SPARK+AI
SUMMIT EUROPE

ML At the Edge: Building Your Production Pipeline With Apache Spark and Tensorflow

Stavros Kontopoulos, Lightbend

#SAISML9

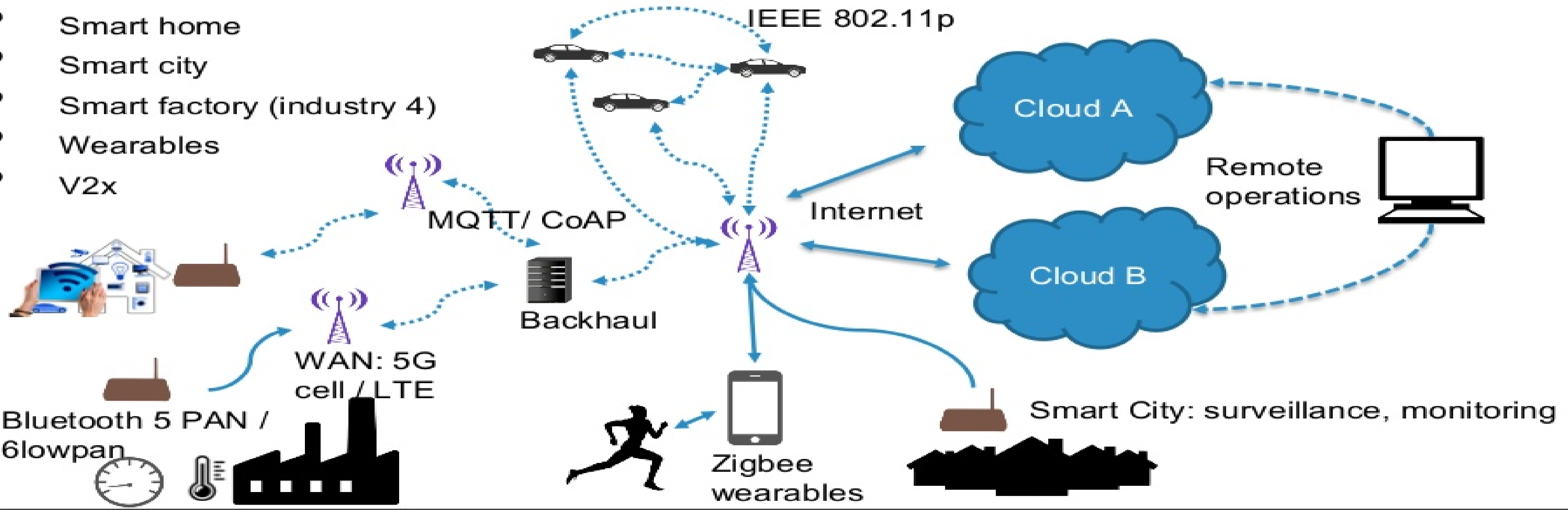
Agenda

- Edge computing today – a review
- Machine learning at the edge
- ML model training/serving techniques with TF, Spark etc.
- Production Pipeline Architecture

https://bit.ly/2xlgKCH #s_kontopoulos

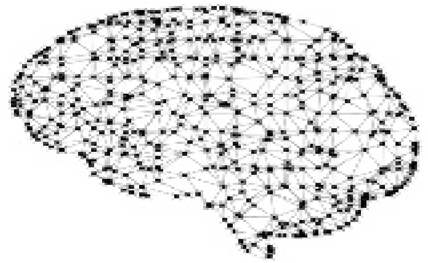
Internet of Everything

- Smart home
- Smart city
- Smart factory (industry 4)
- Wearables
- V2x

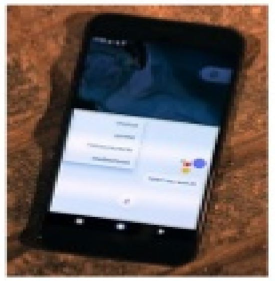


Smart is about…

- Moving from sensing to a system of cyber-physical systems.
- Virtual and physical indicators and actuators modify the physical and virtual environment (autonomy).
- Machine Learning is an enabler for this.



