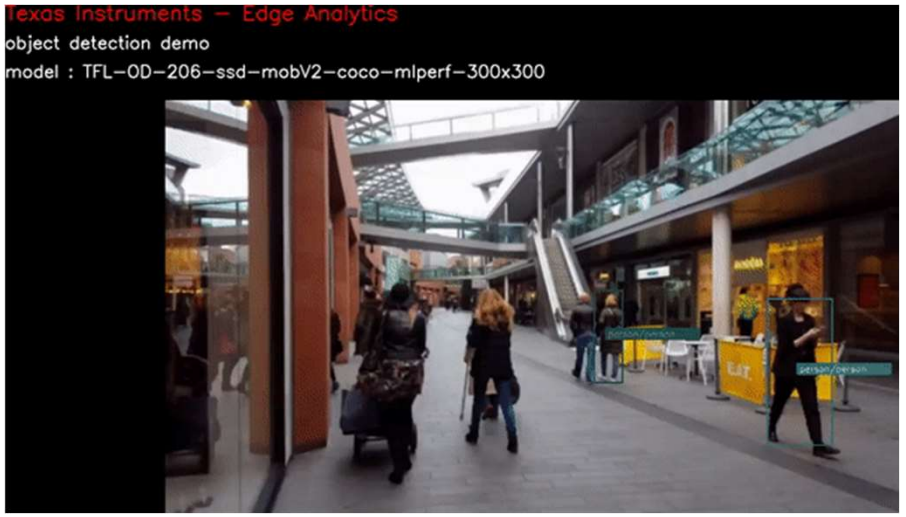
Unlimited possibilities with this kind of acceleration

Demo and code walk-through (10 mins)

Real world use case | AI acceleration

- Inference Time Per Image : 3.3 ms
- >300 fps object detection
- Input sequence of images or video in real-time

Video output clip from the actual Jacinto Hardware



Video input clip: https://pixabay.com/videos/shopping-mall-1887/

Unlimited possibilities with this kind of acceleration

Embedded Edge AI | Example use cases

1

- Automatic object detection
- Adaptive zoom area of interest
- Real-time decision making

Image Classification

2

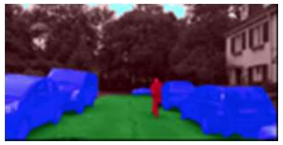
- Collision avoidance
- Hazard detection
- Last-mile delivery

Object Detection

3

- Automate shopping
- Fraud detection
- Convenience

Semantic Segmentation



AI embedded. AI in low power. Unlimited possibilities.

Beyond Hello World Further Development

- Advanced development
  - TI Model zoo – Choose the right model for your applications
  - Other development frameworks
  - Performance estimation and optimization

AI in your stems | Three steps

DL Tools & software to reduces model development time

Accelerated inference using open-source industry standard RunTime Engines
Out of box optimized inference support for 60+ models

Train anywhere, Develop anywhere

1

PYTORCH mxnet Keras Caffe TensorFlowLite

1. TI Model Zoo (60+ models)
   - Model Selection tool
   - New weights
2. Own model

Optional QAT (Quantization artifacts tool) from TI

https://github.com/TexasInstruments/jacinto-ai-devkit

Compile & Optimize for TI SoC
Using industry standard Compilers/RTs

2

TVM Compiler/ TFLite / ONNX-RT

- Common representation
- Post Training Quantization
- Calibration
- Optimization
- Compilation

Deploy on TI SoC
Using industry standard APIs

3

TFLite RT /ONNX-RT/Neo-AI-DLR

TI Edge AI Processor

TIDL RunTime	TIDL Library
Linux	RTOS
Cortex-A	DLA

CH

TI model zoo | Jump start your AI development

1. Jacinto-AI-ModelZoo

- Large collection of pre-trained models that are verified to work on our platform. 60+ models now and growing!
- Information on how to train some of the important models, and also the complexity and what accuracy to expect.
  -- Documentation: **link**
  -- git clone/pull URLs: link

