goto; chicago

Developing Machine Learning Models

Kevin Tsai

goto; chicago

Click ‘Rate Session’
to rate session
and ask questions.

follow us @gotochgo

GOTO Chicago 2018

Developing Machine Learning Models

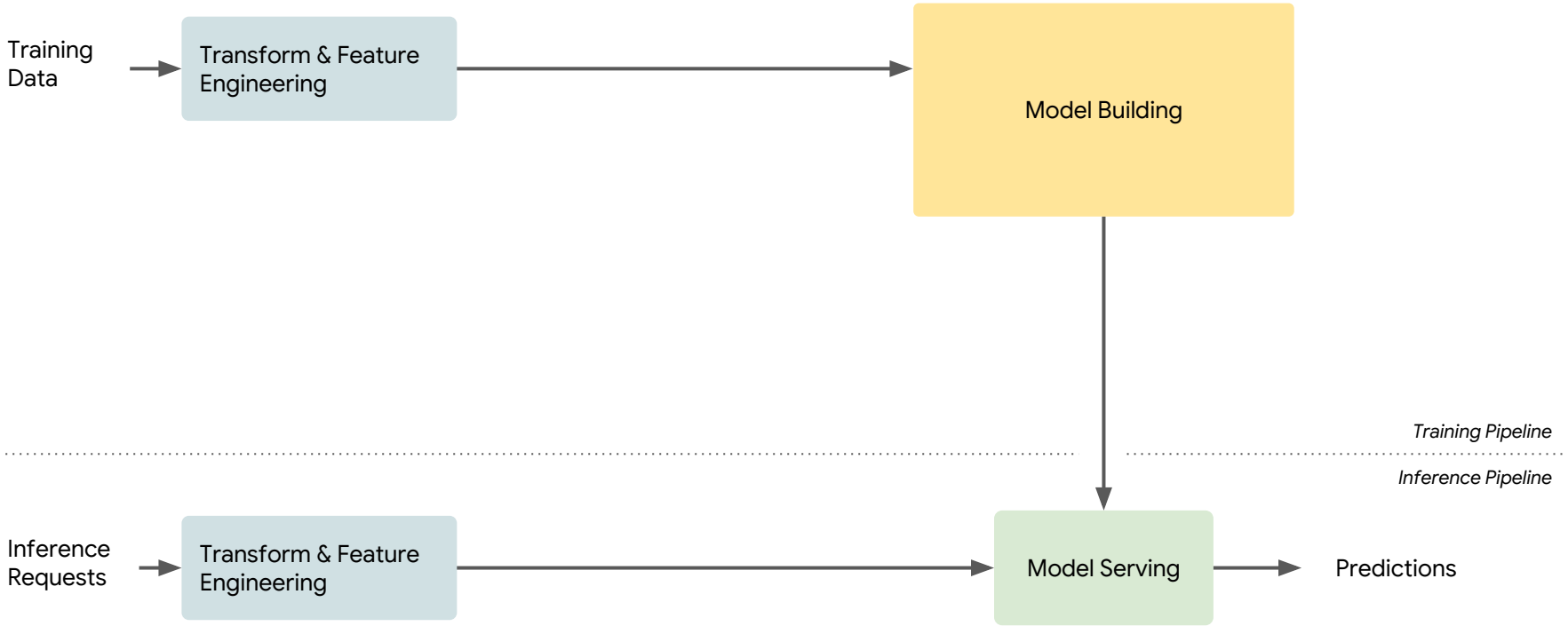
Kevin Tsai, Google

Agenda

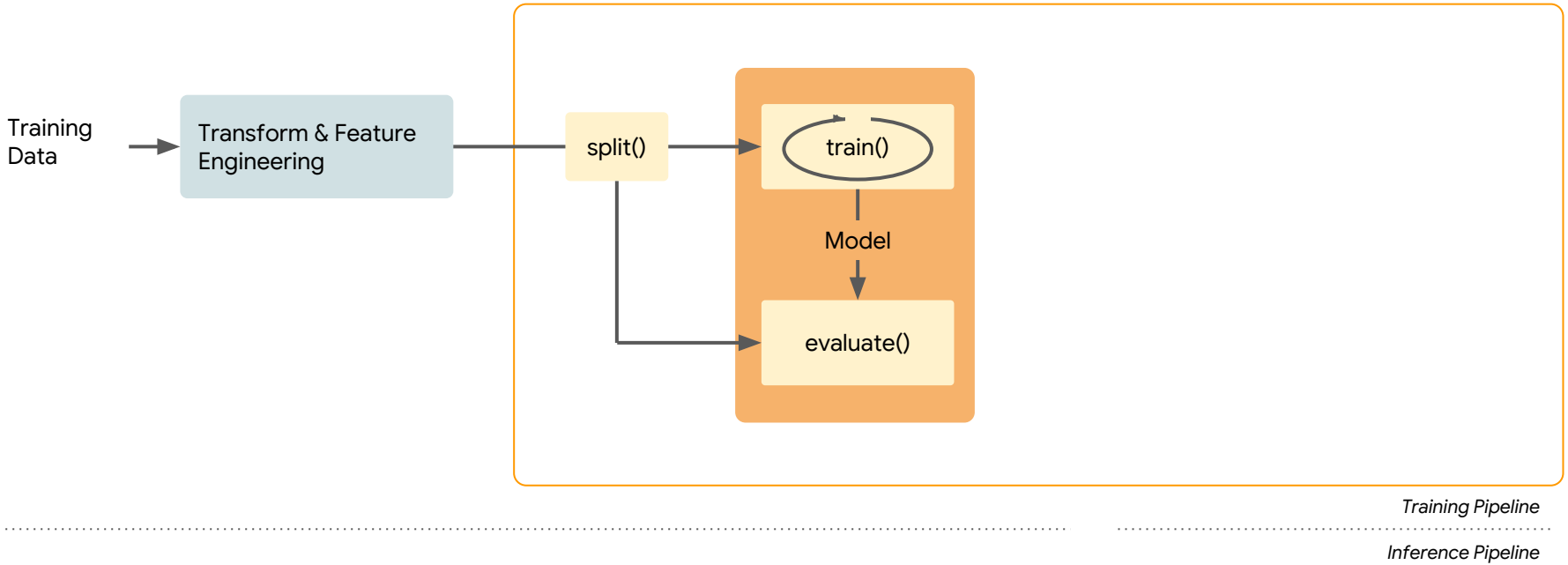
Machine Learning with tf.estimator
Performance pipelines with TFRecords and tf.data
Distributed Training with GPUs and TPUs

