AI & Machine Learning @ AWS

An Introduction

Daniel ZivKovic, Cloud Solutions Architect
TriNimbus, an Onica company

Office Lunch & Learn, September 28, 2018

TODAY WE WILL COVER

1. WHY LISTEN TO AMAZON?

2. MACHINE LEARNING OVERVIEW

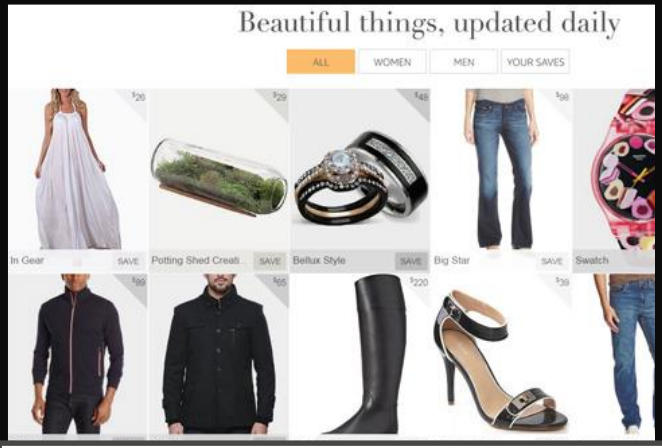
3. AWS ML STACK: APPLICATION SERVICES

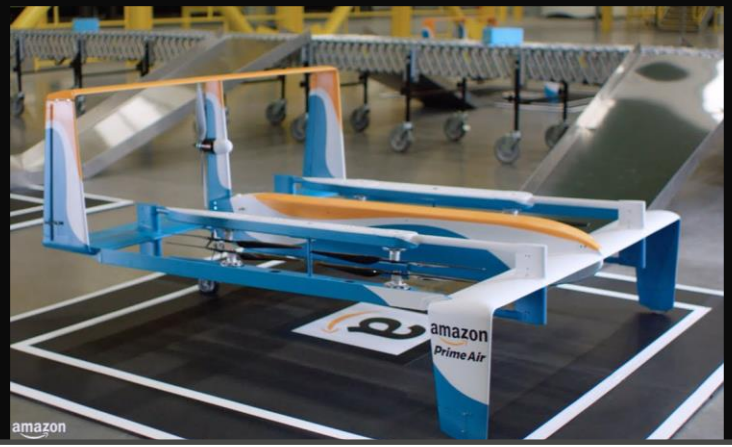
4. AWS ML STACK: PLATFORM SERVICES

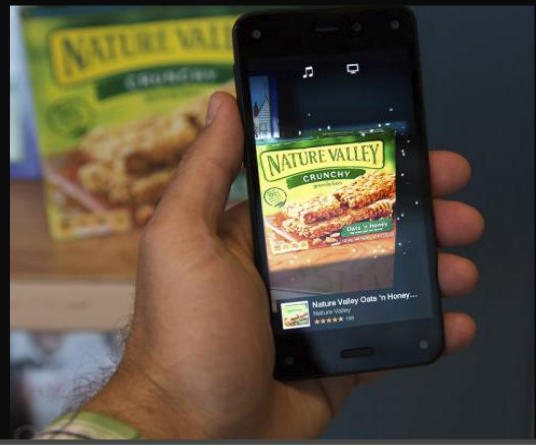
5. KEEP LEARNING

aws

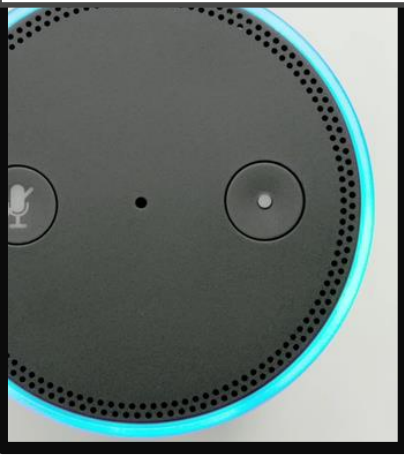
1. WHY LISTEN TO AMAZON?



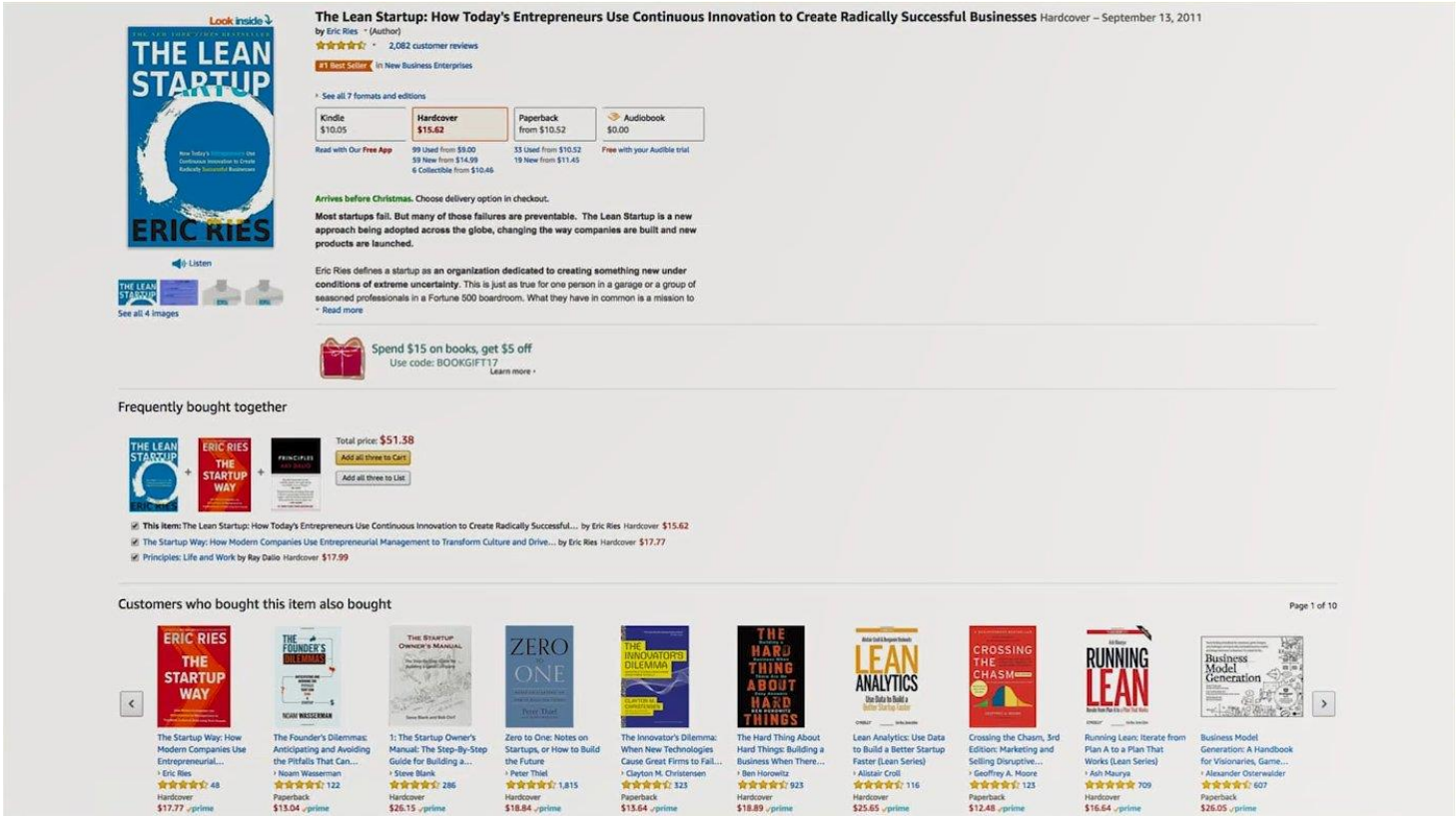




Two Decades of Machine Learning at Amazon.com



UI – PERSONALIZED RECOMMENDATIONS



Amazon.com Recommendations – Item-to-Item Collaborative Filtering algorithm, did to Barnes & Noble what Google PageRank algorithm did to Yahoo's search dominance – which was based on professional web surfers curating the web.

– Jeremy Howard

ALEXA – AI ENABLED #VoiceFirst UI

With voice, for the first time in human evolution ever – we can operate "tools" out of our hand's reach!

– Paul Cutsinger

Alexa was my "wake-up call" for AI, as before it no speech recognition program could handle my thick accent, hoarseness, and pace of speech.