2. Pre-Imported/Compiled Model Artifacts:

- This package provides Pre-Imported/Compiled Model Artifacts created using Jacinto-AI-ModelZoo and Jacinto-AI-Benchmark repositories.
- These artifacts can be used in multiple ways: (1) Jypyter Notebook examples in TIDL (2) For inference/benchmark in the above Jacinto-AI-Benchmark repository
  -- Documentation & download: link

3. Jacinto-AI-Benchmark

- Accuracy benchmarking of deep Learning models is a difficult task.
- We **make it easy** for our platform with a Python package that runs on PC and our EVM. **[Just a few lines of code]**
  -- Documentation: **link**
  -- git clone/pull URLs: link

Custom model | Supported Layers

1. Convolution Layer
2. Spatial Pooling Layer
   - Average and Max Pooling
3. Global Pooling Layer
   - Average and Max Pooling
4. Element Wise Layer
   - Add, Product and Max
5. Inner-Product (FC/Dense/Matmul) Layer
6. Soft-Max Layer
7. Bias Add Layer
8. Concatenate layer
9. Scale Layer
10. Batch Normalization layer
11. Re-size Layer
    - Bi-leaner/Nearest Neighbor Up-sample
12. Arg-max layer
13. ReLU Layer
14. RelU6 layer
15. PReLU (One Parameter per channel)
16. Slice layer
17. Crop layer
18. Flatten layer
19. Shuffle Channel Layer
20. Detection output Layer (SSD - Post Processing As defined in caffe-Jacinto and TF Object detection API)
21. Deconvolution/Transpose convolution
22. Custom/ User Defined Layer (Call Back)

Note : Please refer to TIDL users Guide for up to date information
https://software-dl.ti.com/jacinto7/esd/processor-sdk-rtos-jacinto7/07_03_00_07/exports/docs/tidl_j7_02_00_00_07/ti_dl/docs/user_guide_html/index.html

CH

Processor SDK | out-of-box machine vision demos

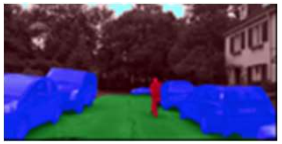
Deep Learning
Hardware accelerated

- Out-of-box example for image classification, object detection and semantic segmentation
- **Model zoo:** 50+ pre-trained TF, PyTorch, TFLite, ONNX and MXNet models validated on TI processors

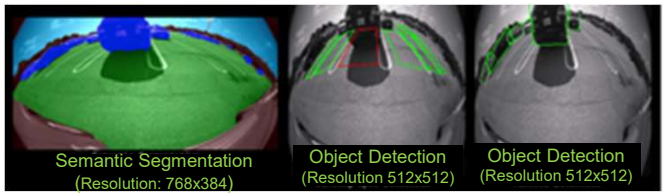
Image Classification

Object Detection

Semantic Segmentation



Demonstrate simultaneous execution of multiple models



- Three simultaneous models
- **Performance: 50.7 fps**
- OD Model: MobileNet V1+SSD
- SS Model: MobileNey V2 + ASPP
- Demo Link

