TensorFlow 未来发展洞察

彭靖田 才云科技

目录

Speech content

- **TensorFlow 发展现状**
- 更实时的命令式编程——Eager Execution
- 更易用的高层 API——Estimator & Keras
- 更灵活的输入流水线——tf.data
- 更成熟的端侧运行时——TensorFlow Lite

TensorFlow 前世今生

2011年 Google DistBelief
第一代机器学习系统

- Google搜索
- Google广告
- Google地图
- Google街景
- Google翻译
- Youtube
- ...

Large-Scale Deep Learning for Intelligent Computer Systems

Jeff Dean

Google Brain team in collaboration with many other teams

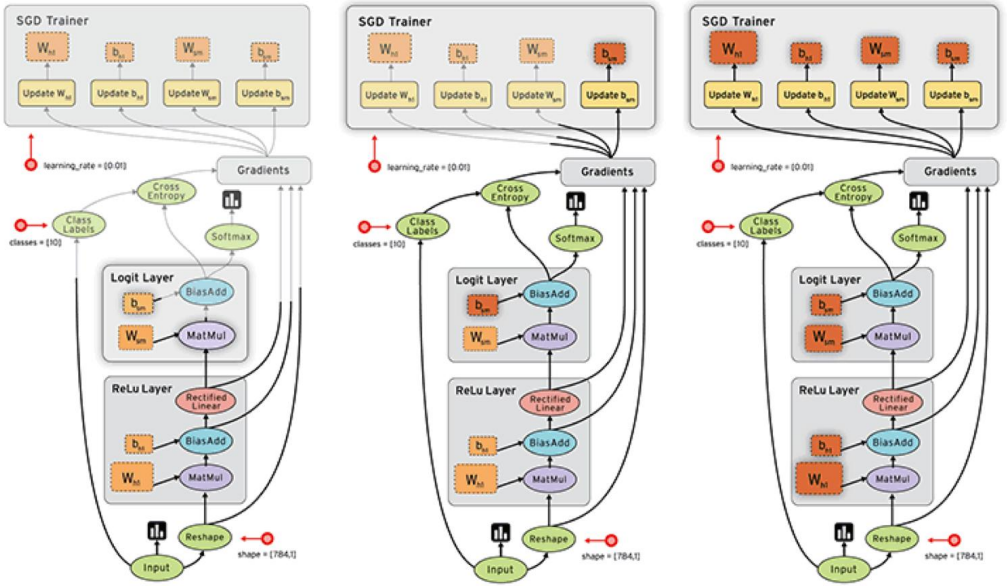
2015年 Google TensorFlow
第二代机器学习系统（开源）

- Airbnb
- Google
- Intel
- Uber
- 京东
- 小米
- ...