– Daniel ZivKovic

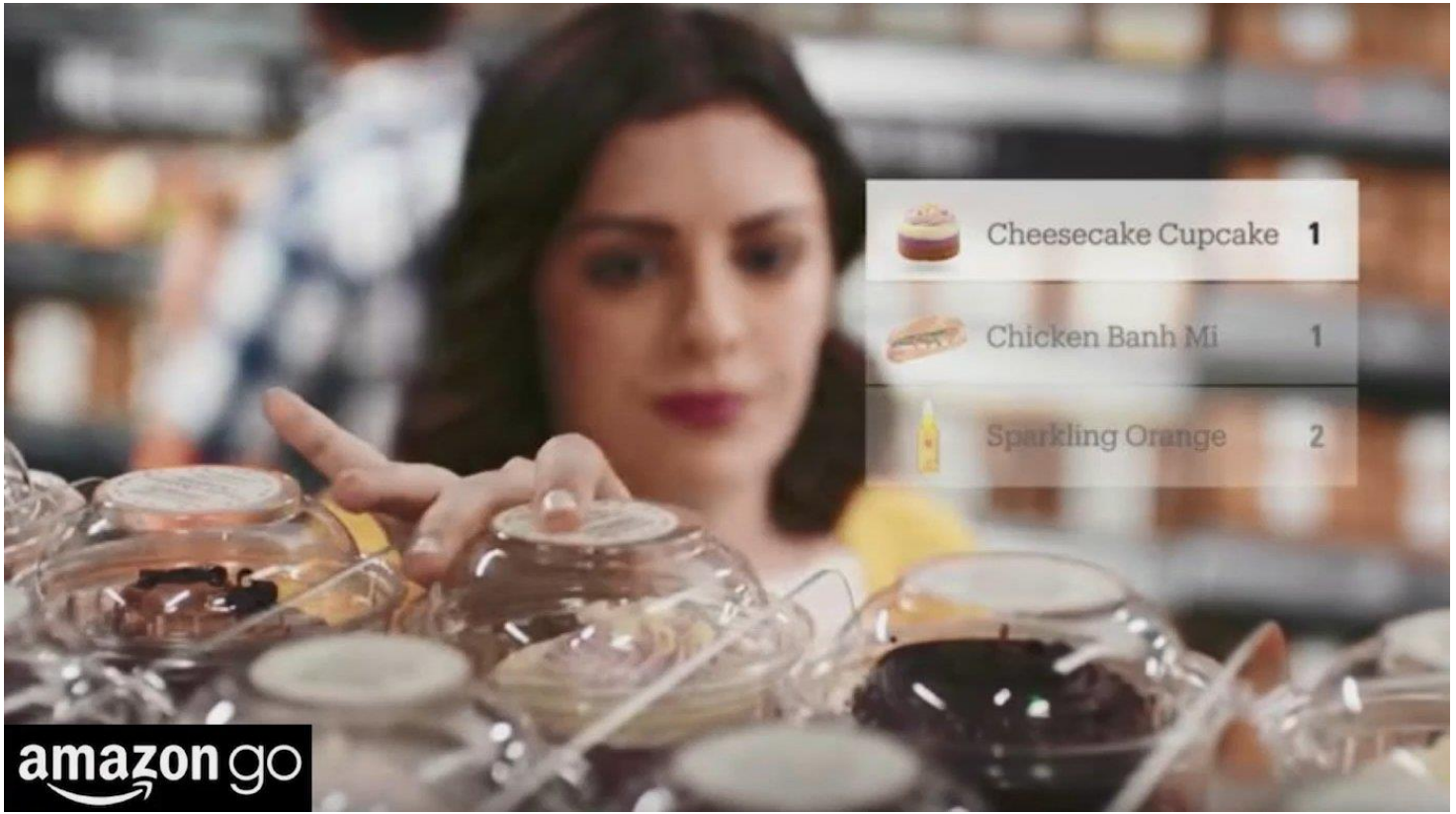
aws

50K ROBOTS ACROSS DELIVERY CENTRES

aws

DRONES – ALGOS LIKE IN SELF-DRIVING CARS

AMAZON GO – NO CHECKOUT, NO CASHIERS



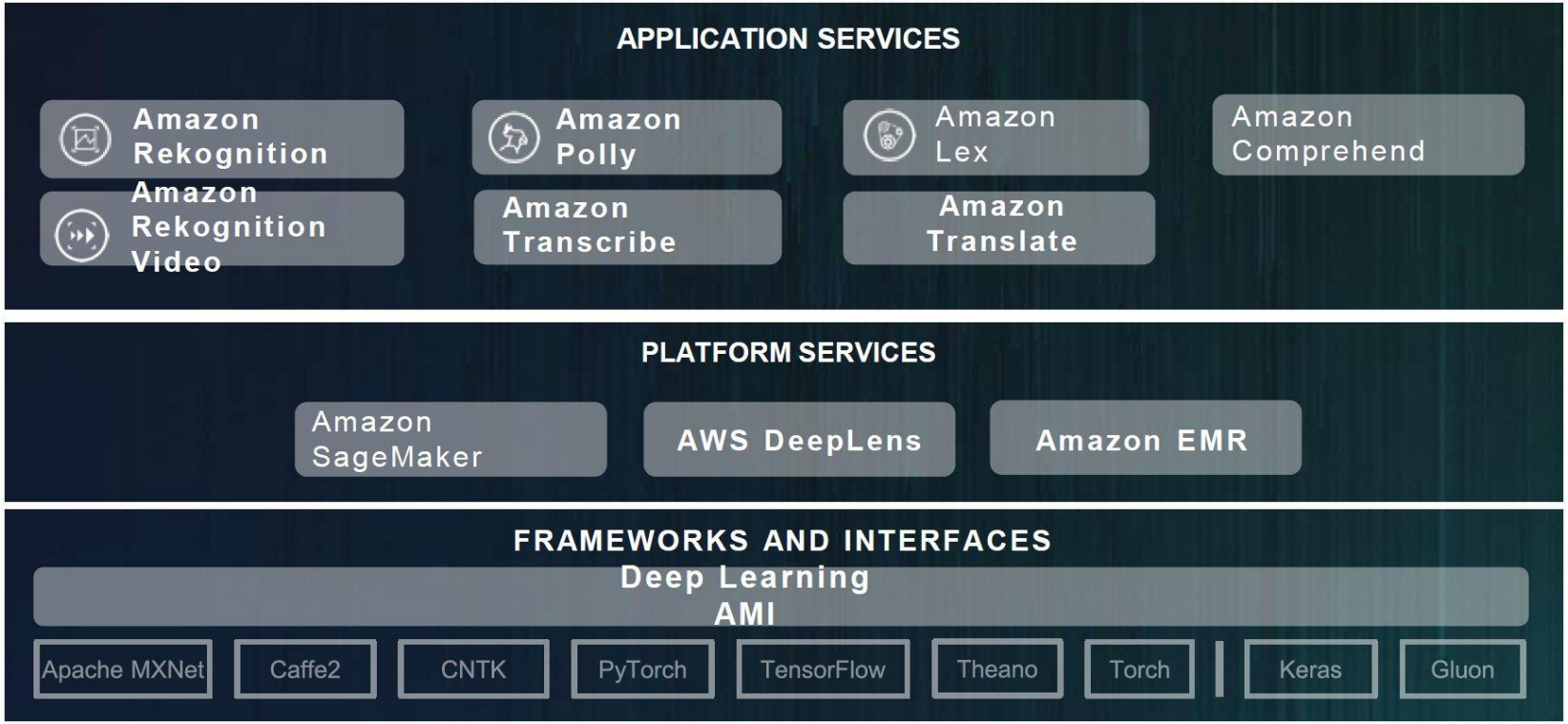
ML @ AWS MISSION

Put Machine Learning in the hands of **every** developer and data scientist

This mission statement feels as big as Bill Gates' - "A computer on every desk and in every home", or Steve Jobs' - "1000 Songs In Your Pocket" mission statements before.

aws

AWS ML STACK



ML USE CASES BY VERTICALS

MANUFACTURING

- Preventative maintenance
- Demand prediction
- Defect detection

FINANCE

- Fraud detection
- Customer UX/UI experience optimization
- Loan risk assessment

RETAIL

- Targeting
- Recommendation engines
- Dynamic pricing

HEALTHCARE

- Preventative care
- Diagnosis in medical imagery
- Drug discovery

OIL & GAS

- Reservoir analysis and modeling
- Alarm management and investigation
- Well test analysis and prediction

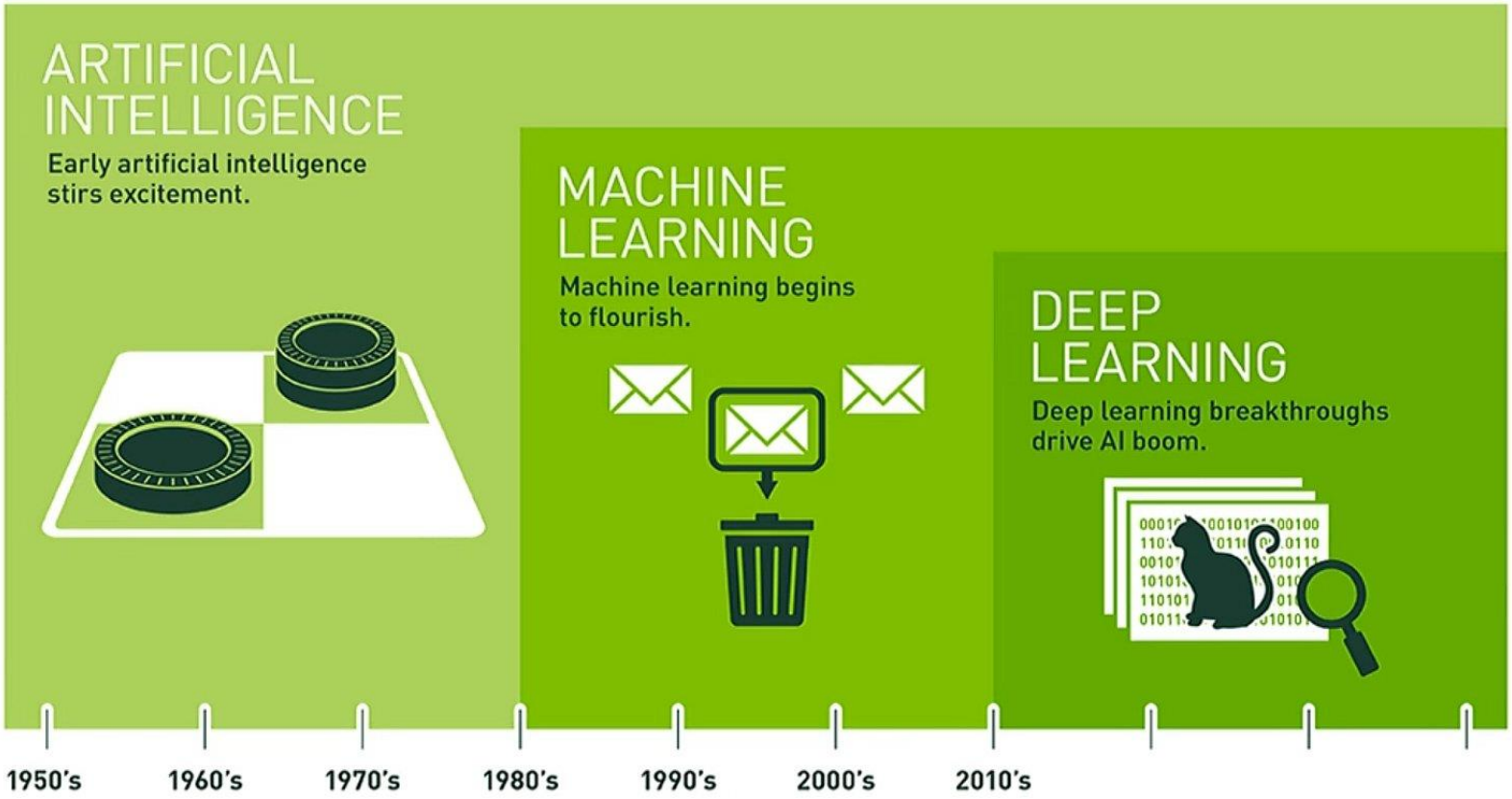
AGRICULTURE

- Predict beneficial plant genes
- Livestock IoT efficiency / optimization
- Crop health monitoring