Machine Learning with tf.estimator

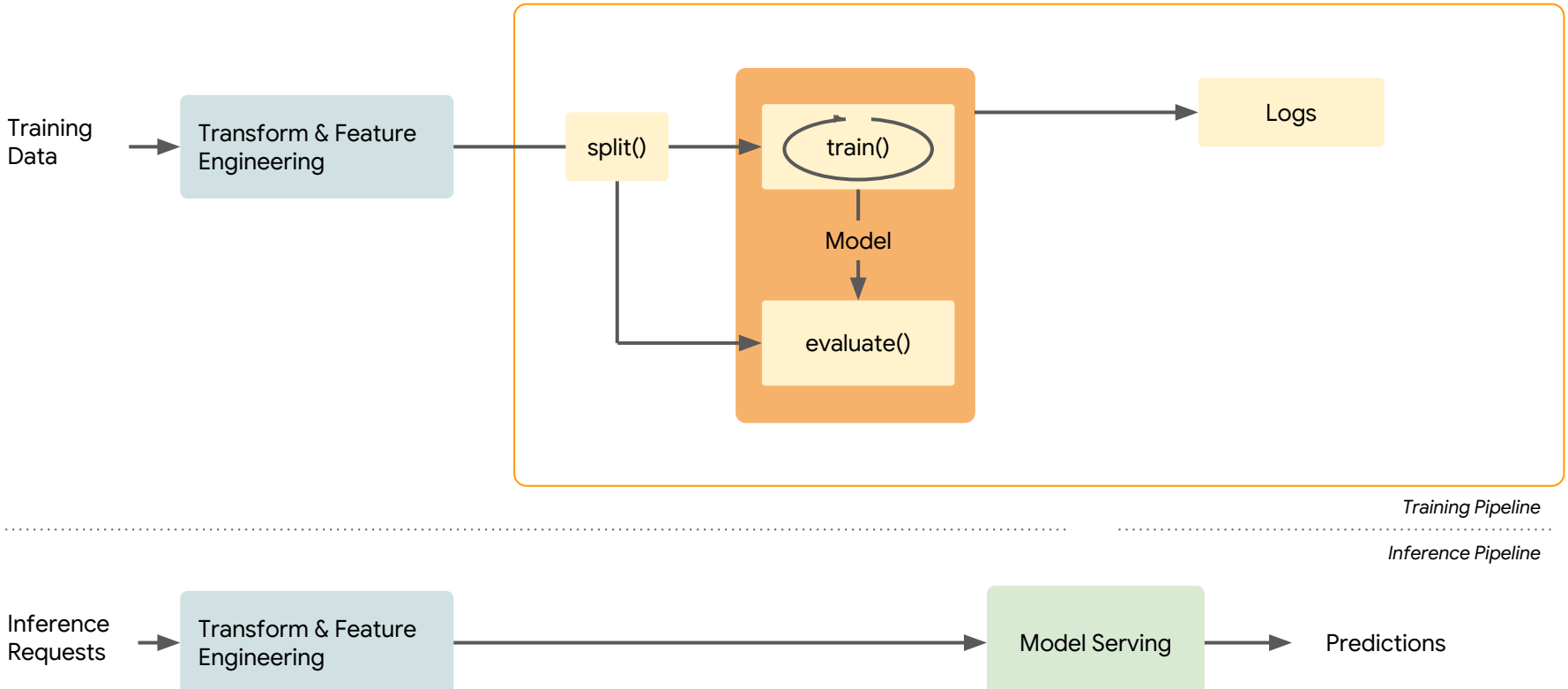
Machine Learning Pipeline