TensorFlow

TensorFlow 社区活跃度

TensorFlow 独特价值



- 教学资源充足
- 语言接口丰富
- 生产环境部署
- 模型设计灵活
- 超大集群并行
- 端云计算协同
- 运算性能强劲

TensorFlow 关键特性发布

2015.11.01
TensorFlow
开源

TensorFlow 关键特性发布

2015.11.01
TensorFlow
开源

2015.11.09
TensorFlow
0.5.0
第一个正式
发布版

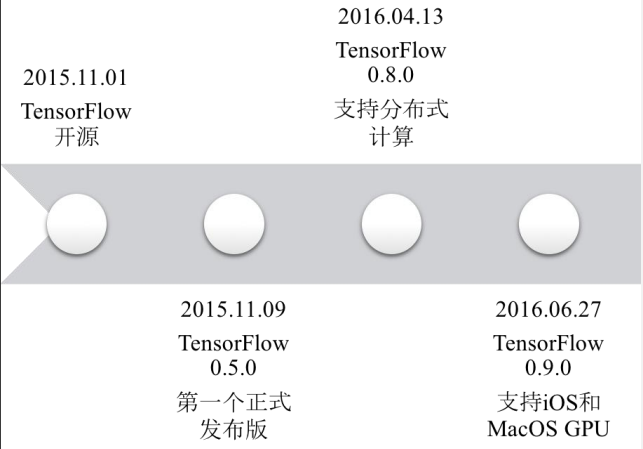
TensorFlow 关键特性发布

2015.11.01
TensorFlow
开源

2015.11.09
TensorFlow
0.5.0
第一个正式
发布版

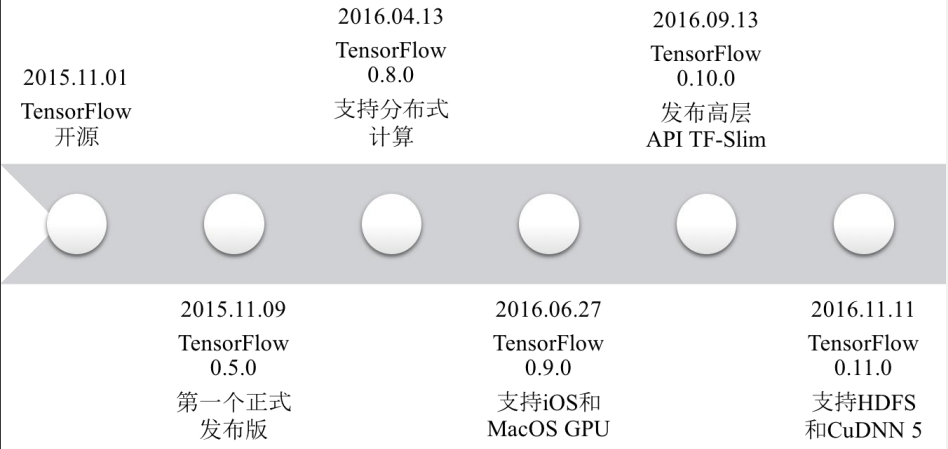
2016.04.13
TensorFlow
0.8.0
支持分布式
计算

TensorFlow 关键特性发布

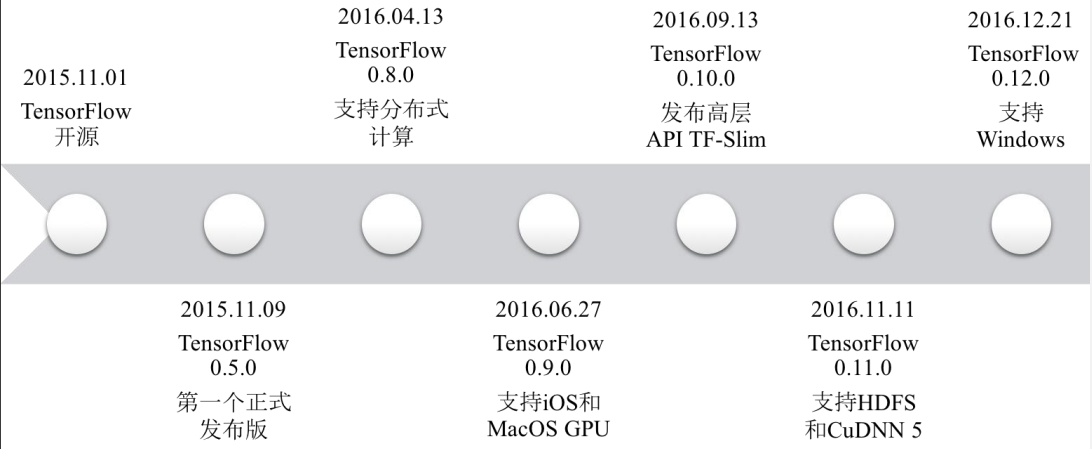


TensorFlow 关键特性发布

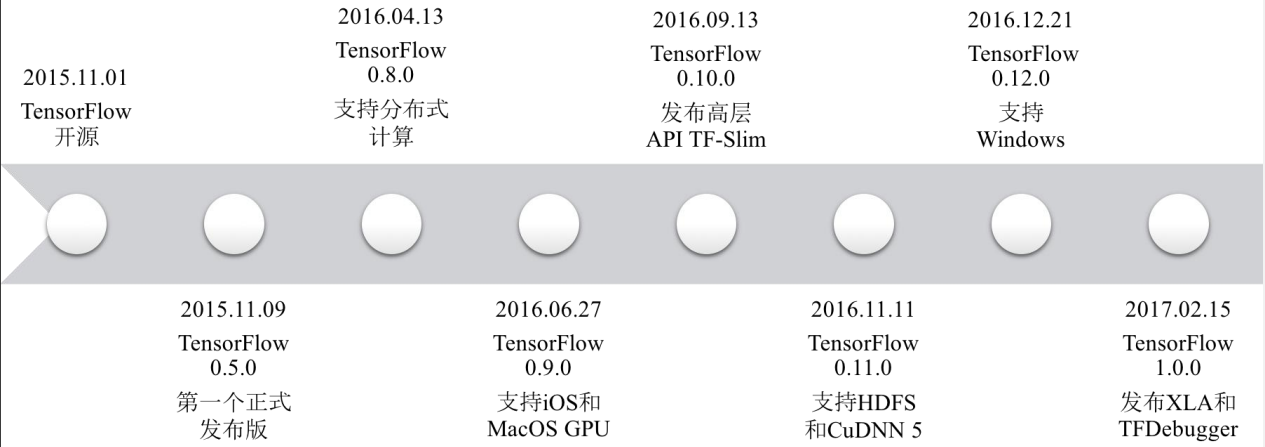
TensorFlow 关键特性发布



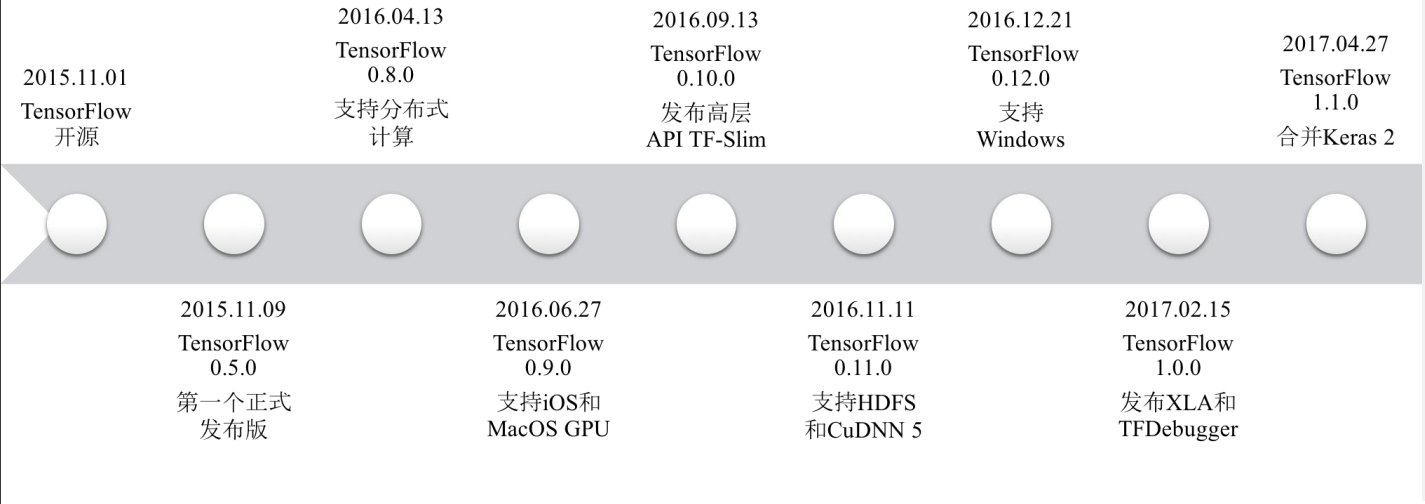
TensorFlow 关键特性发布



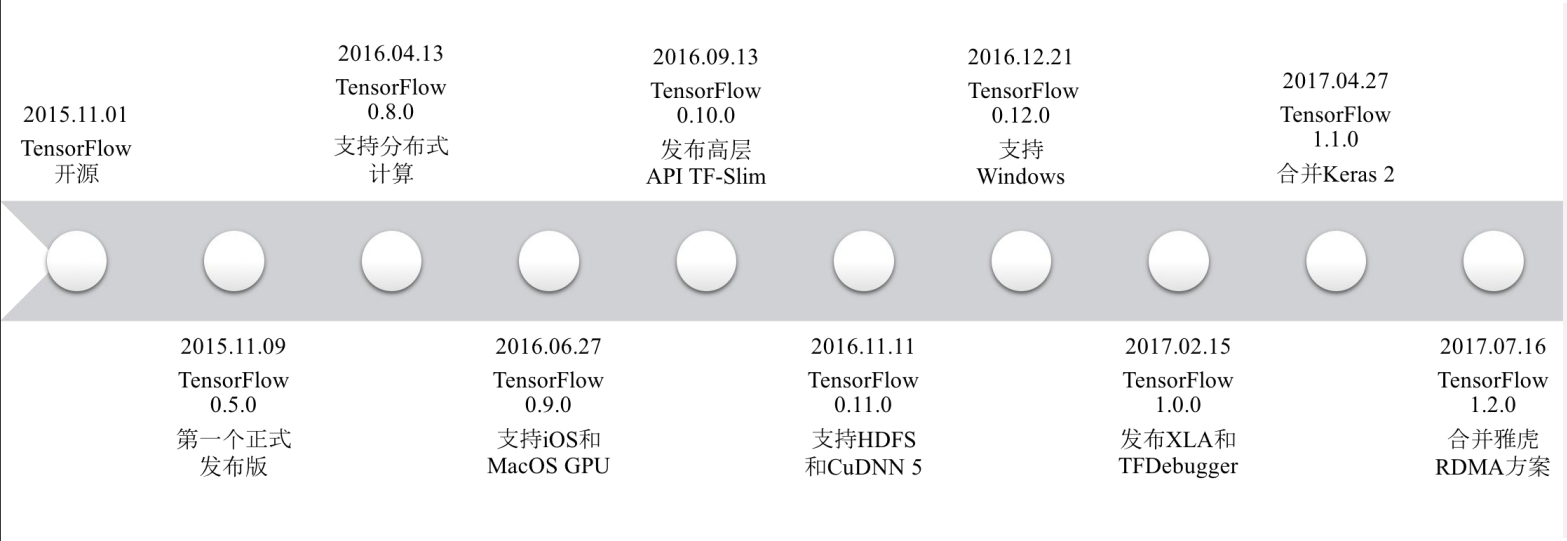
TensorFlow 关键特性发布



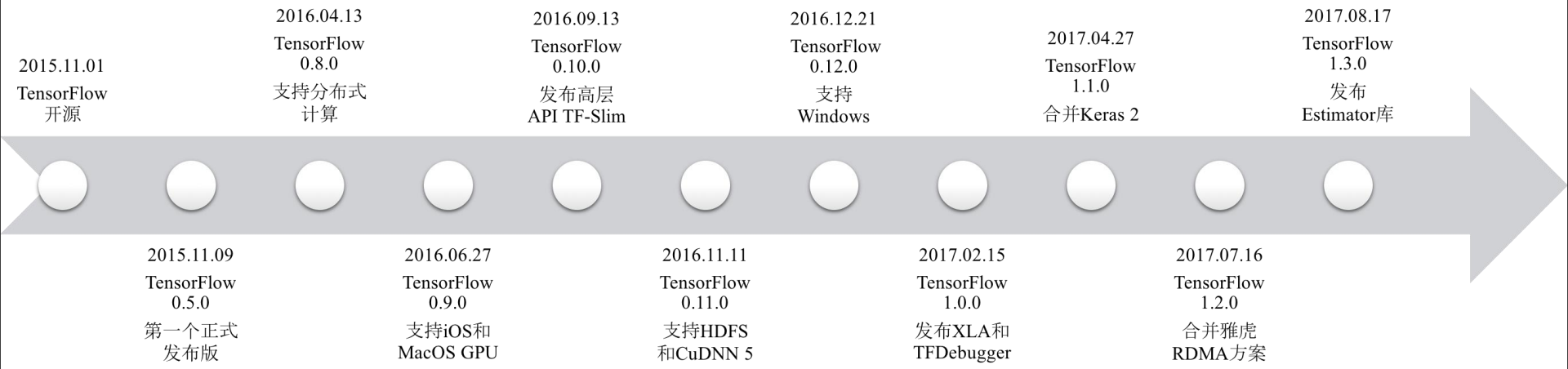
TensorFlow 关键特性发布



TensorFlow 关键特性发布



TensorFlow 关键特性发布

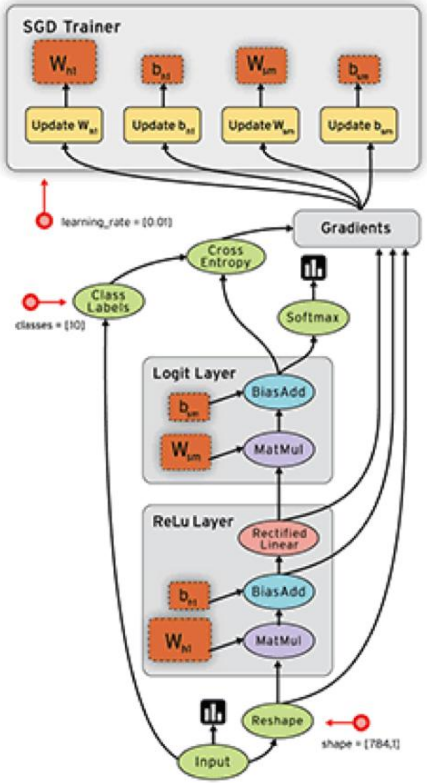


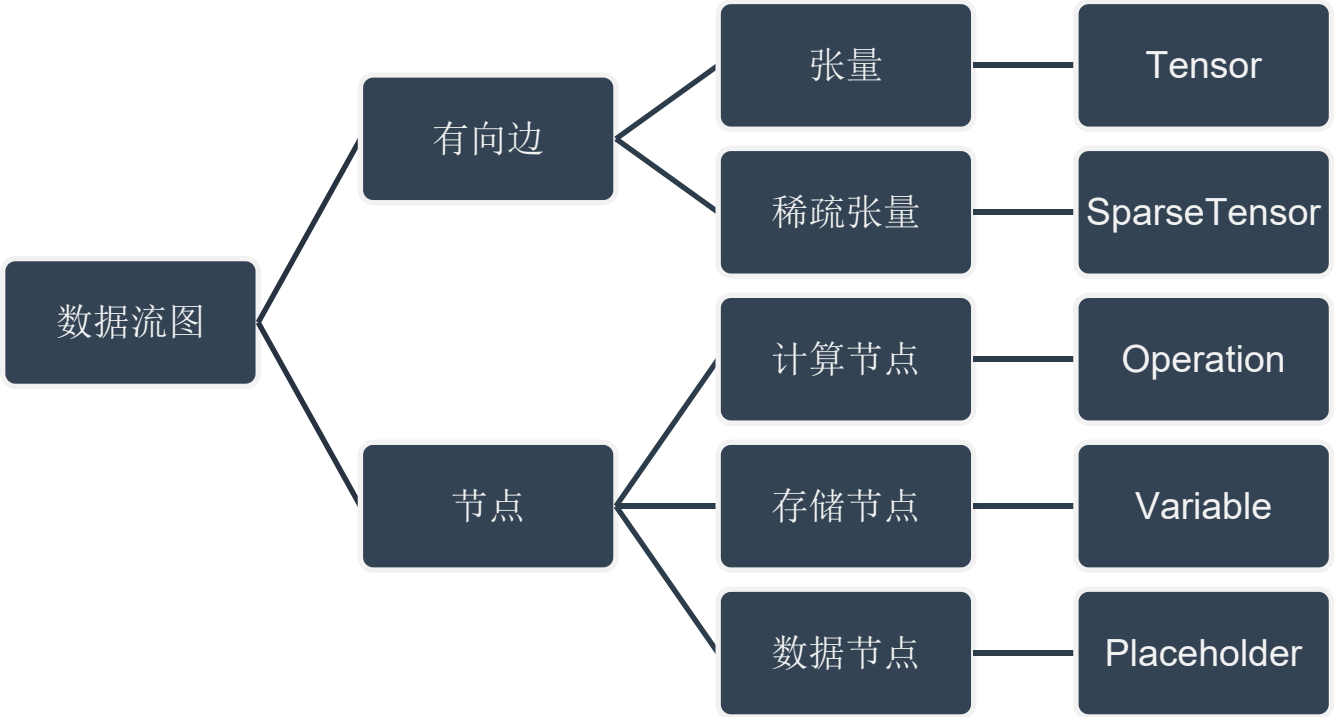
目录

Speech content

- TensorFlow 发展现状
- **更实时的命令式编程——Eager Execution**
- 更易用的高层 API——Estimator & Keras
- 更灵活的输入流水线——tf.data
- 更成熟的端侧运行时——TensorFlow Lite

TensorFlow 数据流图





会话提供了估算张量和执行操作的**运行环境**，它是发放计算任务的客户端，所有计算任务都由它连接的执行引擎完成。一个会话的典型使用流程分为以下3步：

```
# 1. 创建会话
sess = tf.Session(target=..., graph=..., config=...)