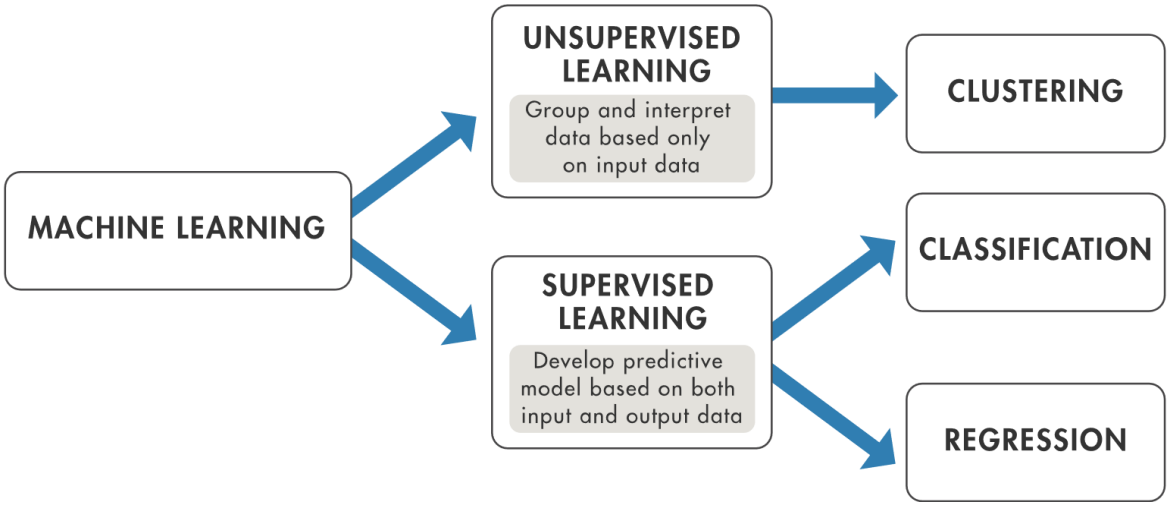
aws

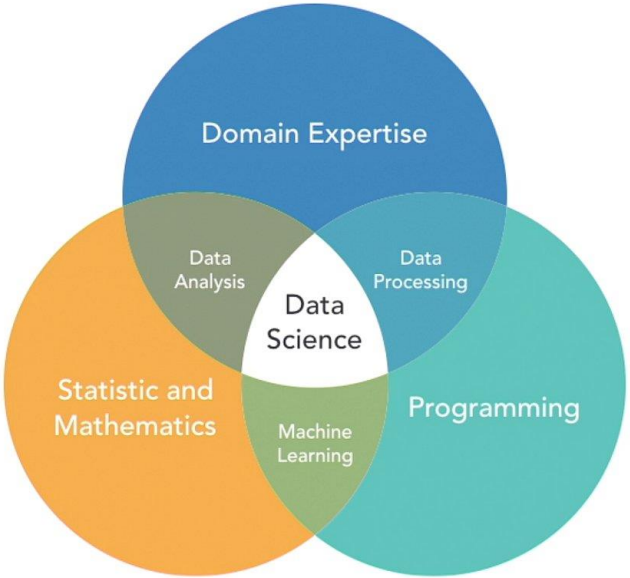
2. MACHINE LEARNING OVERVIEW

DEFINITIONS



MORE DEFINITIONS



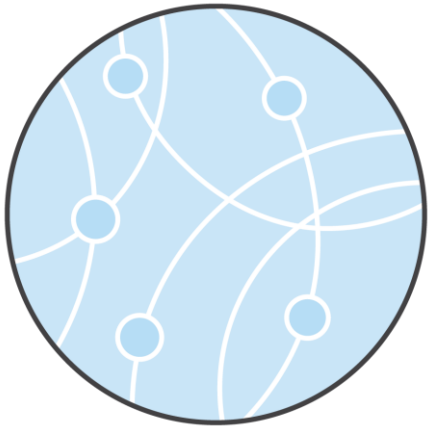


- Unsupervised – **clustering** finds hidden patterns in your data; draws inferences (conclusions) from datasets without labeled data
- Supervised – **regression** predicts future value of continuous variable like temperature or stock price
- Supervised – **classification** predict the likelihood of belonging to a specific group (e.g. cat or a dog)
- Supervised – most commonly includes algorithm, plus **labeled data**
- Data Science…

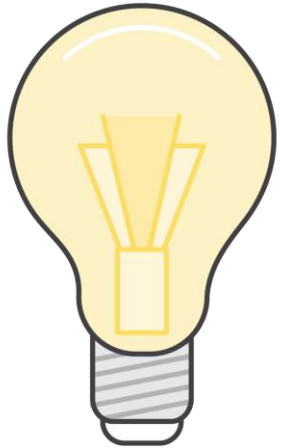
aws

OVERVIEW OF DEEP LEARNING

Data



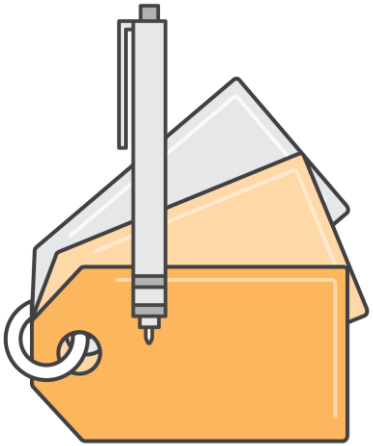
Model training



Inference

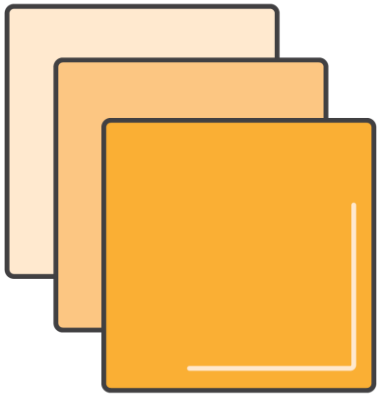
aws

DATA



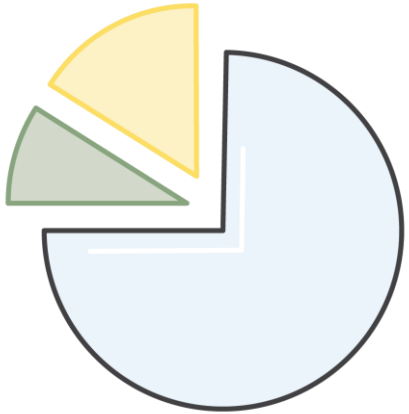
Annotate

Labeling



Pre-process

Sizing, formatting…

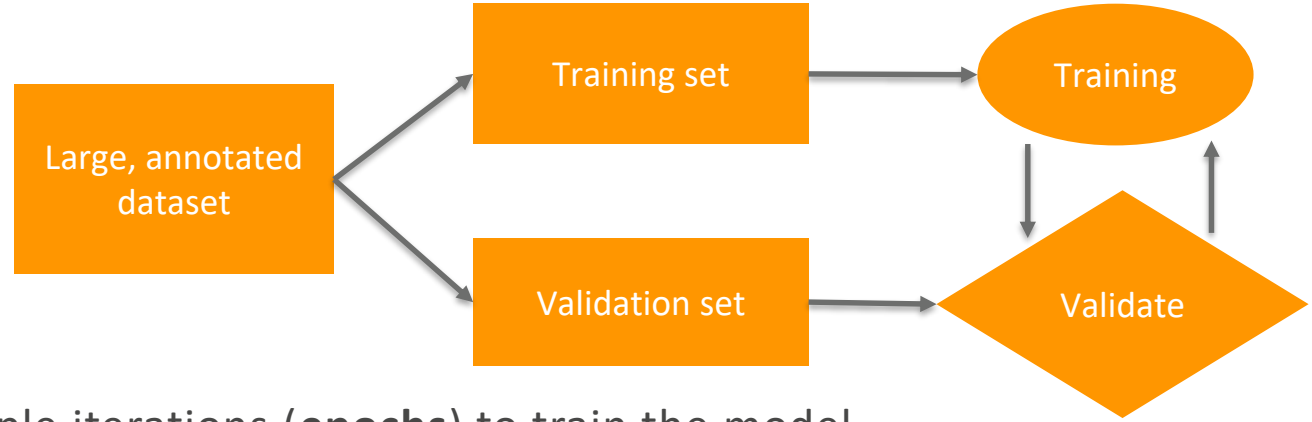


Data split

Training set vs
Validation set

MODEL DEVELOPMENT & TRAINING

- Define model architecture
- Input the annotated and cleaned data into the model



- Multiple iterations (**epochs**) to train the model
- Validate with held back dataset