Other cases



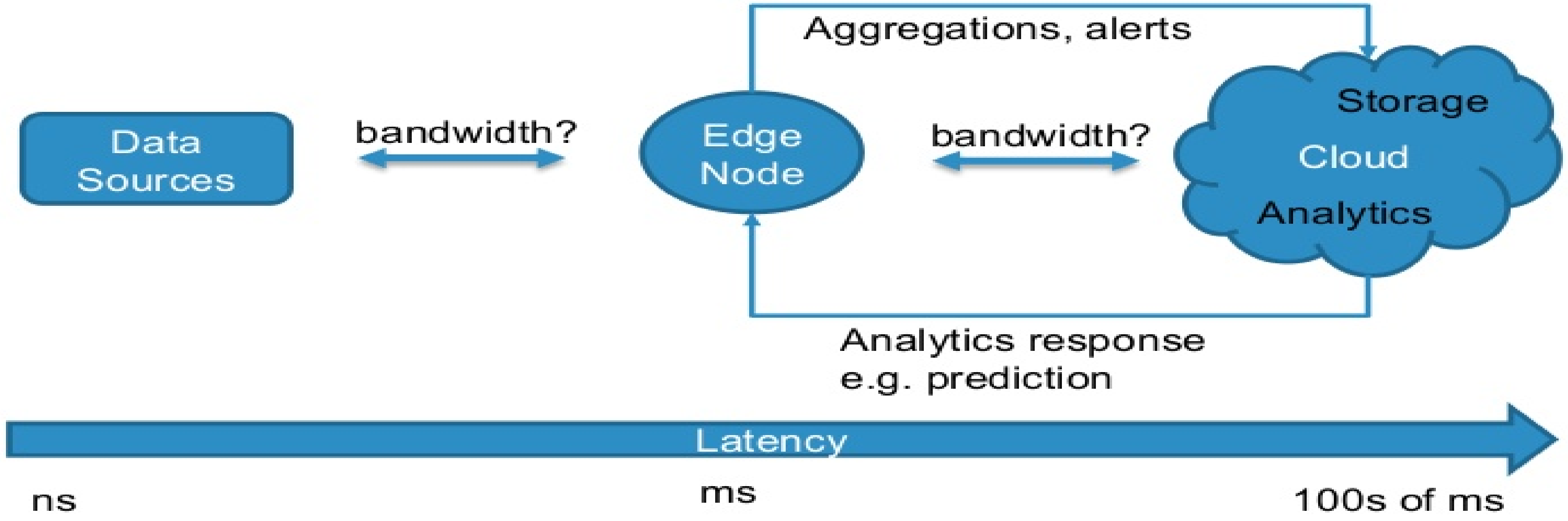
Improving technology:

- Smart mobile apps
- Intelligent drones
- Agents/personal assistants

Improving the Industry:

- Energy: smart meters, status prediction
- E-Health: patient monitoring
- Logistics: asset tracking
- Distributed business management: e.g. chain of stores

End-to-End Pipeline



Typical Requirements

- How fast I respond?
  - A credit card transaction for example needs to finish in several ms.
- What bandwidth do I have?
  - Security cams deliver around 100GB/day/camera. A connected car, several TBs.

Edge Computing

- Definition: “Optimizing applications by moving parts of computation at the edge nodes where physical world lies. Does not require a cloud backend service.”
- Fog Computing?
  - Consortium: https://www.openfogconsortium.org

Edge Computing - Processing

- Data analytics
  - Preprocessing (filtering out, transformations, aggregations, approximate counting etc)
  - Alerting
- Machine Learning
  - Model Serving
  - Model training (less common)

Edge Computing - Benefits

- Reduced latency for real-time applications
- No single point of failure (eg. cloud infrastructure)
- Efficient use of resources (bandwidth, storage)
- Better privacy control
- Reduction in costs and energy consumption

Edge Computing - Challenges

- Limited resources.
  - Edge nodes cannot perform billions of floating points operations to train a very complex deep learning model. They cannot store TBs of data etc.
- Edge node interconnection. Information sharing.