# 2. 估算张量或执行操作
sess.run(...)
# 3. 关闭会话
sess.close()
```

参数名称	功能说明
target	会话连接的执行引擎
graph	会话加载的数据流图
config	会话启动时的配置项

```
import tensorflow as tf
# 创建数据流图: z = x * y
x = tf.placeholder(tf.float32, name='x')
y = tf.placeholder(tf.float32, name='y')
z = tf.multiply(x, y, name='z')
# 创建会话
sess = tf.Session()
# 向数据节点x和y分别填充浮点数3.0和2.0，并输出结果
print(sess.run(z, feed_dict={x: 3.0, y:2.0}))
'''
输出6.0
'''
```

```
import tensorflow as tf
# 创建数据流图: c = a + b
a = tf.constant(1.0, name='a')
b = tf.constant(2.0, name='b')
c = tf.add(a, b, name='c')
# 创建会话
sess = tf.Session()
# 估算张量C的值
print(sess.run(c))
'''
输出3.0
'''
```

TensorFlow Debug 困难原因：

- 数据流图是领域特定语言
- 无法实时交互计算
- 难以定位问题节点

Eager Execution 命令式编程前端

- 原生 Python Control Flow，支持循环和条件分支
- 全新的梯度计算
- 删除会话，通过 Python 解释器直接连接计算引擎
- 状态：近期发布 Alpha 版本

Dataflow Graph

```
# Demo
a = tf.constant(6)
while a != 1:
    if a % 2 == 0:
        a = a / 2
    else:
        a = 3 * a + 1
    print(a)
```

```
Tensor("add_1:0", shape=(), dtype=int32)
Tensor("add_2:0", shape=(), dtype=int32)
Tensor("add_3:0", shape=(), dtype=int32)
Tensor("add_4:0", shape=(), dtype=int32)
Tensor("add_5:0", shape=(), dtype=int32)