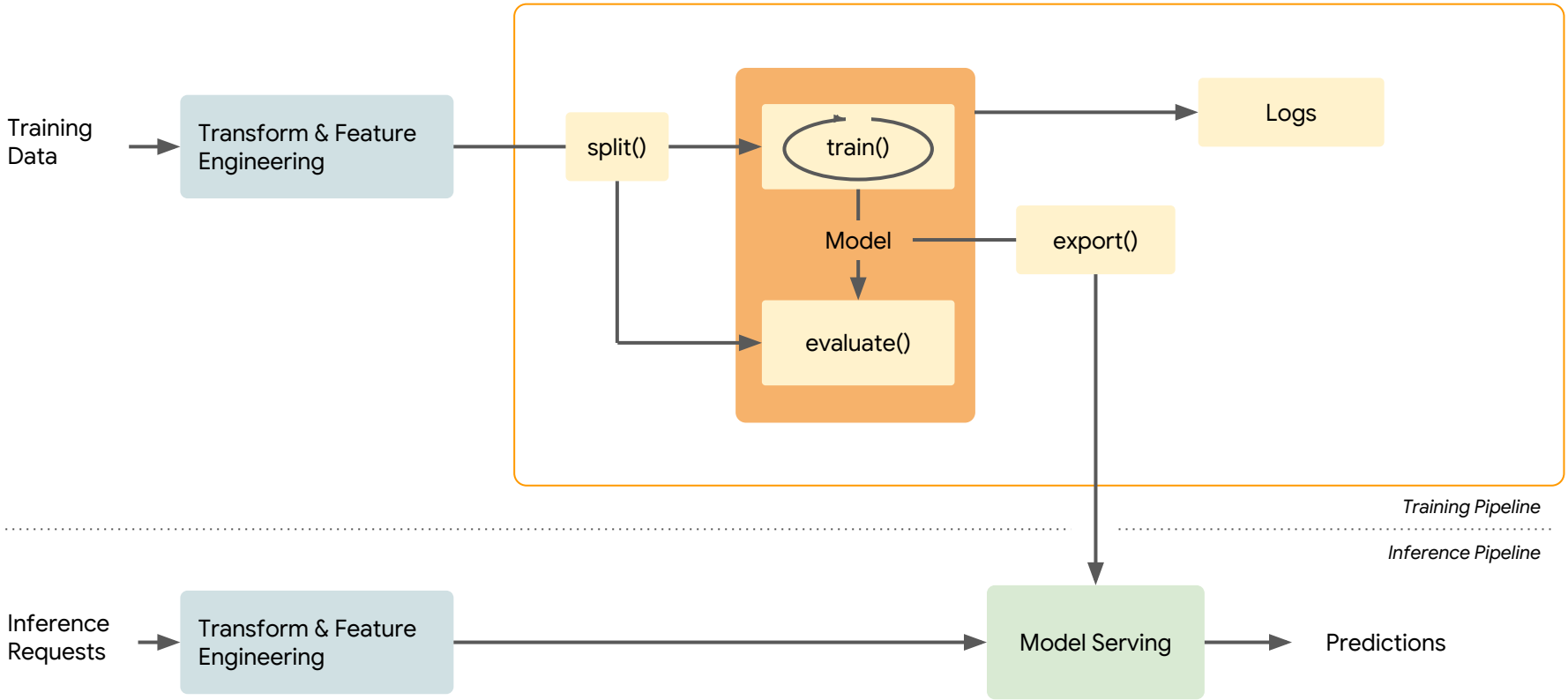
Machine Learning Pipeline



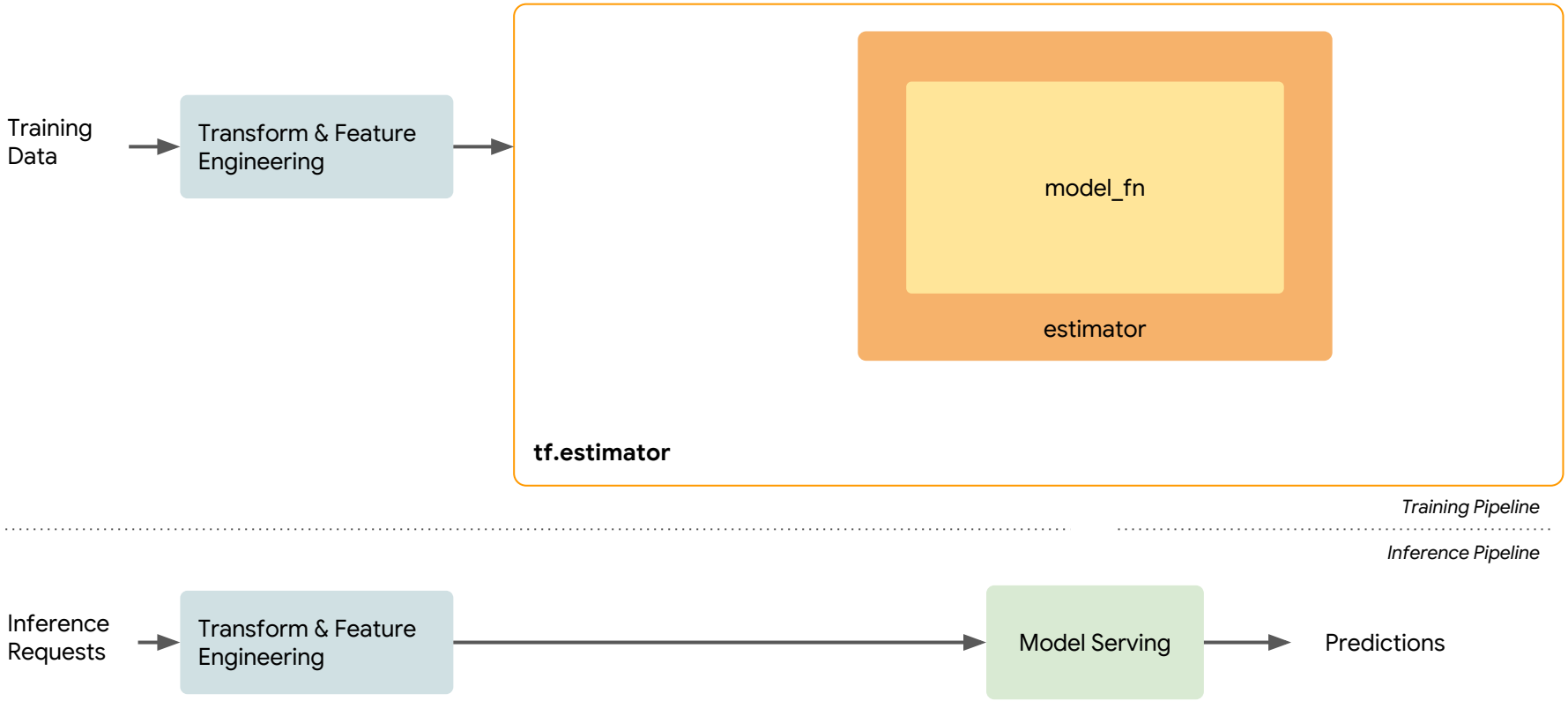