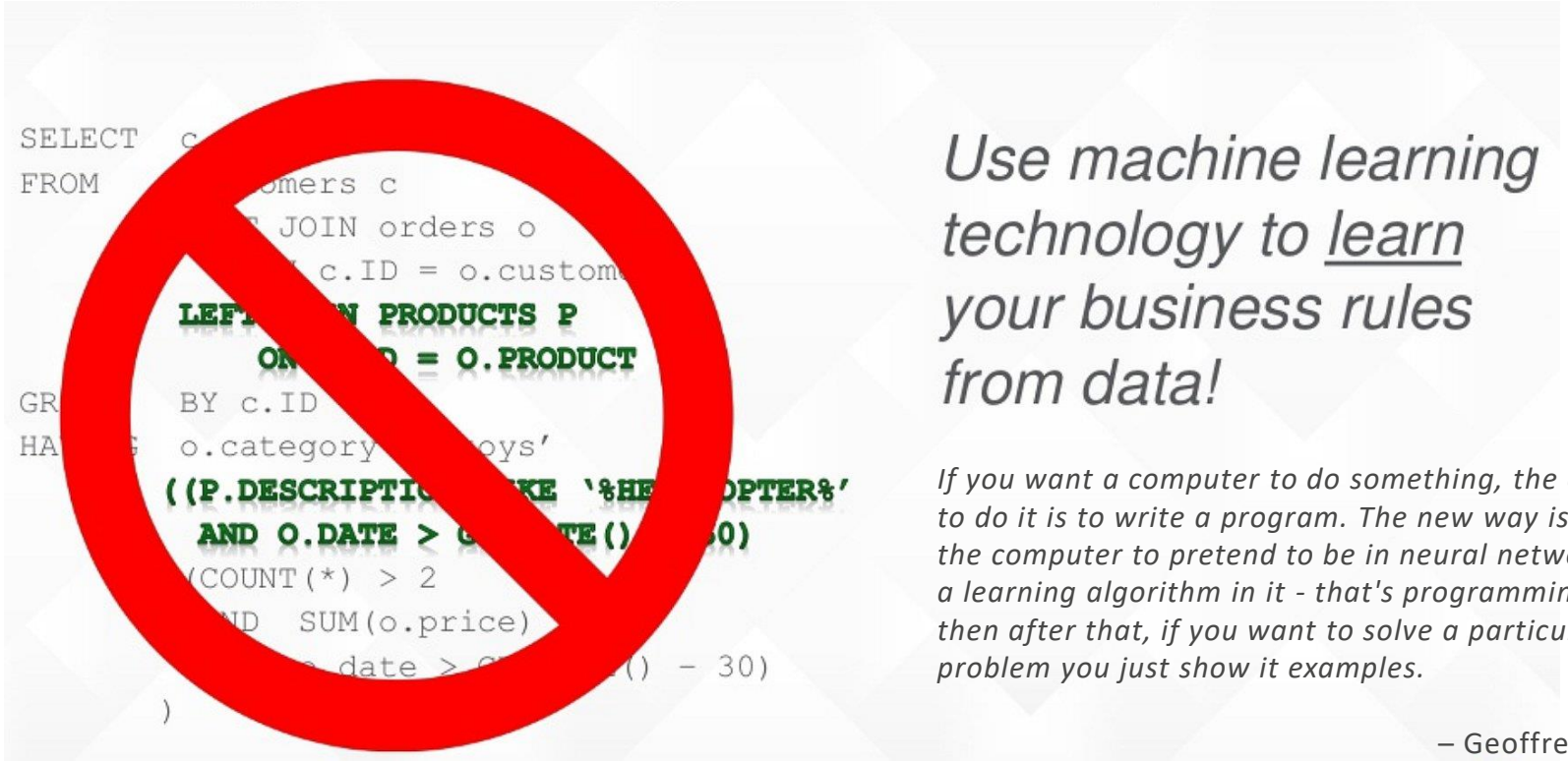
aws

INFERENCE

It’s where the magic happens!

1. Preprocess **new** data/image just like training set.
2. Feed image back to the trained model to get a predicted output.

THE OLD WAY VS. THE NEW WAY



Use machine learning technology to learn your business rules from data!

If you want a computer to do something, the old way to do it is to write a program. The new way is you tell the computer to pretend to be in neural network with a learning algorithm in it - that's programming, but then after that, if you want to solve a particular problem you just show it examples.

– Geoffrey Hinton

aws

4. AWS ML STACK: APPLICATION SERVICES

AMAZON REKOGNITION

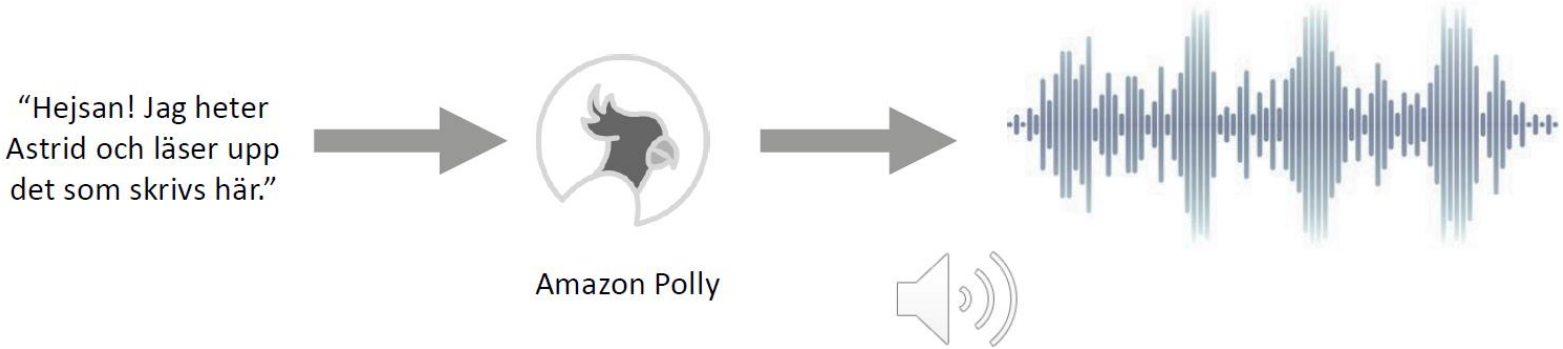
Amazon
Rekognition

Object and scene detection
Facial analysis
Face comparison
Celebrity recognition
Image moderation
Text in image