Tensor("add_6:0", shape=(), dtype=int32)
Tensor("add_7:0", shape=(), dtype=int32)
```

Eager Execution

```
# Outputs
tf.Tensor(3, dtype=int32)
tf.Tensor(10, dtype=int32)
tf.Tensor(5, dtype=int32)
tf.Tensor(16, dtype=int32)
tf.Tensor(8, dtype=int32)
tf.Tensor(4, dtype=int32)
tf.Tensor(2, dtype=int32)
tf.Tensor(1, dtype=int32)
```

Gradients

- Operations executed are recorded on a tape.
- Convert TensorFlow dataflow graph to Python function

对函数求二阶梯度

```
grad = tfe.gradients_function(square)
gradgrad = tfe.gradients_function(
    lambda x: grad(x)[0])    # 二阶
```

对自定义函数求梯度

```
def f(x):
    y = tf.square(x)
    def grad_fn(dresult):
        return [x+y]
    return prod, grad_fn

grad = tfe.gradient_function(f)
```

使用优化器求梯度

```
# 创建模型
x = tf.placeholder(tf.float32, [None, 784]) # 图像数据
W = tf.Variable(tf.zeros([784, 10]))       # 模型权重
b = tf.Variable(tf.zeros([10]))            # 模型偏置
y = tf.matmul(x, W) + b             # 推理操作
y_ = tf.placeholder(tf.float32, [None, 10])    # 图像标签
# 使用交叉熵作为损失值
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
# 创建梯度下降优化器
optimizer = tf.train.GradientDescentOptimizer(FLAGS.learning_rate)
# 定义单步训练操作
train_op = optimizer.minimize(cross_entropy)
```