Machine Learning Pipeline



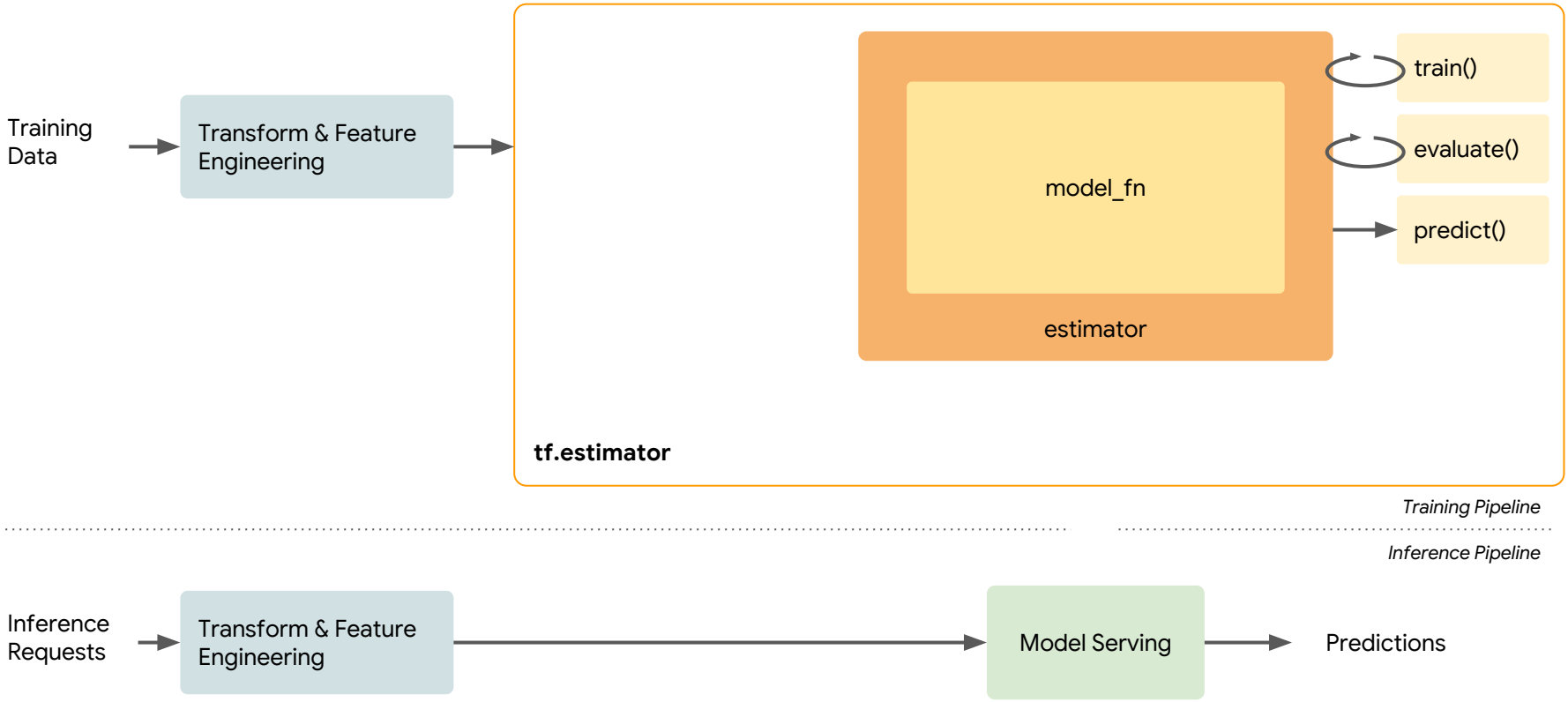
Machine Learning Pipeline