aws

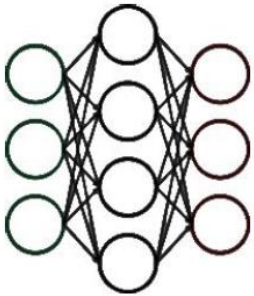
AMAZON REKOGNITION VIDEO

AMAZON POLLY



AMAZON TRANSLATE

Real-time translation



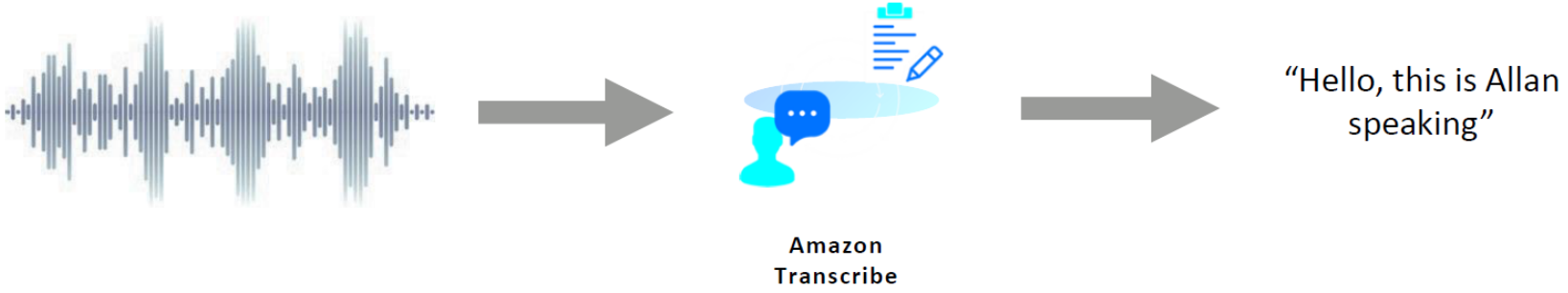
Powered by Deep Learning

12 Language pairs (more to come)

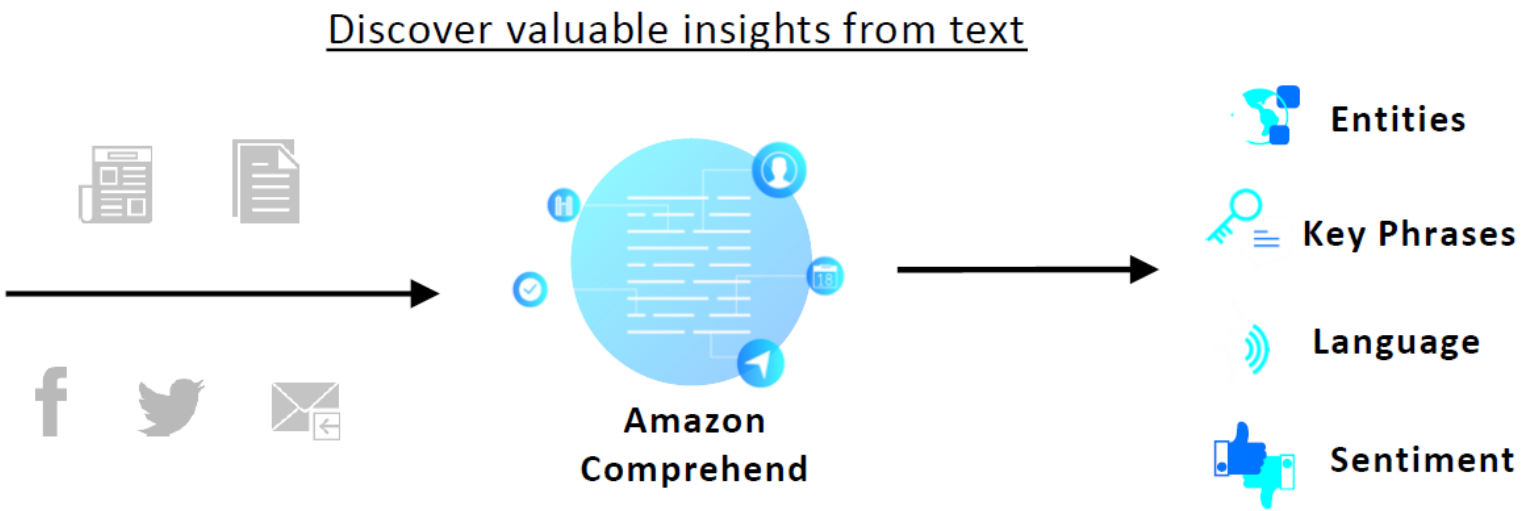
Language detection

aws

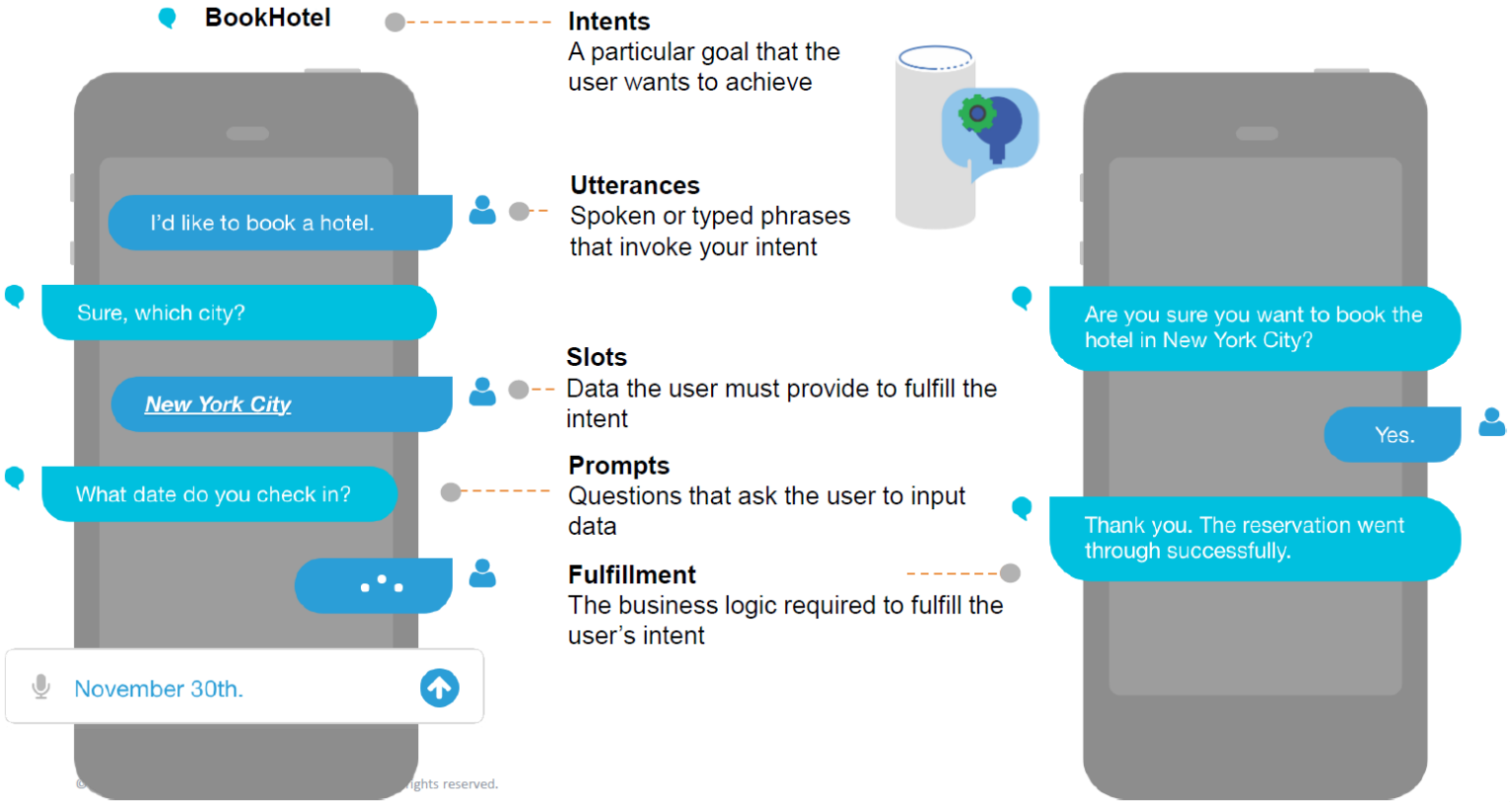
AMAZON TRANSCRIBE



AMAZON COMPREHEND



AMAZON LEX



Intents
A particular goal that the user wants to achieve

Utterances
Spoken or typed phrases that invoke your intent

Slots
Data the user must provide to fulfill the intent

Prompts
Questions that ask the user to input data

Fulfillment
The business logic required to fulfill the user's intent

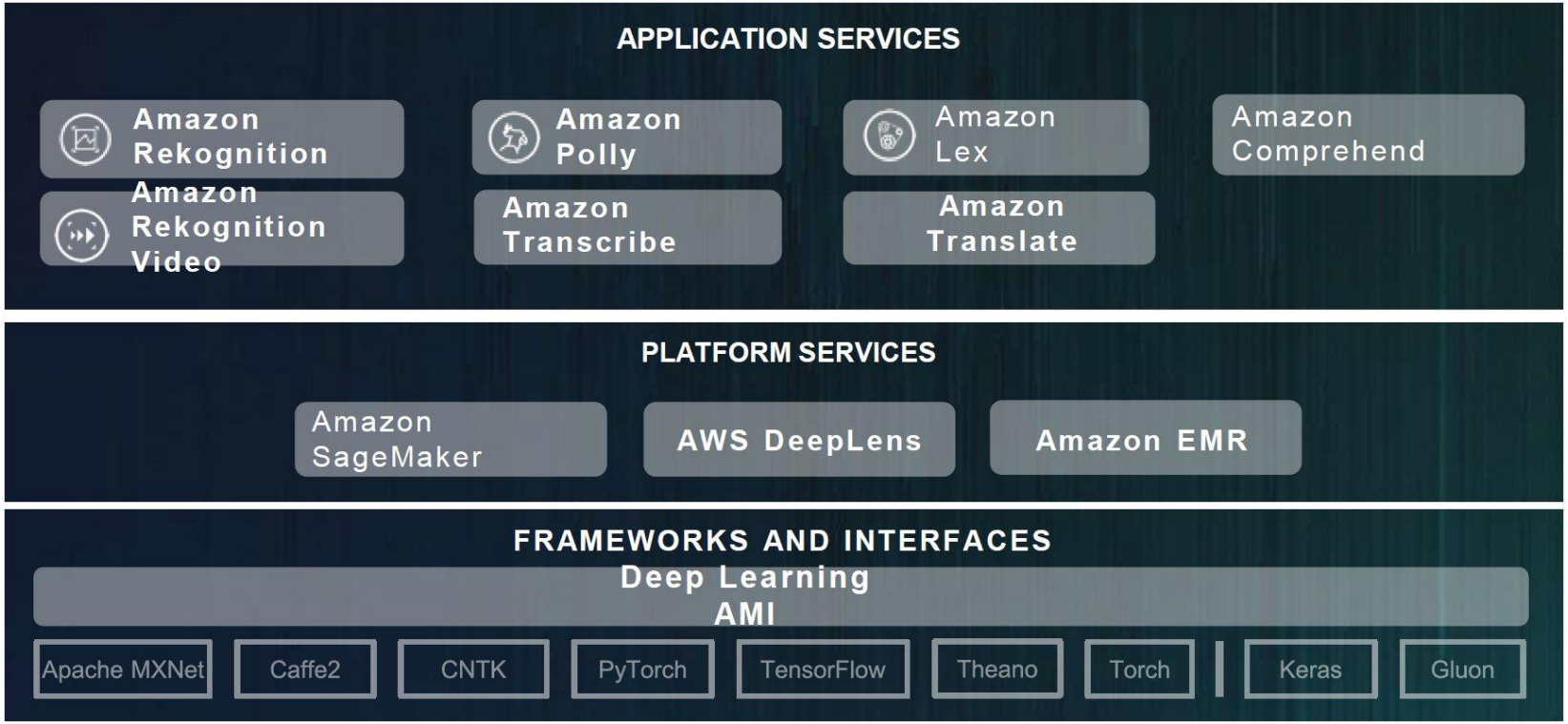
GO EXPERIMENT

"No Shame in Console Game"
– Abby Fuller

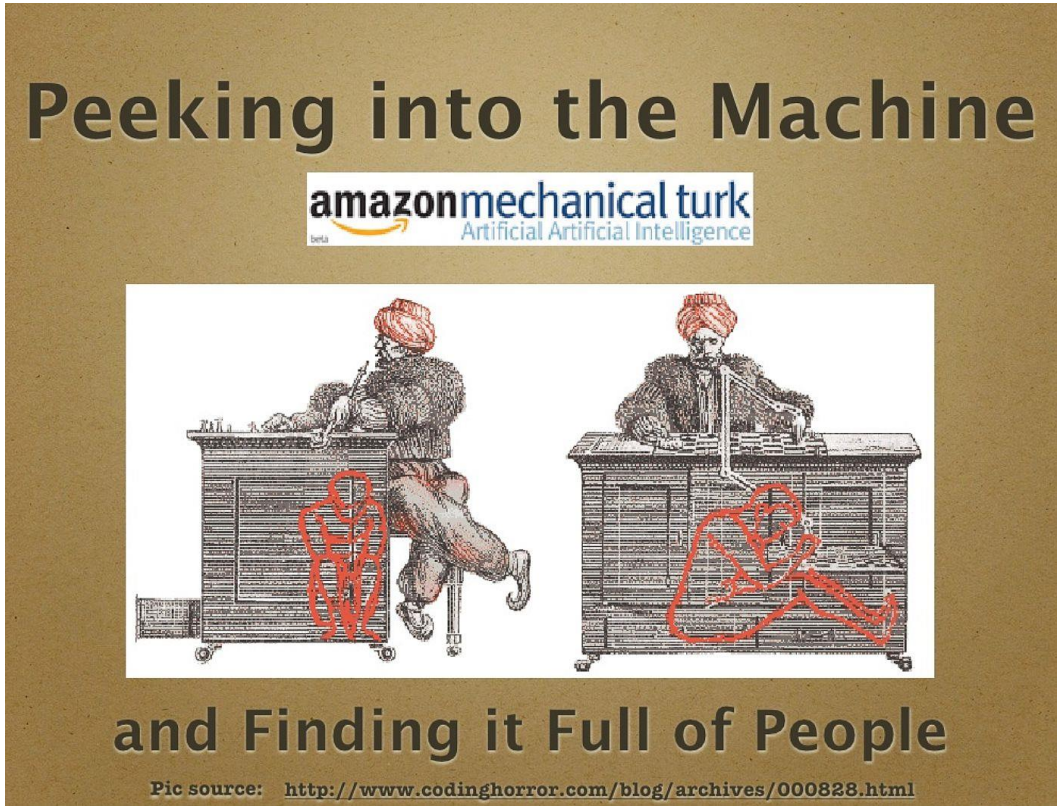
aws

4. AWS ML STACK: PLATFORM SERVICES

AWS ML STACK



PLUS: AMAZON MECHANICAL TURK



It is “artificial Artificial Intelligence” also counted as a Platform Service: https://docs.aws.amazon.com/mturk/