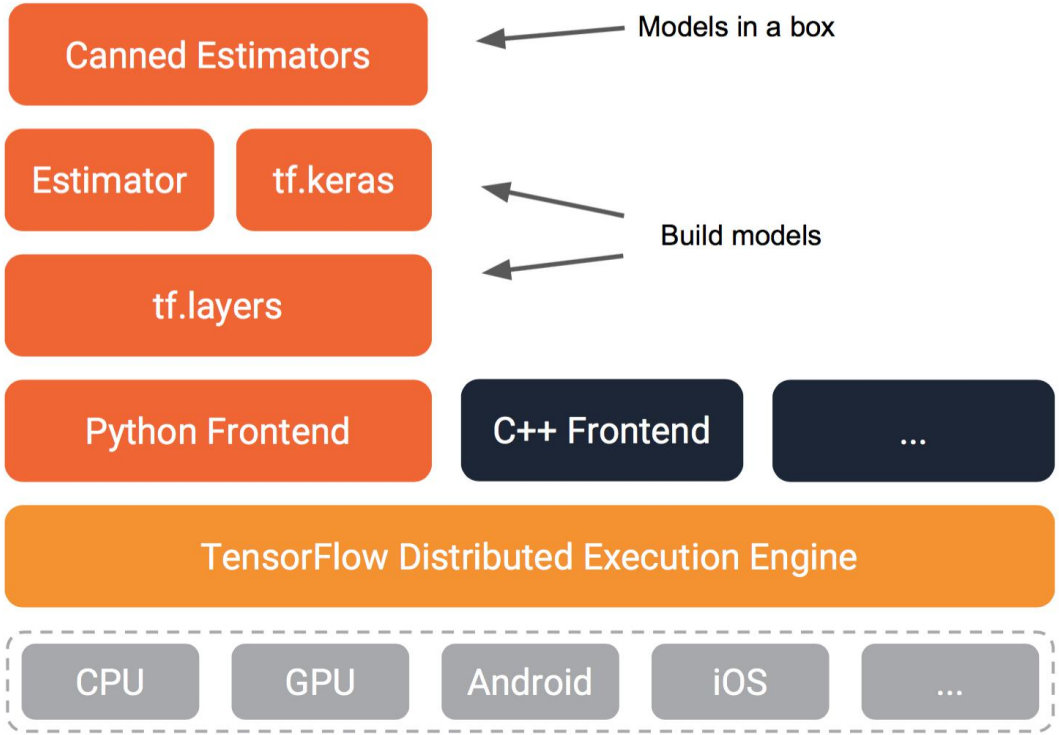
目录

Speech content

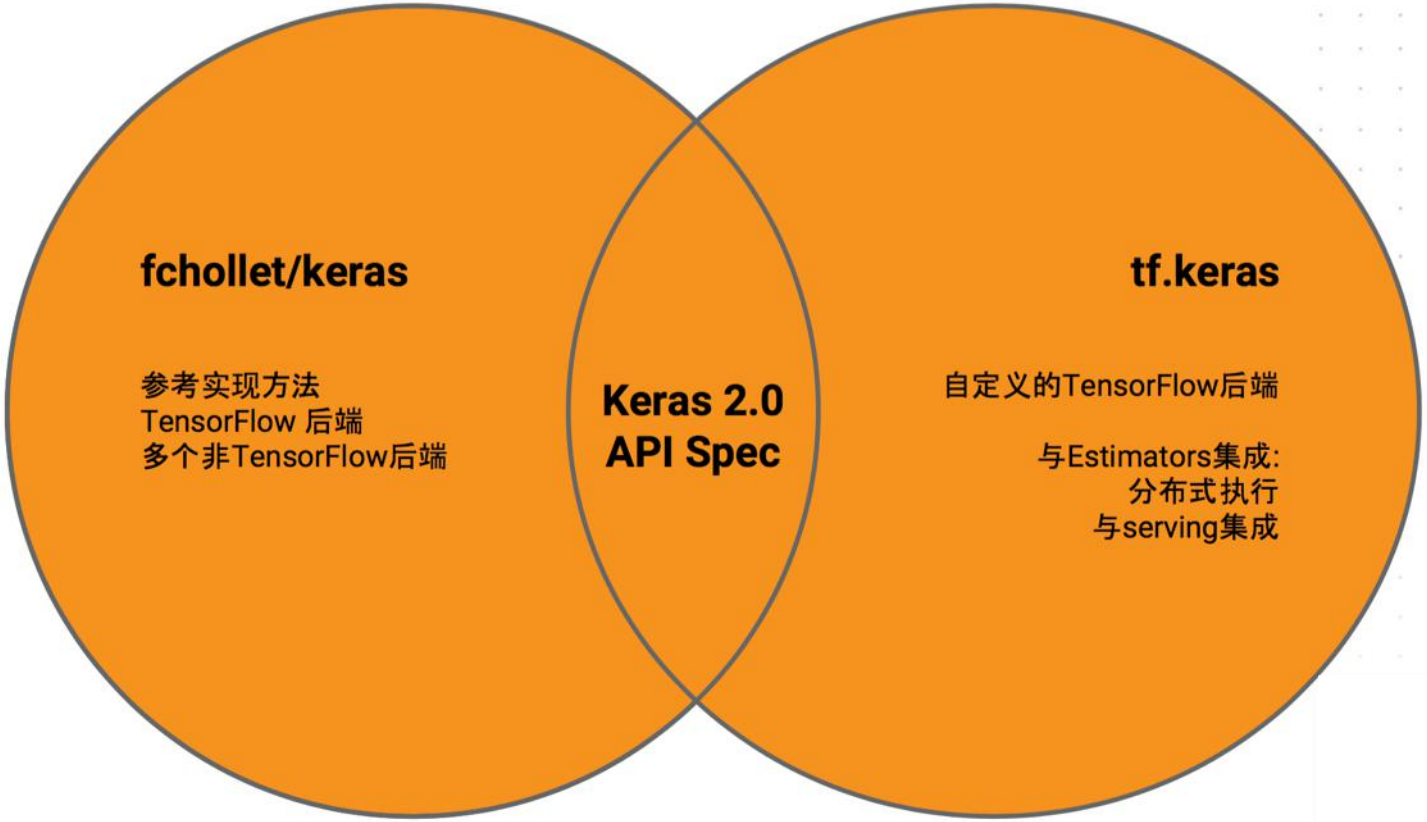
- TensorFlow 发展现状
- 更实时的命令式编程——Eager Execution
- 更易用的高层 API——Estimator & Keras
- 更灵活的输入流水线——tf.data
- 更成熟的端侧运行时——TensorFlow Lite

Estimators : A high-level TensorFlow API that greatly simplifies ML programming.

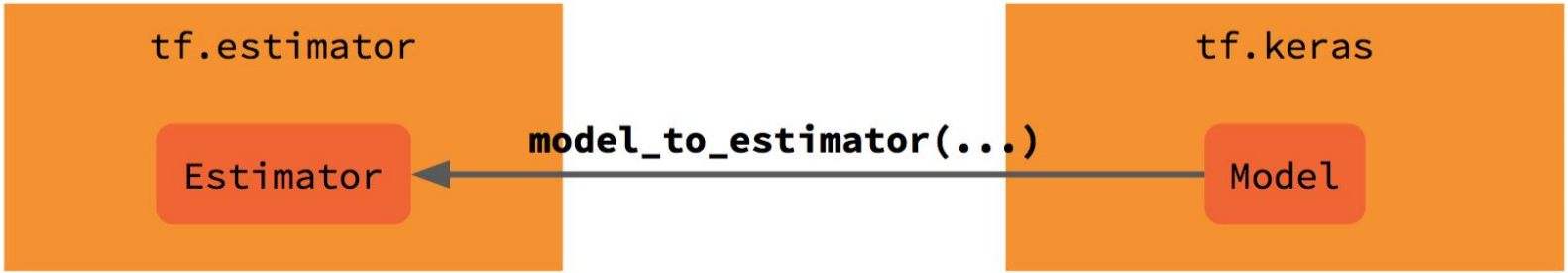
- 实现了：
  - 训练循环
  - checkpointing
  - 与 TensorBoard 集成
  - 与 Serving 集成
- Canned Estimator -> models
- 可执行分布式：
  - Worker 失败后自动恢复
  - 与加速器结合