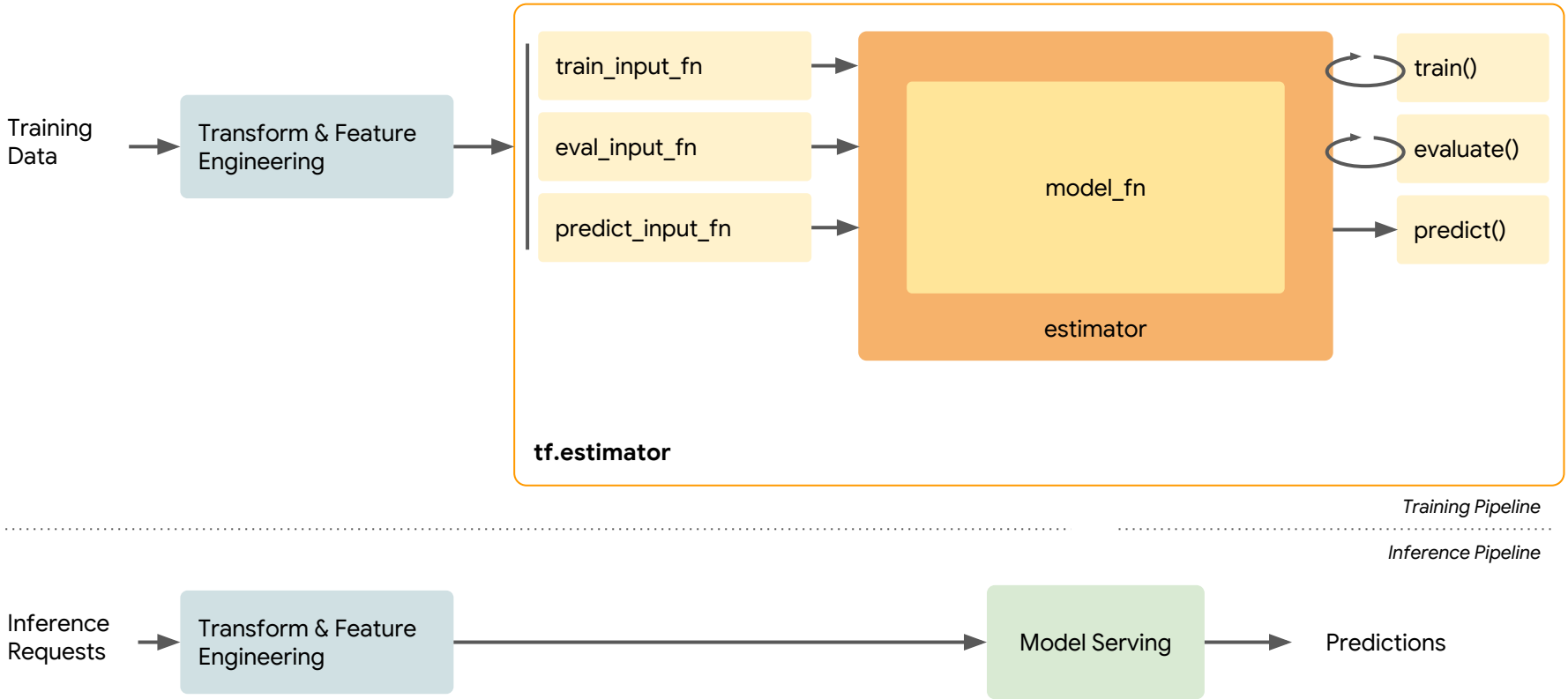
Machine Learning Pipeline with tf.estimator