Crowdsourced labor directly into applications using an API to complete jobs that humans can do better than computers.

aws

AMAZON ELASTIC MAP REDUCE (EMR)

Amazon EMR

Spark

MLlib

The Machine Learning Library

Spark 1.2.2 Overview Programming Guides API Docs Deploying More

Machine Learning Library (MLlib) Programming Guide

MLlib is Spark's scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives, as outlined below:

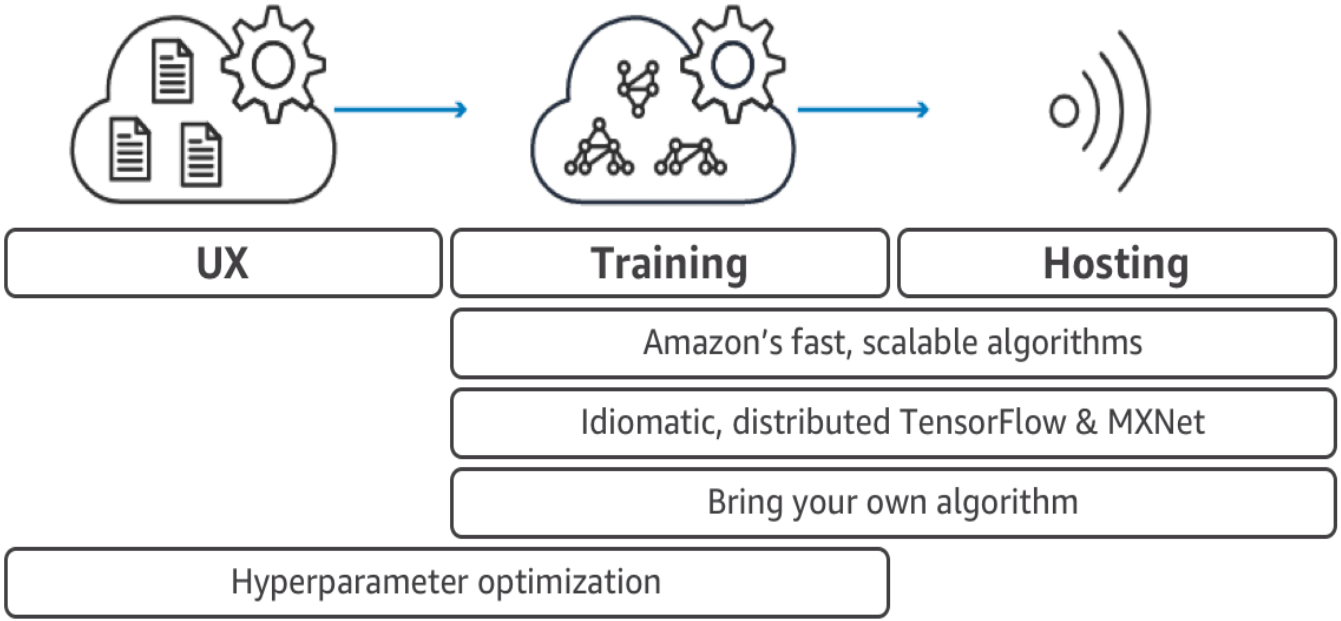
- Data types
- Basic statistics
  - summary statistics
  - correlations
  - stratified sampling
  - hypothesis testing
  - random data generation
- Classification and regression
  - linear models (SVMs, logistic regression, linear regression)
  - naive Bayes
  - decision trees
  - ensembles of trees (Random Forests and Gradient-Boosted Trees)
- Collaborative filtering
  - alternating least squares (ALS)
- Clustering
  - k-means
- Dimensionality reduction
  - singular value decomposition (SVD)
  - principal component analysis (PCA)
- Feature extraction and transformation
- Optimization (developer)
  - stochastic gradient descent
  - limited-memory BFGS (L-BFGS)

MLlib is under active development. The APIs marked `Experimental/DeveloperApi` may change in future releases, and the migration guide below will explain all changes between releases.

aws

SAGEMAKER – FULLY-MANAGED PLATFORM

The quickest and easiest way to get ML models from idea to production



End-to-End Machine Learning Platform

Zero Setup

Flexible Model Training

Pay By The Second

aws

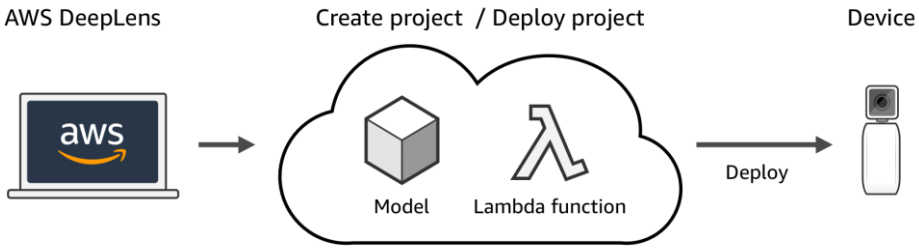
DEEPLENS

SPECIFICATIONS:

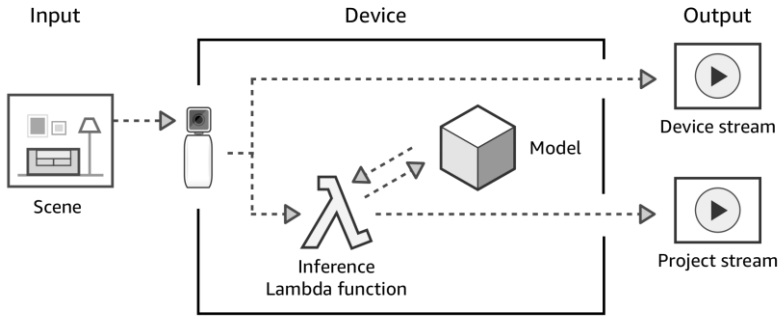
- Intel Atom Processor
- Gen9 graphics
- Ubuntu OS- 16.04 LTS
- 100 GFLOPS performance
- Dual band Wi-Fi
- 8 GB RAM
- 16 GB Storage (eMMC)
- 32 GB SD card
- 4 MP camera with MJPEG
- H.264 encoding at 1080p resolution
- 2 USB ports
- Micro HDMI
- Audio out
- AWS Greengrass preconfigured
- clDNN Optimized for MXNet