Image pre-processing
Hardware accelerated

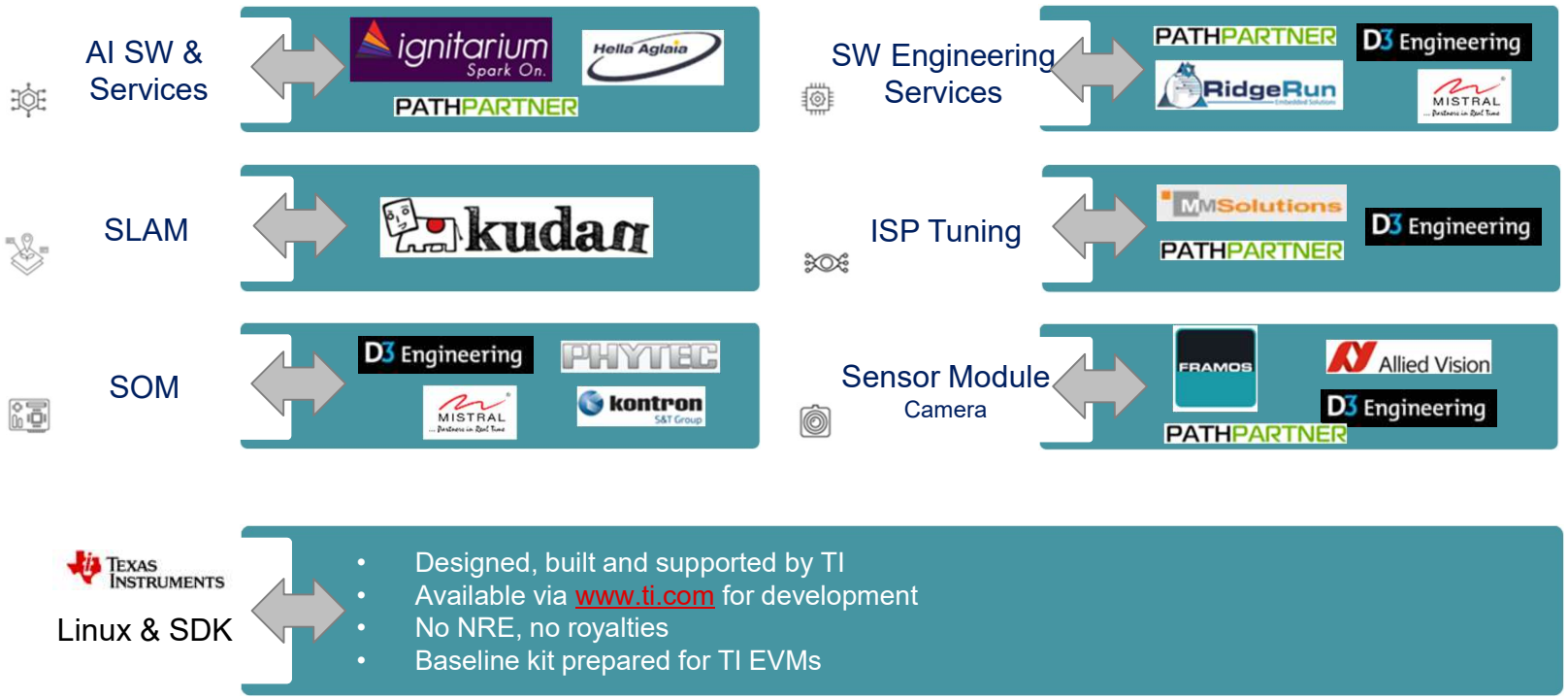
8x 2MP @ 30 fps real-time image processing
  - Room to process 2 more cameras
- Demonstrate RAW to RGB processing
- Image distortion correction
- Flexible programming sub-system

Extensive demos
Available now!

- Image Pre-processing demos
- Deep Learning demos

Available in SDK: Demo applications for deep learning & image processing

3rd party ecosystem | for broad range of applications



CH

Call to action

- ❑ Run the “hello world” example yourself
  - Download the example code and try with different images and video

- ❑ Reimagine “what’s possible” for your application with embedded edge AI

  Cloud Tool: https://dev.ti.com/edgeai

  Product Folder: https://www.ti.com/product/TDA4VM

  TDA4 EVM: http://www.ti.com/tool/TDA4VMXEVM

- ❑ Contact TI for support (e2e.ti.com)
  - Please also let us know any specific topics you want us to cover in the future webinars
  - Possible topics for future webinars (Subject to change)
    - Jul 21: Deep dive into **model compilation and optimization** in the embedded hardware
    - Aug 21: **Performance and power** benchmarking
    - Sep 21: Deep dive into **gStreamer** (video processing pipeline)

More Information: https://dev.ti.com/edgeai