Machine Learning At The Edge

Criteria for moving ML computation to the Edge:

- Data size
- Computational power
- Specialized hardware

Interesting scenarios:

- Per edge node ML model e.g. personal ML
- Distributed Local Learning
- Transfer Learning

Machine Learning At The Edge

- Different types of models required per use case.
  - Manufacturing: Bayesian networks for fault diagnosis, Uncertainty Quantification for performance metrics. RNNs for signal analysis, fault prediction.
  - HealthCare: Decision trees, rule engines for health prediction etc.

Machine Learning At The Edge

- Training:
  - Random Forests (usually uses bagging, low resources, mainly supervised learning)
  - RNNs, CNNs difficult to train, mainly for inference
- Serving:
  - Resource restrictions impose model compression
  - Scoring may be problematic if requires intensive computation

Machine Learning At The Edge

Drift Detection / Model Statistics examples:

- Use biased reservoir sampling for model re-training due to space constraints
- Sample data changes and feed change means to a drift detector like ADWIN.

Knowledge Discovery from Data Streams (Joao Gama, https://dl.acm.org/citation.cfm?id=1855075)

Tracking Drift Types in Changing Data Streams (https://rd.springer.com/chapter/10.1007/978-3-642-53914-5_7)

Concept Drift: Monitoring Model Quality in Streaming ML Applications (https://info.lightbend.com/webinar-concept-drift-monitoring-model-quality-in-streaming-ml-apps-recording.html)

Model Size - Compression

- How can we deploy our models to edge nodes?
- Important for NNs. Several techniques like:
  - Parameter pruning and sharing
  - Quantization
  - Knowledge transfer (student teacher paradigm)

Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding (https://arxiv.org/abs/1510.00149)

A Survey of Model Compression and Acceleration for Deep Neural Networks (https://arxiv.org/abs/1710.09282)

Model Formats

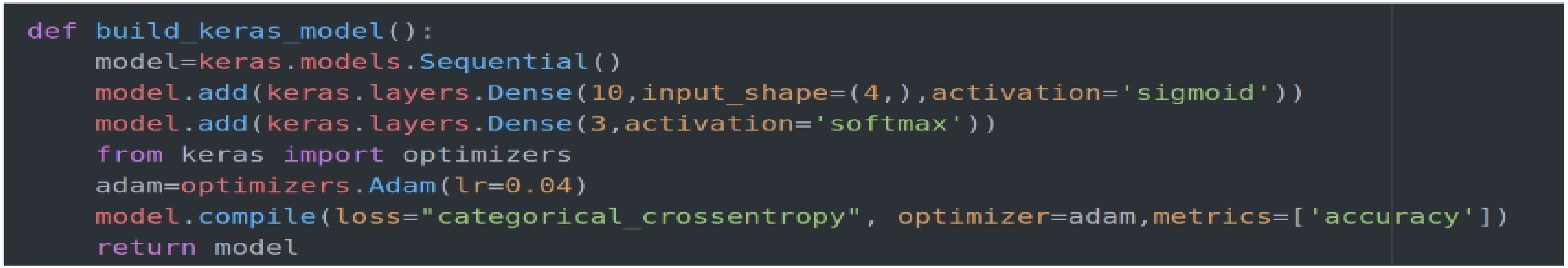
- Generic
  - PMML/ PFA (also for backend services)
- Optimized for the edge
  - CoreML, TFLite, Google ML kit (Learn2Compress).

Training/Serving Techniques at the Edge and more…

- Tensorflow flavors: TFlite, TF.js
- Smile for the jvm
- Performance Evaluation
- Spark for distributed training etc

Keras/Tensorflow

Build the model (Iris dataset):



```python
def build_keras_model():
    model=keras.models.Sequential()
    model.add(keras.layers.Dense(10,input_shape=(4,),activation='sigmoid'))
    model.add(keras.layers.Dense(3,activation='softmax'))
    from keras import optimizers
    adam=optimizers.Adam(lr=0.04)
    model.compile(loss="categorical_crossentropy", optimizer=adam,metrics=['accuracy'])
    return model
```