fchollet/Keras 项目与 tf.keras 模块



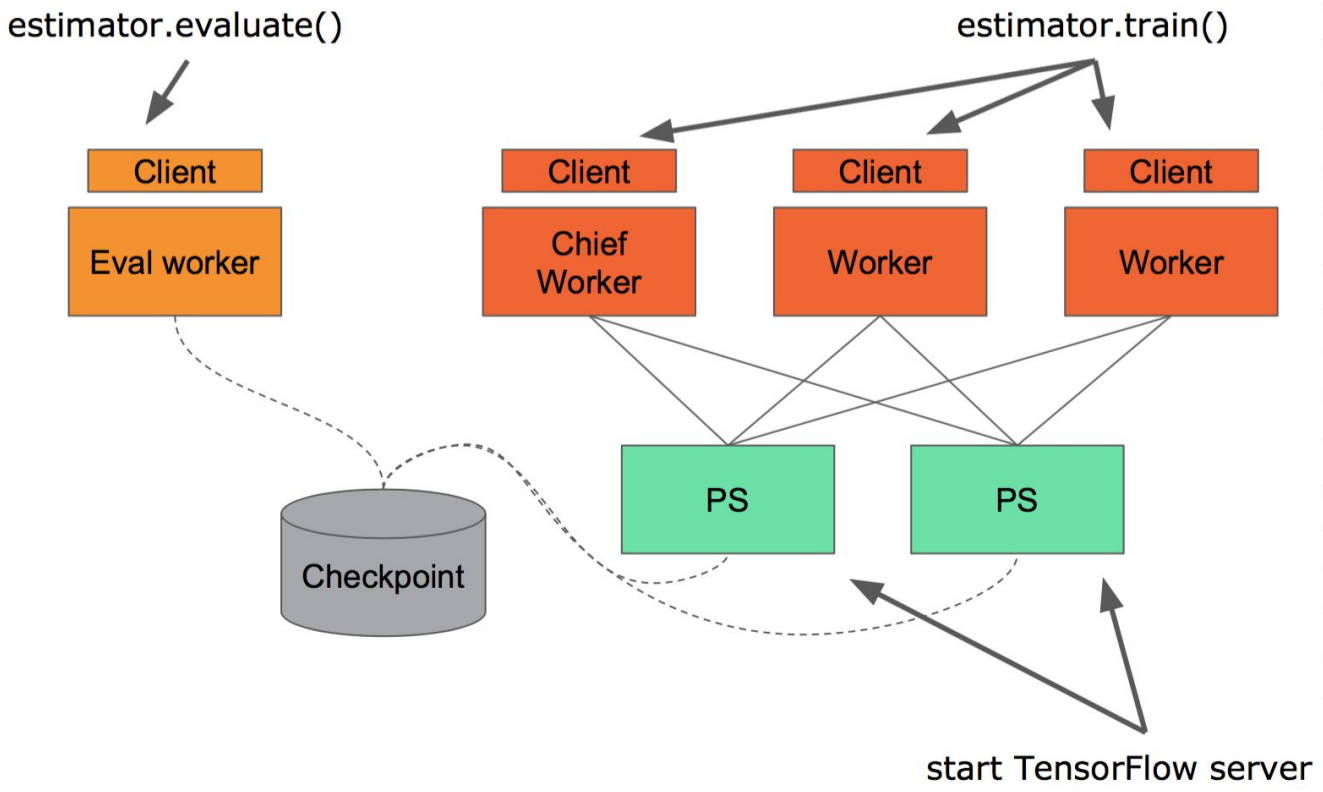
Estimators 与 Keras 的集成



TensorFlow 1.4 使用建议

1. 尽可能使用最高层API
2. 使用 tf.layers / tf.keras （官方持续维护）
3. 使用 Estimator 进行单机和分布式训练

使用 Estimator 进行分布式训练



目录

Speech content

- TensorFlow 发展现状
- 更实时的命令式编程——Eager Execution
- 更易用的高层 API——Estimator & Keras
- **更灵活的输入流水线——tf.data**
- 更成熟的端侧运行时——TensorFlow Lite

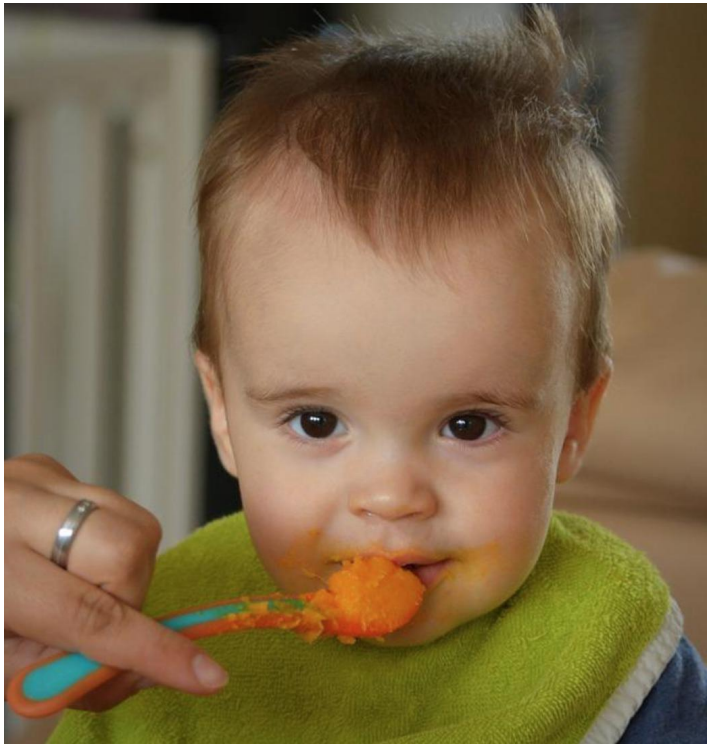
Why New Pipeline?