DEEPLENS – UNDER THE COVERS

- Cloud to device

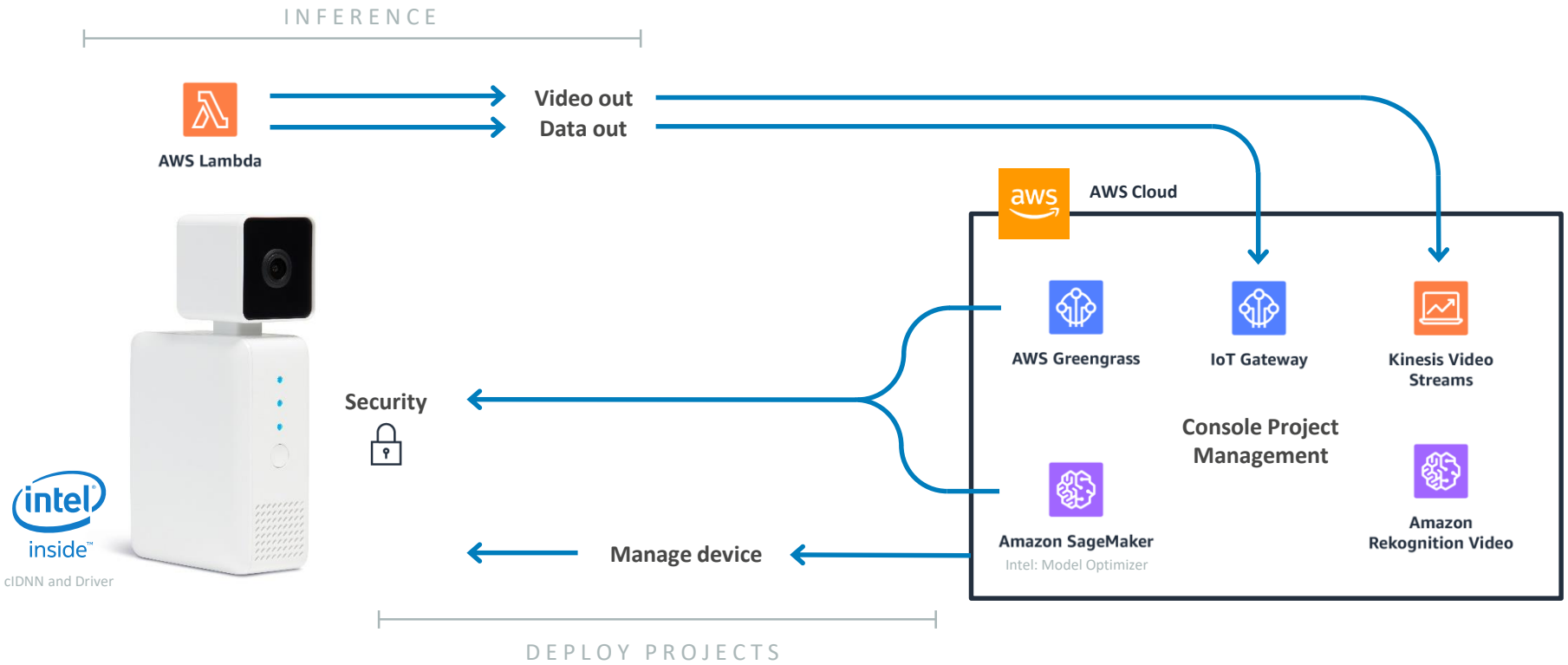


- On the device



aws

AWS DEEPLENS ARCHITECTURE



GET STARTED WITH SAMPLE PROJECTS

HOT DOG / NOT HOT DOG

OBJECT DETECTION

FACE DETECTION

ACTIVITY DETECTION

HEAD POSE DETECTION

ARTISTIC STYLE TRANSFER

CAT VS. DOG

Or build **custom** deep learning models in the cloud using

Amazon SageMaker

aws

ML BEHIND OBJECT DETECTION PROJECT

object-detection.py

```python
def greengrass_infinite_infer_run():
    """ Entry point of the lambda function"""
    try:
        # This object detection model is implemented as single shot detector (ssd), since
        # the number of labels is small we create a dictionary that will help us convert
        # the machine labels to human readable labels.
        model_type = 'ssd'
        output_map = {1: 'aeroplane', 2: 'bicycle', 3: 'bird', 4: 'boat', 5: 'bottle', 6: 'bus',
                      7 : 'car', 8 : 'cat', 9 : 'chair', 10 : 'cow', 11 : 'dinning table',
                      12 : 'dog', 13 : 'horse', 14 : 'motorbike', 15 : 'person',
                      16 : 'pottedplant', 17 : 'sheep', 18 : 'sofa', 19 : 'train',
                      20 : 'tvmonitor'}
        # Create an IoT client for sending to messages to the cloud.
        client = greengrasssdk.client('iot-data')
        iot_topic = '$aws/things/{}/infer'.format(os.environ['AWS_IOT_THING_NAME'])
        # Create a local display instance that will dump the image bytes to a FIFO
        # file that the image can be rendered locally.
        local_display = LocalDisplay('480p')
        local_display.start()
        # The sample projects come with optimized artifacts, hence only the artifact
        # path is required.
        model_path = '/opt/awscam/artifacts/mxnet_deploy_ssd_resnet50_300_FP16_FUSED.xml'
        # Load the model onto the GPU.
        client.publish(topic=iot_topic, payload='Loading object detection model')
        model = awscam.Model(model_path, {'GPU': 1})
        client.publish(topic=iot_topic, payload='Object detection model loaded')
        # Set the threshold for detection
        detection_threshold = 0.25
        # The height and width of the training set images
        input_height = 300
        input_width = 300
        # Do inference until the lambda is killed.
        while True:
            # Get a frame from the video stream
            ret, frame = awscam.getLastFrame()
            if not ret:
                raise Exception('Failed to get frame from the stream')
            # Resize frame to the same size as the training set.
            frame_resize = cv2.resize(frame, (input_height, input_width))
            # Run the images through the inference engine and parse the results using
            # the parser API, note it is possible to get the output of doInference
            # and do the parsing manually, but since it is a ssd model,
            # a simple API is provided.
            parsed_inference_results = model.parseResult(model_type,
                                                         model.doInference(frame_resize))
            # Compute the scale in order to draw bounding boxes on the full resolution
            # image.
            yscale = float(frame.shape[0]/input_height)
            xscale = float(frame.shape[1]/input_width)
```

300

object-detection.py

```python
            # Resize frame to the same size as the training set.
            frame_resize = cv2.resize(frame, (input_height, input_width))
            # Run the images through the inference engine and parse the results using
            # the parser API, note it is possible to get the output of doInference
            # and do the parsing manually, but since it is a ssd model,
            # a simple API is provided.
            parsed_inference_results = model.parseResult(model_type,
                                                         model.doInference(frame_resize))
            # Compute the scale in order to draw bounding boxes on the full resolution
            # image.
            yscale = float(frame.shape[0]/input_height)
            xscale = float(frame.shape[1]/input_width)
            # Dictionary to be filled with labels and probabilities for MQTT
            cloud_output = {}
            # Get the detected objects and probabilities
            for obj in parsed_inference_results[model_type]:
                if obj['prob'] > detection_threshold:
                    # Add bounding boxes to full resolution frame
                    xmin = int(xscale * obj['xmin']) \
                           + int((obj['xmin'] - input_width/2) + input_width/2)
                    ymin = int(yscale * obj['ymin'])
                    xmax = int(xscale * obj['xmax']) \
                           + int((obj['xmax'] - input_width/2) + input_width/2)
                    ymax = int(yscale * obj['ymax'])
                    # See https://docs.opencv.org/3.4.1/d6/d6e/group__imgproc__draw.html
                    # for more information about the cv2.rectangle method.
                    # Method signature: image, point1, point2, color, and tickness.
                    cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (255, 165, 20), 10)
                    # Amount to offset the label/probability text above the bounding box.
                    text_offset = 15
                    # See https://docs.opencv.org/3.4.1/d6/d6e/group__imgproc__draw.html
                    # for more information about the cv2.putText method.
                    # Method signature: image, text, origin, font face, font scale, color,
                    # and tickness
                    cv2.putText(frame, "{}: {:.2f}%".format(output_map[obj['label']],
                                                            obj['prob'] * 100),
                                (xmin, ymin-text_offset),
                                cv2.FONT_HERSHEY_SIMPLEX, 2.5, (255, 165, 20), 6)
                    # Store label and probability to send to cloud
                    cloud_output[output_map[obj['label']]] = obj['prob']
            # Set the next frame in the local display stream.
            local_display.set_frame_data(frame)
            # Send results to the cloud
            client.publish(topic=iot_topic, payload=json.dumps(cloud_output))
        except Exception as ex:
            client.publish(topic=iot_topic, payload='Error in object detection lambda: {}'.format(ex))

greengrass_infinite_infer_run()
```


aws

REUSABLE CODE

object-detection.py

```
        while True:
            # Get a frame from the video stream
            ret, frame = awscam.getLastFrame()
            if not ret:
                raise Exception('Failed to get frame from the stream')
            # Resize frame to the same size as the training set.
            frame_resize = cv2.resize(frame, (input_height, input_width))
            # Run the images through the inference engine and parse the results using
            # the parser API, note it is possible to get the output of doInference
            # and do the parsing manually, but since it is a ssd model,
            # a simple API is provided.
            parsed_inference_results = model.parseResult(model_type,
                                                         model.doInference(frame_resize
            # Compute the scale in order to draw bounding boxes on the full resolution
            # image.
            yscale = float(frame.shape[0]/input_height)
            xscale = float(frame.shape[1]/input_width)
            # Dictionary to be filled with labels and probabilities for MQTT
            cloud_output = {}
            # Get the detected objects and probabilities
            for obj in parsed_inference_results[model_type]:
                if obj['prob'] > detection_threshold:
                    # Add bounding boxes to full resolution frame
                    xmin = int(xscale * obj['xmin']) \
                           + int((obj['xmin'] - input_width/2) + input_width/2)
                    ymin = int(yscale * obj['ymin'])
                    xmax = int(xscale * obj['xmax']) \
                           + int((obj['xmax'] - input_width/2) + input_width/2)
                    ymax = int(yscale * obj['ymax'])
                    # See https://docs.opencv.org/3.4.1/d6/d6e/group__imgproc__draw.htm
                    # for more information about the cv2.rectangle method.
                    # Method signature: image, point1, point2, color, and tickness.
                    cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (255, 165, 20), 10
                    # Amount to offset the label/probability text above the bounding bo
                    text_offset = 15
                    # See https://docs.opencv.org/3.4.1/d6/d6e/group__imgproc__draw.htm
                    # for more information about the cv2.putText method.
                    # Method signature: image, text, origin, font face, font scale, col
                    # and tickness
                    cv2.putText(frame, "{}: {:.2f}%".format(output_map[obj['label']],
                                                            obj['prob'] * 100),
                                (xmin, ymin-text_offset),
                                cv2.FONT_HERSHEY_SIMPLEX, 2.5, (255, 165, 20), 6)
                    # Store label and probability to send to cloud
                    cloud_output[output_map[obj['label']]] = obj['prob']
            # Set the next frame in the local display stream.
            local_display.set_frame_data(frame)
            # Send results to the cloud
            client.publish(topic=iot_topic, payload=json.dumps(cloud_output))
    except Exception as ex:
```

hotdog-detection.py

```
        # Since this is a binary classifier only retrieve 2 classes.
        num_top_k = 2
        # The height and width of the training set images
        input_height = 224
        input_width = 224
        # Do inference until the lambda is killed.
        while True:
            # Get a frame from the video stream
            ret, frame = awscam.getLastFrame()
            if not ret:
                raise Exception('Failed to get frame from the stream')
            # Resize frame to the same size as the training set.
            frame_resize = cv2.resize(frame, (input_height, input_width))
            # Run the images through the inference engine and parse the results using
            # the parser API, note it is possible to get the output of doInference
            # and do the parsing manually, but since it is a classification model,
            # a simple API is provided.
            parsed_inference_results = model.parseResult(model_type,
                                                         model.doInference(frame_resize))
            # Get top k results with highest probabilities
            top_k = parsed_inference_results[model_type][0:num_top_k-1]
            # Get the probability of 'hotdog' label, which corresponds to label 934 in SqueezeNet.
            prob_hotdog = 0.0
            for obj in top_k:
                if obj['label'] == 934:
                    prob_hotdog = obj['prob']
                    break
            # Compute the probability of a hotdog not being in the image.
            prob_not_hotdog = 1.0 - prob_hotdog
            # Add two bars to indicate the probability of a hotdog being present and
            # the probability of a hotdog not being present.
            # See https://docs.opencv.org/3.4.1/d6/d6e/group__imgproc__draw.html
            # for more information about the cv2.rectangle method.
            # Method signature: image, point1, point2, color, and tickness.
            cv2.rectangle(frame, (0, 0), (int(frame.shape[1] * 0.2 * prob_not_hotdog), 80),
                          (0, 0, 255), -1)
            cv2.rectangle(frame, (0, 90), (int(frame.shape[1] * 0.2 * prob_hotdog), 170), (0, 255, 0), -1)
            font = cv2.FONT_HERSHEY_SIMPLEX
            # See https://docs.opencv.org/3.4.1/d6/d6e/group__imgproc__draw.html
            # for more information about the cv2.putText method.
            # Method signature: image, text, origin, font face, font scale, color,
            # and tickness
            cv2.putText(frame, 'Not hotdog', (10, 70), font, 3, (225, 225, 225), 8)
            cv2.putText(frame, 'Hotdog', (10, 160), font, 3, (225, 225, 225), 8)
            # Send the top k results to the IoT console via MQTT
            client.publish(topic=iot_topic, payload=json.dumps({'Hotdog': prob_hotdog,
                                                                'Not hotdog': prob_not_hotdog}))
            # Set the next frame in the local display stream.
            local_display.set_frame_data(frame)
    except Exception as ex:
```