Keras to TF

```python
import tensorflow as tf
tfm = tf.keras.models.load_model('./keras_iris.h5')
with tf.keras.backend.get_session() as sess:
    sess.run(tf.global_variables_initializer())
    tf.saved_model.simple_save(
        sess,
        './irisModel/1',
        inputs={'input_image': tfm.input},
        outputs={t.name:t for t in tfm.outputs})
```

```
saved_model_cli run --dir ./ --tag_set serve --signature_def serving_default --input_exp 'input_image=np.array([ [5.5, 4.2, 1.4, 0.2]])'
```

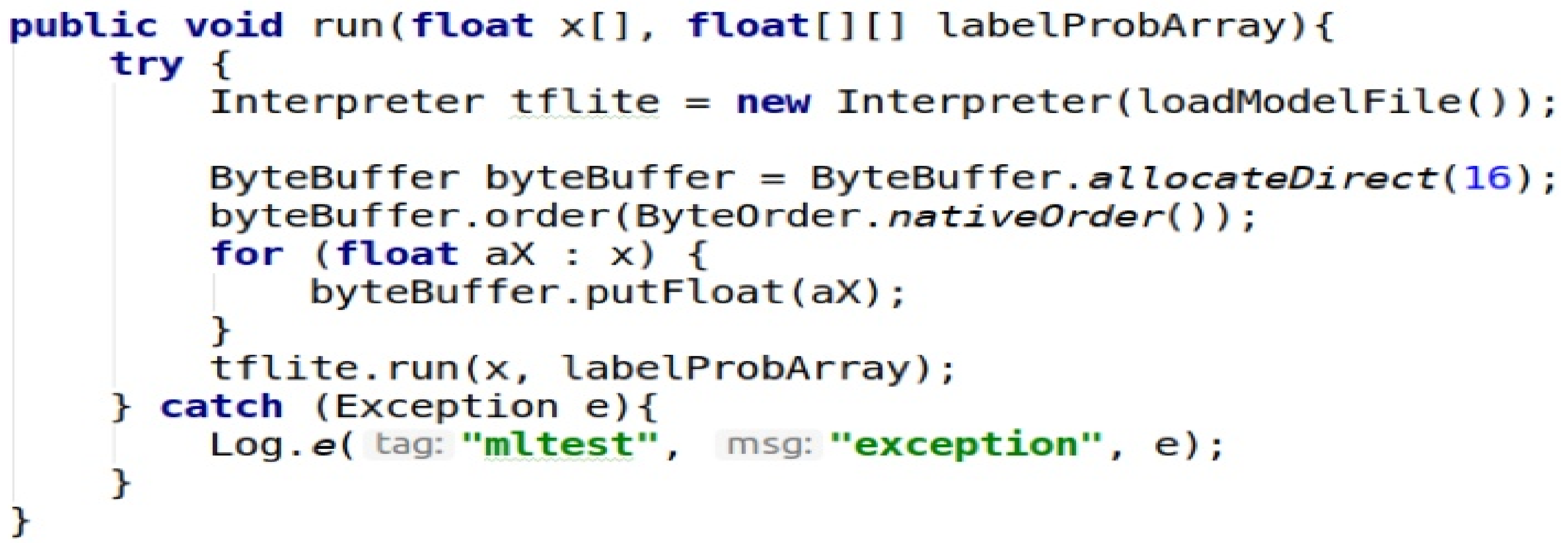
Keras to TFLite

```
# Save tf.keras model in HDF5 format.
keras_file = "keras_iris.h5"
tf.keras.models.save_model(model, keras_file)

# Convert to TensorFlow Lite model.
converter = tf.contrib.lite.TocoConverter.from_keras_model_file(keras_file)
tflite_model = converter.convert()
open("iris.tflite", "wb").write(tflite_model)
```

Tip: On android make sure the model is not stored compressed in your assets.