- Input data is the lifeblood of machine learning
- Modern accelerators need faster input pipelines
- Getting your data into TensorFlow can be painful

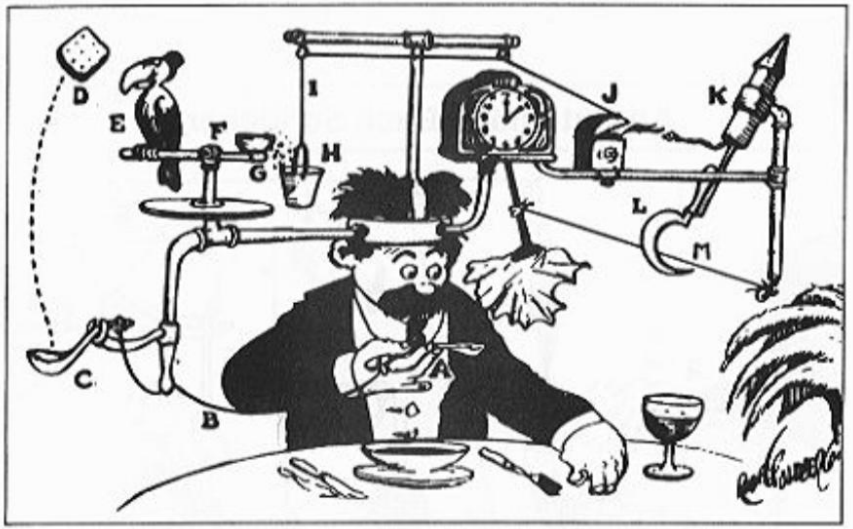
更灵活的输入流水线——tf.data

```
# Feeding

sess.run(...,
    feed_dict={x: features,
               y: labels})
```

灵活　　低效



“Starting the queue runners”

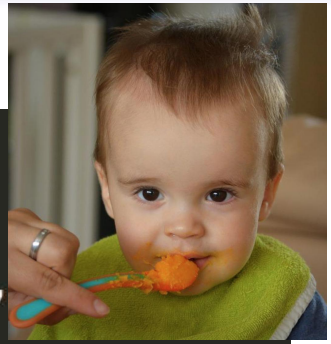



```
# Queues

files = string_input_producer(...)
record = TFRecordReader().read(files)
parsed = parse_example(record, ...)
batch = batch(parsed, 32)
```

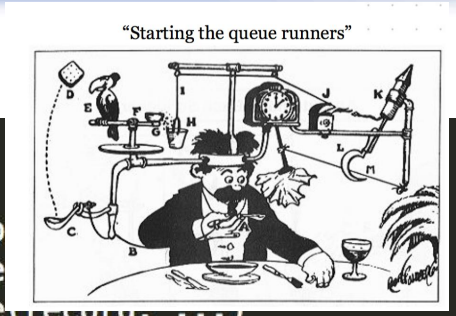
死板　　高效

更灵活的输入流水线——tf.data

```
# Feeding

sess.run(...,
    feed_dict={x: features,
               y: labels})
```



```
# Queues

files = string_input_p
record = TFRecordReade
parsed = parse_example
batch = batch(parsed, 32)
```



灵活　　低效

死板　　高效

灵活高效

函数式输入流水线

New Input Pipeline = Lazy Lists

Functional programming to the rescue!

- Data elements have the same type
- Dataset might be too large to materialize all at once... or infinite
- Compose functions like map() and filter() to preprocess

- A well-studied area, applied in existing languages.
- Huge literature on optimization (stream fusion etc.)

Dataset 接口 —— Data sources and functional transformations

Create a Dataset from one or more tf.Tensor objects:

- Dataset.from_tensors((features, labels))
- Dataset.from_tensor_slices((features, labels))
- TextLineDataset(filenames)

Or create a Dataset from another Dataset:

- dataset.map(lambda x: tf.decode_jpeg(x))
- dataset.repeat(NUM_EPOCHS)
- dataset.batch(BATCH_SIZE)

Dataset 接口 —— Data sources and functional transformations

Or (in TensorFlow 1.4) create a Dataset from a Python generator:

- def generator():
  while True:
  yield ...
- Dataset.from_generator(generator, tf.int32)

Iterator 接口 —— Sequential access to Dataset elements

Create an Iterator from a Dataset:

- dataset.make_one_shot_iterator()
- dataset.make_initializable_iterator()

Initialize the Iterator if necessary:

- sess.run(iterator.initializer, feed_dict=PARAMS)

Get the next element from the Iterator:

- next_element = iterator.get_next()

```
while ...:
    sess.run(next_element)
```

更灵活的输入流水线——tf.data


```
dataset = …

# A one-shot iterator automatically initializes itself on first use.
iterator = dataset.make_one_shot_iterator()

# The return value of get_next() matches the dataset element type.
images, labels = iterator.get_next()
train_op = model_and_optimizer(images, labels)

# Loop until all elements have been consumed.
try:
  while True:
    sess.run(train_op)
except tf.errors.OutOfRangeError:
  pass
```

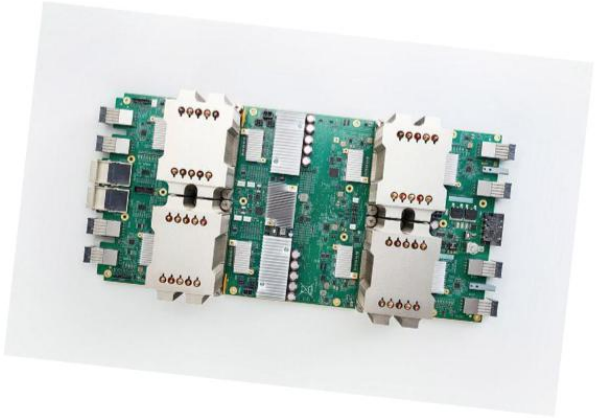
目录

Speech content

- TensorFlow 发展现状
- 更实时的命令式编程——Eager Execution
- 更易用的高层 API——Estimator & Keras
- 更灵活的输入流水线——tf.data
- 更成熟的端侧运行时——TensorFlow Lite

M1计算平台发展趋势

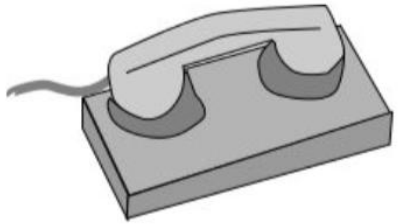
More on-device ML



More ML hardware

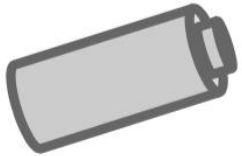
Why on-device?