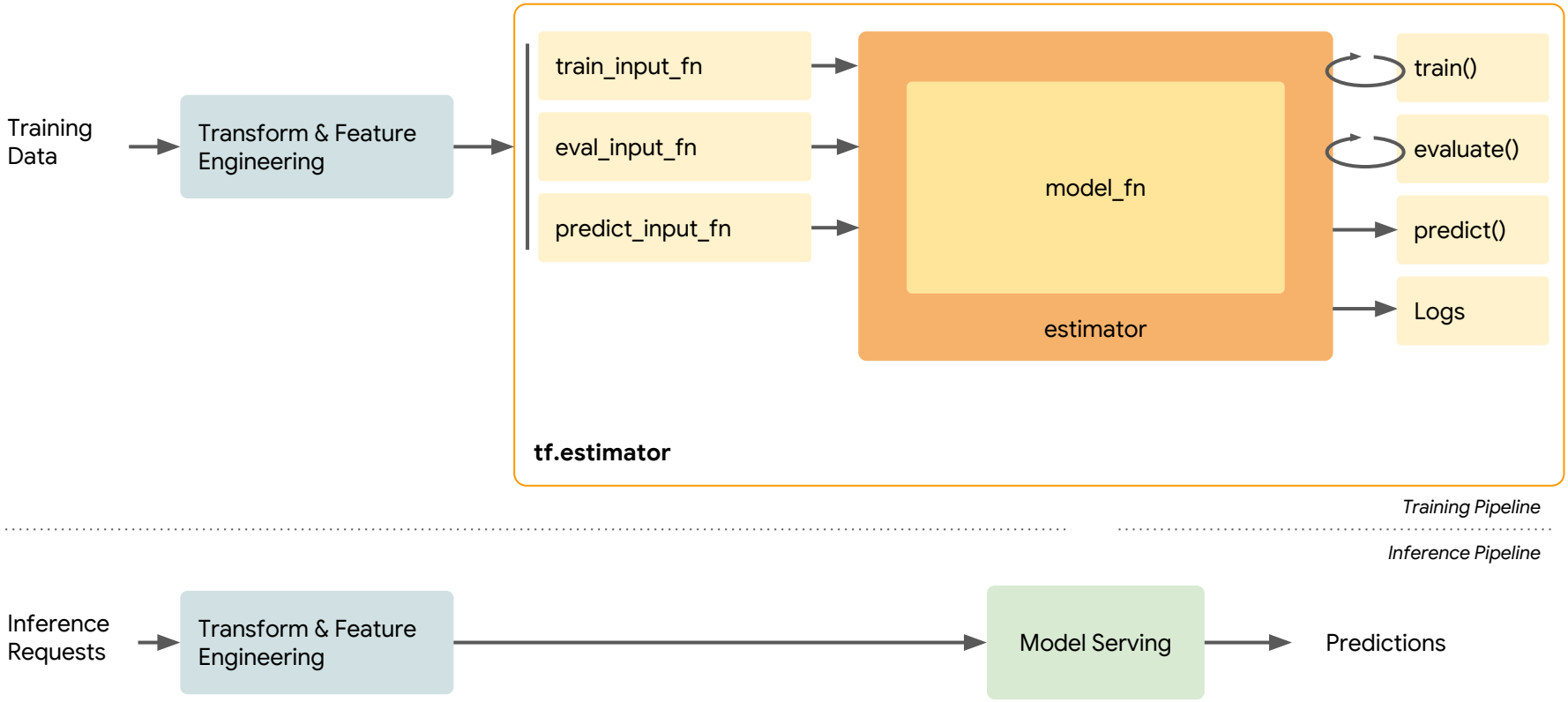
Machine Learning Pipeline with tf.estimator