TFLite on Android



```java
public void run(float x[], float[][] labelProbArray){
    try {
        Interpreter tflite = new Interpreter(loadModelFile());

        ByteBuffer byteBuffer = ByteBuffer.allocateDirect(16);
        byteBuffer.order(ByteOrder.nativeOrder());
        for (float aX : x) {
            byteBuffer.putFloat(aX);
        }
        tflite.run(x, labelProbArray);
    } catch (Exception e){
        Log.e( tag: "mltest",  msg: "exception", e);
    }
}
```

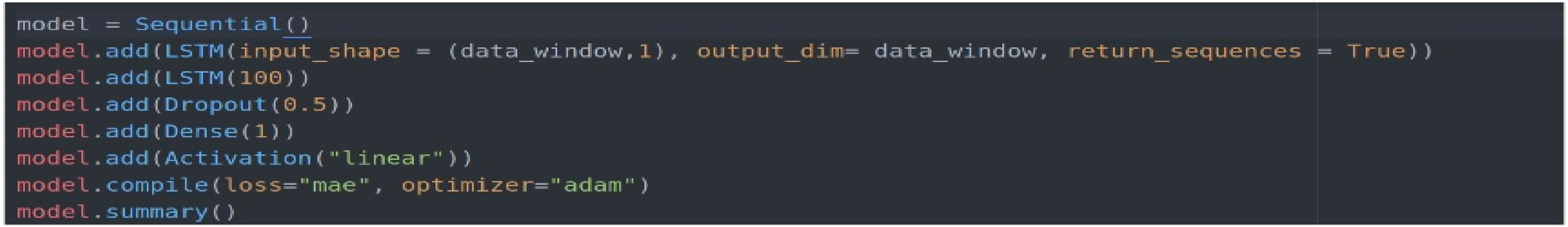
TFLite

- Uses Quantization and other techniques for model compression.
- Does not support training on the mobile device:
  - Training on device: https://github.com/tensorflow/tensorflow/issues/17328
  - It could be possible to train a model offline, and re-train it on the device (C++ API).

TFLite

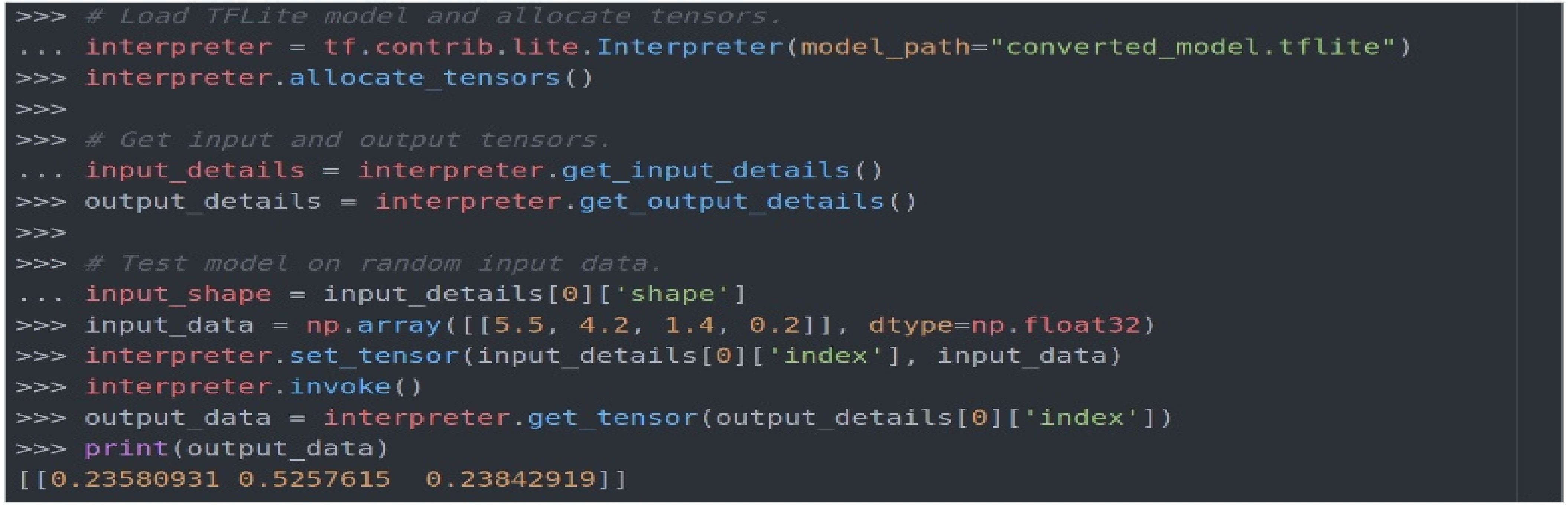
- LSTM (RNNs)?