Offline



Low-bandwidth

Latency



Power

困难与挑战

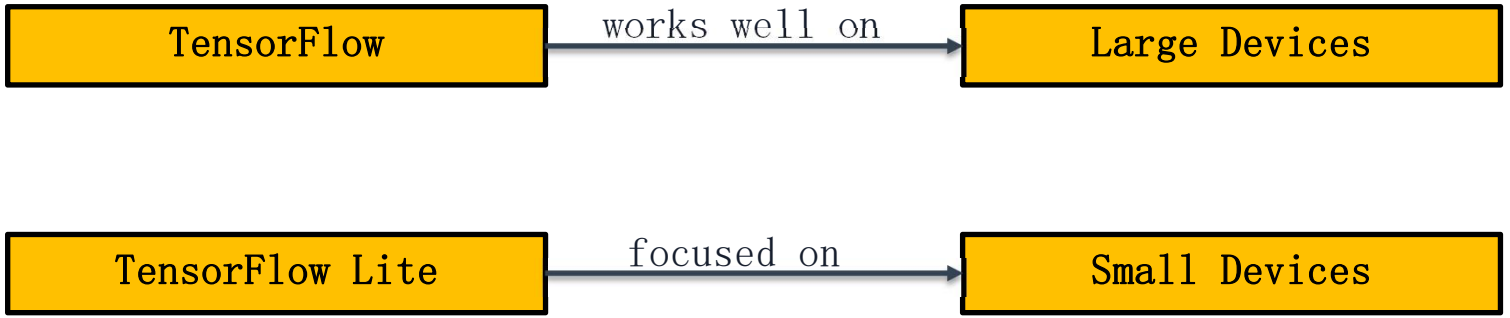
Resources

- Bandwidth
- Memory
- Computation

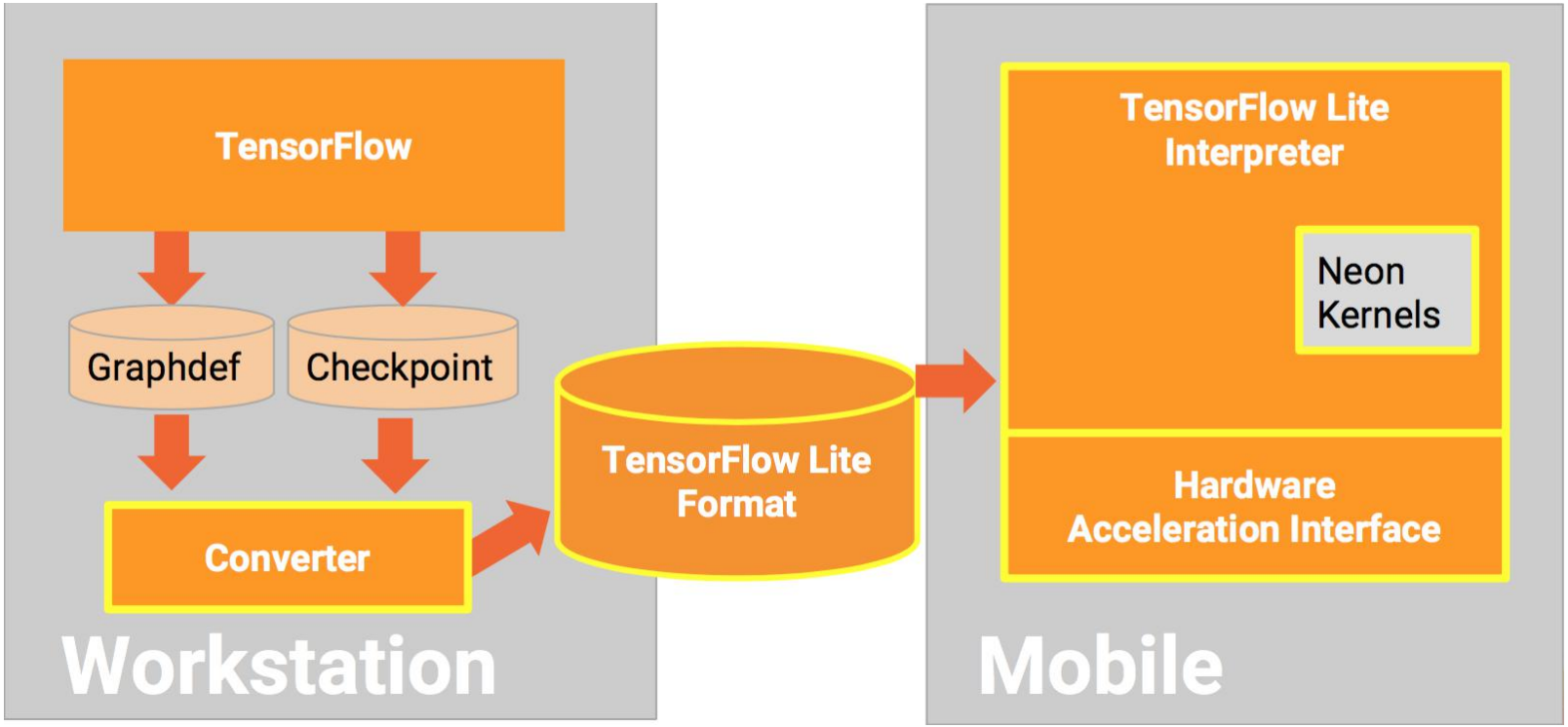
Heterogeneity

- GPUs
- CPUs
- DSPs
- ...

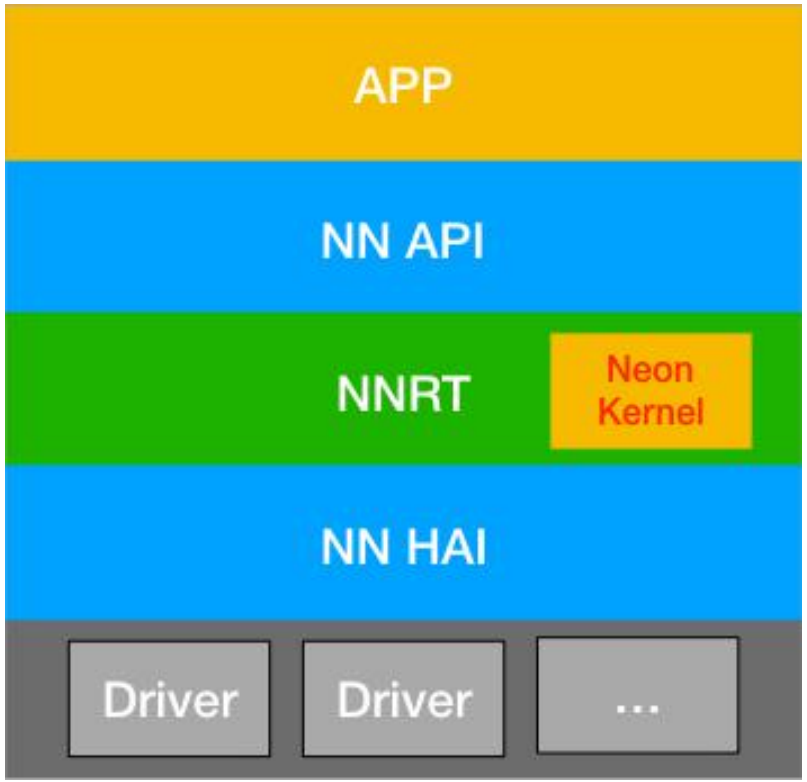
TensorFlow vs TensorFlow Lite



TensorFlow Lite 逻辑架构



Neural Network API in Android Framework



caicloud
才云

谢谢大家！