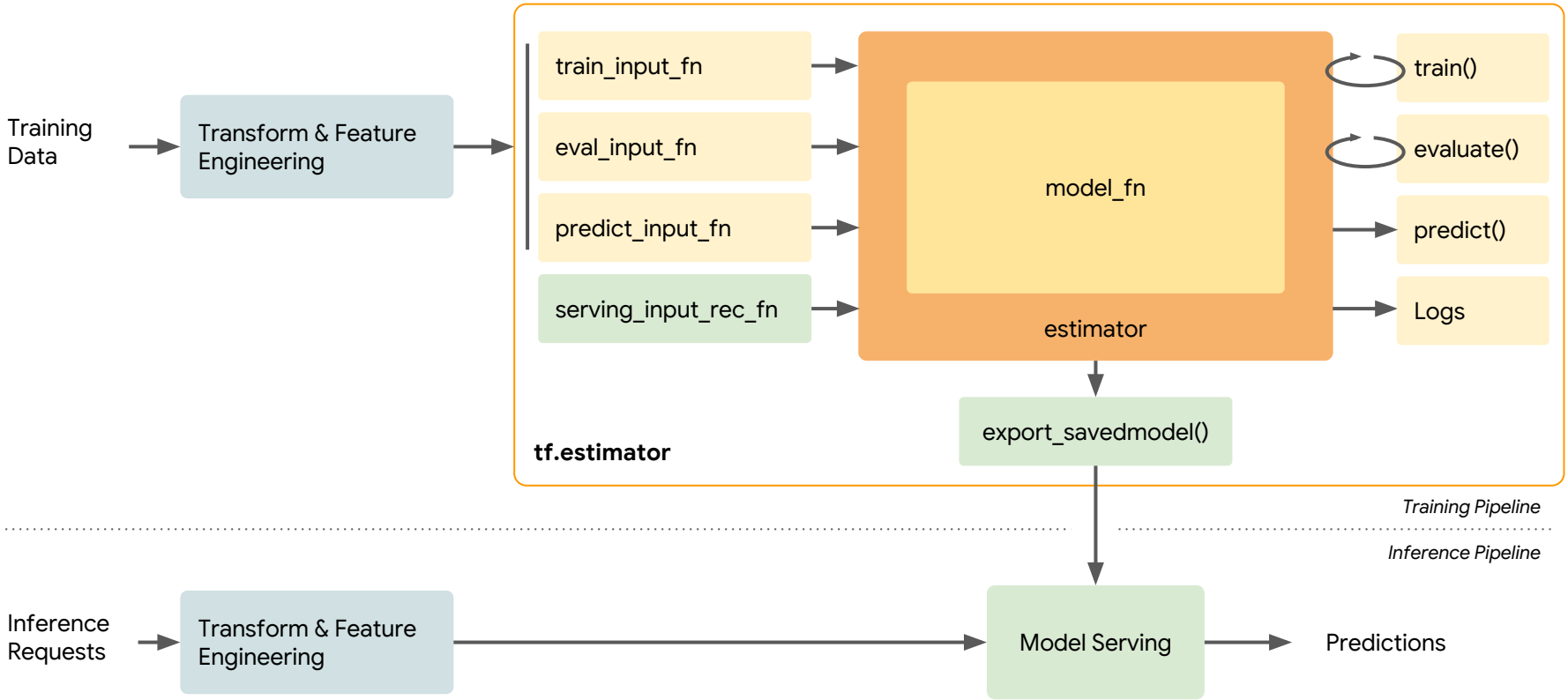
Machine Learning Pipeline with tf.estimator



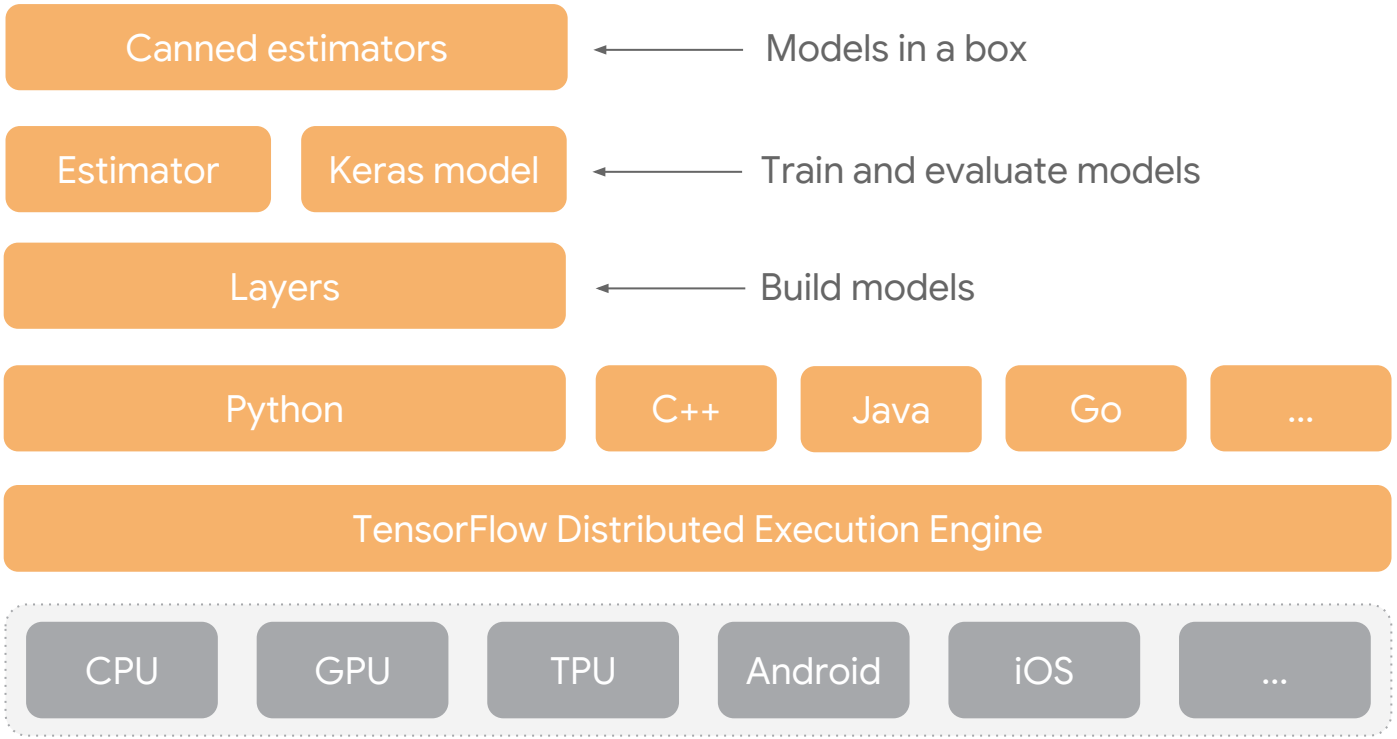
Machine Learning Pipeline with tf.estimator