  No out of the box: https://github.com/tensorflow/tensorflow/issues/15805



```
model = Sequential()
model.add(LSTM(input_shape = (data_window,1), output_dim= data_window, return_sequences = True))
model.add(LSTM(100))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation("linear"))
model.compile(loss="mae", optimizer="adam")
model.summary()
```

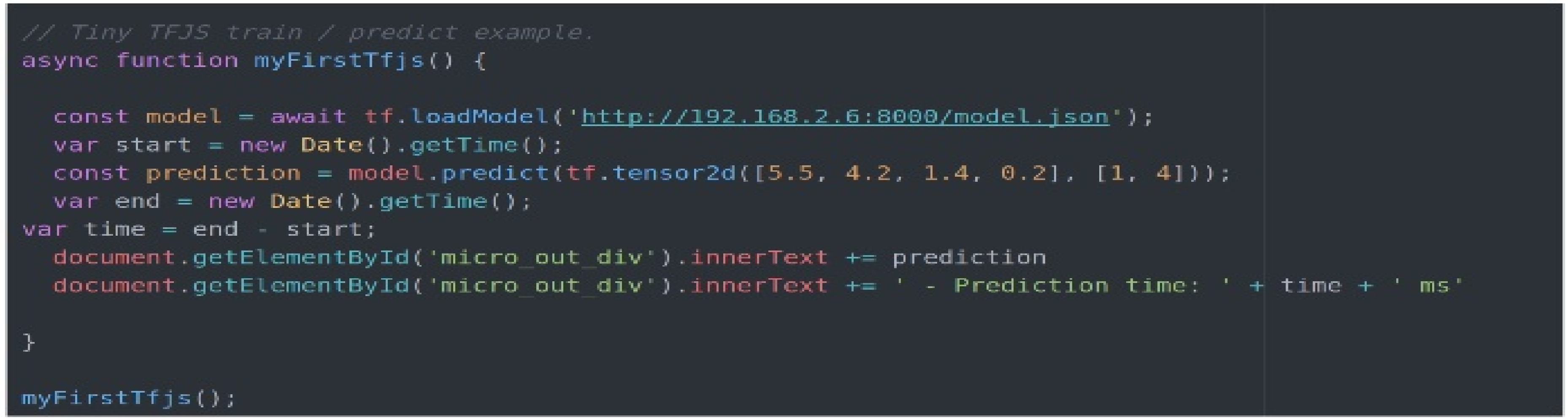
TFLite – Local test



```
>>> # Load TFLite model and allocate tensors.
... interpreter = tf.contrib.lite.Interpreter(model_path="converted_model.tflite")
>>> interpreter.allocate_tensors()
>>>
>>> # Get input and output tensors.
... input_details = interpreter.get_input_details()
>>> output_details = interpreter.get_output_details()
>>>
>>> # Test model on random input data.
... input_shape = input_details[0]['shape']
>>> input_data = np.array([[5.5, 4.2, 1.4, 0.2]], dtype=np.float32)
>>> interpreter.set_tensor(input_details[0]['index'], input_data)
>>> interpreter.invoke()
>>> output_data = interpreter.get_tensor(output_details[0]['index'])
>>> print(output_data)
[[0.23580931 0.5257615  0.23842919]]
```

Tensorflow.JS

Easy to use for both training and serving
(A WebGL accelerated framework).



```
// Tiny TFJS train / predict example.
async function myFirstTfjs() {

  const model = await tf.loadModel('http://192.168.2.6:8000/model.json');
  var start = new Date().getTime();
  const prediction = model.predict(tf.tensor2d([5.5, 4.2, 1.4, 0.2], [1, 4]));
  var end = new Date().getTime();
var time = end - start;
  document.getElementById('micro_out_div').innerText += prediction
  document.getElementById('micro_out_div').innerText += ' - Prediction time: ' + time + ' ms'

}

myFirstTfjs();
```