SEE WHAT OTHERS HAVE BUILT

VIEW ALL 23 PROJECTS: **aws.amazon.com/deeplens/community-projects**

AWS DEEPLENS HACKATHON WINNERS:

First Place

ReadToMe
Created by Alex Schultz

ReadToMe is a deep learning enabled application that is able to read books to kids. In this case, reading Green Eggs and Ham, by Dr. Seuss.

Second Place

Dee
Created by Matthew Clark

Dee is a fun AWS DeepLens interactive device for children. The device asks children to answer questions by showing a picture of the answer.

Third Place

SafeHaven
Created by Nathan Stone and Peter McLean

SafeHaven uses Alexa and AWS DeepLens to bring peace of mind for vulnerable people and their families.

SUBMITTED TO

AWS DeepLens Challenge

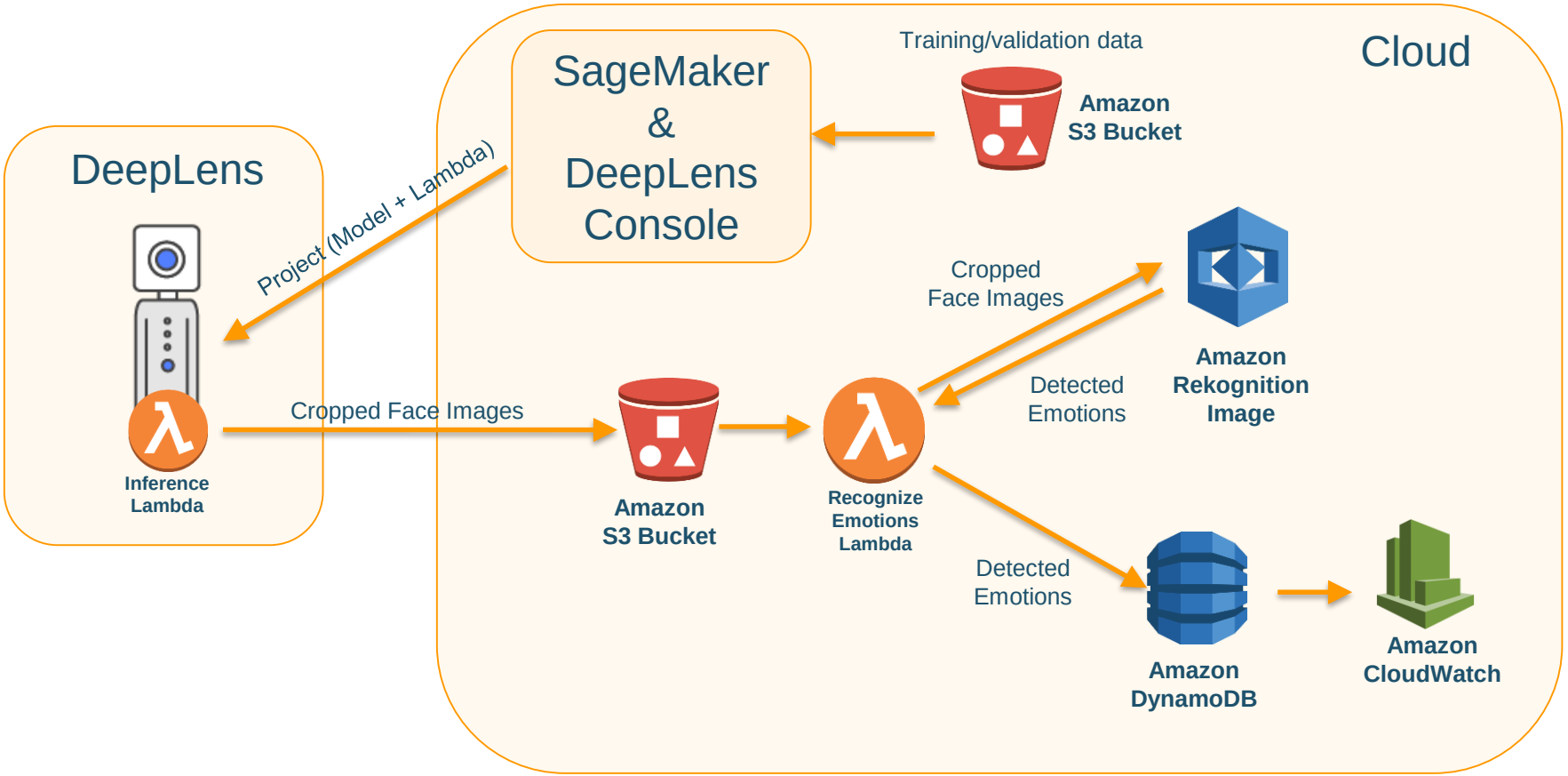
CREATED BY

Ricardo Mota

Jidesh Veeramachaneni

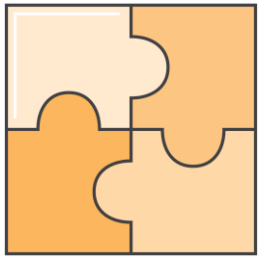
Report inappropriate content

Watch the demo:
https://youtu.be/2oMGkWc46Lw



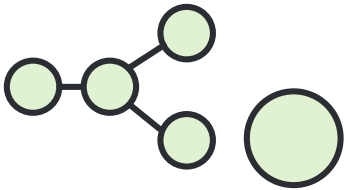
AMAZON SAGEMAKER

The quickest and easiest way to get ML models from idea to production

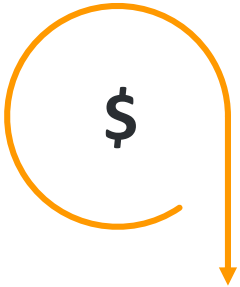


End-to-End Machine Learning Platform

Zero Setup



Flexible Model Training



Pay By The Second

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

5. KEEP LEARNING

CHOOSE YOUR MENTORS WISELY

- Unlike 25+ years ago when there was a lack of information, Internet overwhelms us with information. New problem is finding good mentors (not just those who sound good) – because nowadays everyone writes, records videos, and brags.

- **Join local communities!** (“Collaborative Filtering” the old fashioned way ;)

- Follow the links in my curated list of **awesome** people and resources where to go for knowledge next…

And let’s make the world a better place through AI & ML!

aws

DEEPLENS & SAGEMAKER

- **Julien Simon**, AI & Machine Learning @ AWS https://medium.com/@julsimon
  - Deep Learning at the edge with AWS DeepLens - Julien Simon (AWS) https://www.youtube.com/watch?v=Vw-bX1RRZGQ
  - An overview of Amazon SageMaker https://www.youtube.com/watch?v=ym7NEYEx9x4
- Dev Guide https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-dg.pdf
- **Lynn Langit**: "Amazon Web Services Machine Learning Essential Training" 3hr course on Lynda.com or https://www.linkedin.com/learning/amazon-web-services-machine-learning-essential-training

- **Randall Hunt**, Sr Technical Evangelist @ AWS https://aws.amazon.com/blogs/aws/author/randhunt/

ALEXA & VOICE-FIRST AI

- **Paul Cutsinger**, Head of Alexa Voice Design Education @ AWS https://twitter.com/PaulCutsinger
- Alexa Blogs https://developer.amazon.com/blogs/alexa/
- ML uses case "Chatbots & Voice User Interfaces (the Amazon way)" – samples at http://goo.gl/H5CEpW slides at https://www.slideshare.net/dzivkovi/3in1-talk-on-serverless-chatbots-alexa-skills-voice-ui-best-practices-the-amazon-way

aws

ML & DEEP LEARNING

- **Robert Newman** @ Beyondsoft “How to Leverage Machine Learning” https://www.brighttalk.com/webcast/16661/321515/how-to-leverage-machine-learning

- **Geoffrey Hinton** “The Foundations of Deep Learning” https://www.youtube.com/watch?v=zl99lZvW7rE
- Deep Learning **Crash Course** https://gluon.mxnet.io/chapter01_crashcourse/introduction.html
- **3Blue1Brown** playlist “Neural Networks” https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

- **Jeremy Howard** https://en.wikipedia.org/wiki/Jeremy_Howard_(entrepreneur)
  - Preview our Deep Learning Certificate with a lecture by Jeremy Howard https://www.usfca.edu/data-institute/certificates/deep-learning-part-one
  - The wonderful and **terrifying** implications of computers that can learn https://www.youtube.com/watch?v=t4kyRyKyOpo
  - Practical Deep Learning for Coders http://course.**fast.ai**/start.html (2018 edition of 7-week course)

- **UN Sustainable Development Goals** https://www.globalgoals.org/ (17 goals for a better world by 2030)
- AI grounded in human values, empowering people to live better lives https://www.PartnershipOnAI.org
- Make the world a better place through AI https://ai4good.org/

aws