GR COSMOTE 100% 8:34

192.168.2.6:8000

Tiny TFJS example

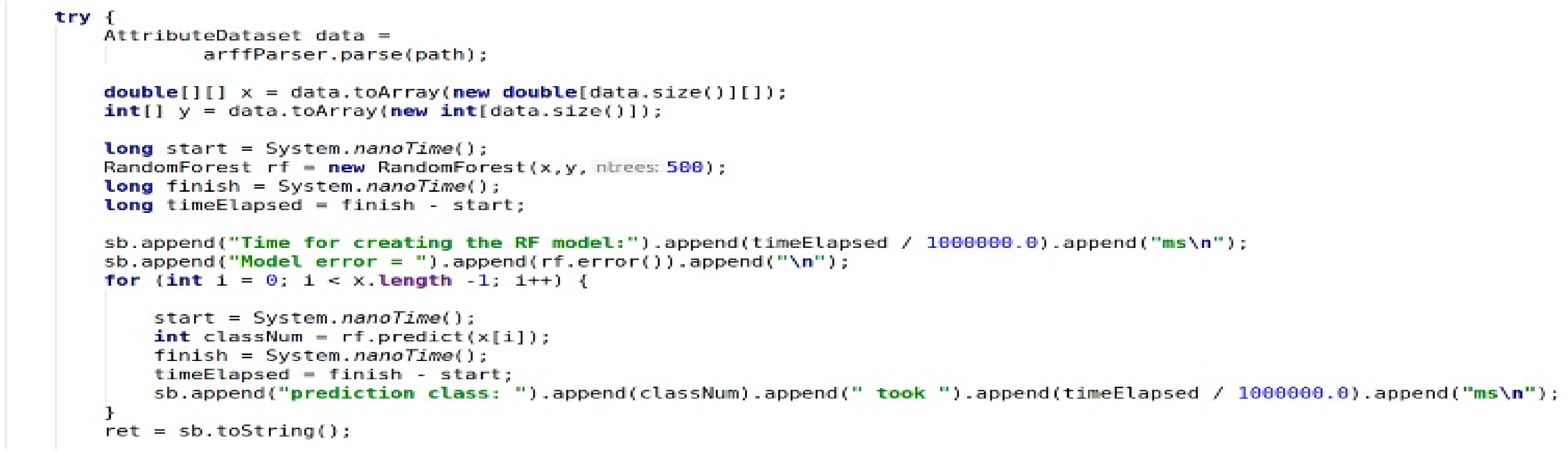
Tensor
[illegible] - Prediction time: [illegible] ms

Tensorflow.JS – Tips

- Run a local server for quick testing
  - eg. python –m SimpleHTTPServer
- Chrome developer tools -> remote devices is your friend for debugging on the mobile browser and check js errors.
- Check your browser support for js.

Smile On Android

Machine Learning Lib (https://haifengl.github.io/smile) with Scala, Java support. Easy to use. Uses Xstream for serialization, models can be used by Spark for scoring.



```
try {
    AttributeDataset data =
            arffParser.parse(path);

    double[][] x = data.toArray(new double[data.size()][]);
    int[] y = data.toArray(new int[data.size()]);

    long start = System.nanoTime();
    RandomForest rf = new RandomForest(x,y, ntrees: 500);
    long finish = System.nanoTime();
    long timeElapsed = finish - start;

    sb.append("Time for creating the RF model:").append(timeElapsed / 1000000.0).append("ms\n");
    sb.append("Model error = ").append(rf.error()).append("\n");
    for (int i = 0; i < x.length -1; i++) {

        start = System.nanoTime();
        int classNum = rf.predict(x[i]);
        finish = System.nanoTime();
        timeElapsed = finish - start;
        sb.append("prediction class: ").append(classNum).append(" took ").append(timeElapsed / 1000000.0).append("ms\n");
    }
    ret = sb.toString();
```

Performance Evaluation

- Relative performance matters more than absolute numbers here, but it depends on the SLA for a real scenario.
- Average prediction time for TFlite for the minimal example (NN) ~15ms.
- TF.js prediction time around 300ms for the same NN model.
  - On laptop: ~100ms
- Average prediction time for RF with Smile: ~1ms.
  - Model construction 1s.
  - On laptop: 0.1ms prediction time and model build 100ms.
- Tests on Huawei P20. Results vary with the device!

Spark for Model Training/Serving

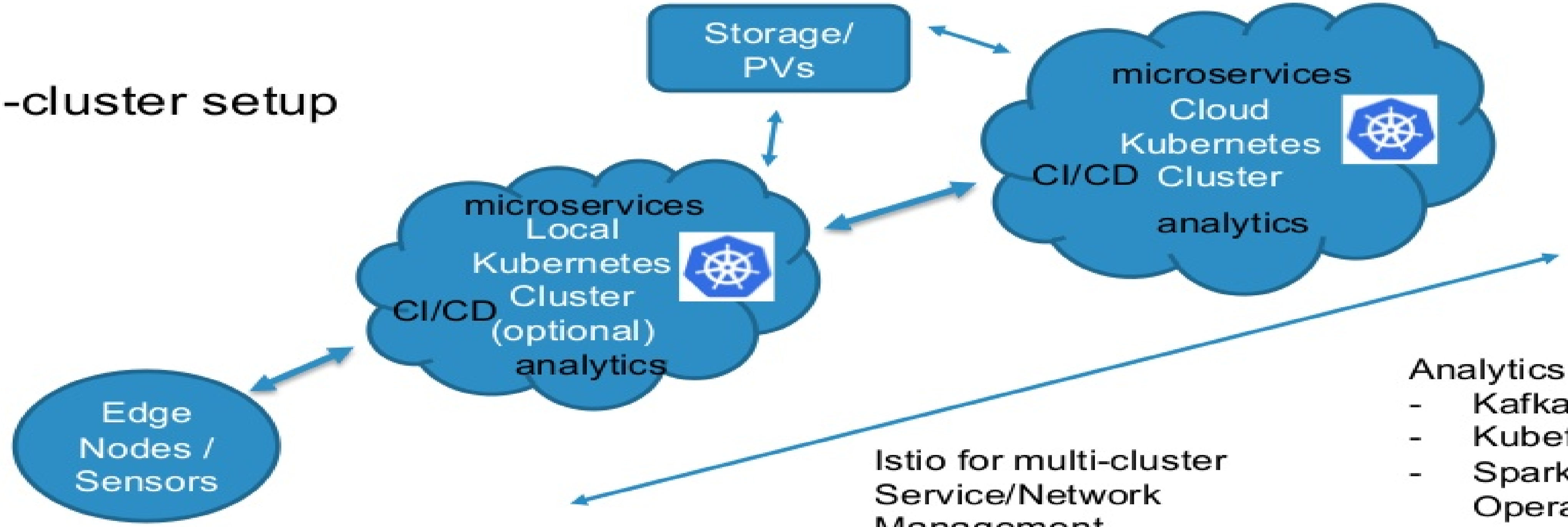
- MLlib for model training:
  - Parallel training of models
  - Distributed training of one model
  - Spark ML pipelines export formats (new in 2.4): https://issues.apache.org/jira/browse/SPARK-11239
  - Future: Project Hydrogen?
- Serving of models:
  - Directly with Spark Deep Learning
  - Custom sink + structured streaming?

Spark Tensorflow Connector

- Part of the data cleaning/transformation can be done in Spark before fed to TF.
- Uses dataframes to write TF records at the final step.
- tf.TFRecordReader() + a graph can be used to build a model for further processing.

Production Pipeline Architecture

Multi-cluster setup



Istio for multi-cluster Service/Network Management

Analytics:
- Kafka (Strimzi)
- Kubeflow
- Spark / Spark Operator

Why Kubeflow?

- Manages TF distributed training, serving and hyperparameter tuning in a declerative way: TFjobs.
- Easy access for data scientists with Jyputer notebook integration.
- Integration with Istio (https://istio.io/) for metrics and A/B testing.
- MXNet, PyTorch, MPI support (Horovod).
- Maturity?

Kubeflow Tips

- Configuration is passed via TF_CONFIG var.
  - image entrypoint needs to pass info for parameter servers, worker hosts to your TF code: (https://github.com/kubeflow/kubeflow/blob/master/tf-controller-examples/tf-cnn/launcher.py).
- Use PVs for storing your models or your data.
- Plan for your resources like Gpus etc.

QUESTIONS?