Machine Learning Pipeline with tf.estimator



Estimator in TensorFlow



Example: MNIST

MNIST: Yann LeCunn

Steps to Run Training on Premade Estimator

Training:

1. Import data
2. Create input functions
3. Create estimator
4. Create train and evaluate specs
5. Run train and evaluate

Test Inference:

6. Test run predict()
7. Export model

Premade and Custom Estimators

Premade Estimators

DNNClassifier
DNNRegressor
LinearClassifier
LinearRegressor
DNNLinearCombinedClassifier
DNNLinearCombinedRegressor
+ tf.contrib.estimator

or

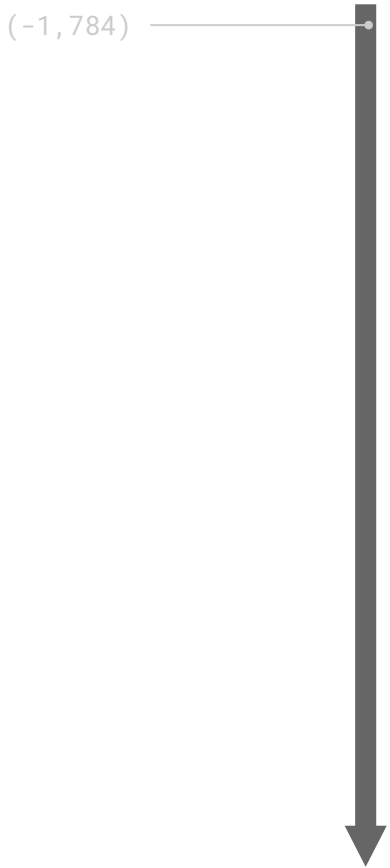
Custom Estimator

Creating Custom Estimator

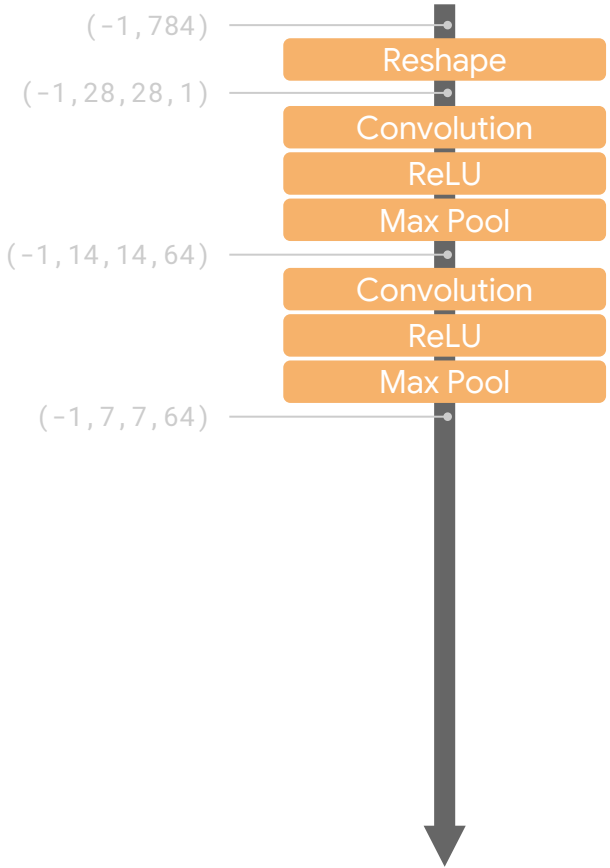
```
def custom_model_fn:
    #1 Inference()
    #2 Calculations and Metrics
    #3 MODE = PREDICT
    #4 MODE = TRAIN
    #5 MODE = EVAL
```

#1 Inference()

```
def custom_model_fn:
  #1 Inference()
  #2 Calculations
  #3 MODE = PREDICT
  #4 MODE = TRAIN
  #5 MODE = EVAL
```

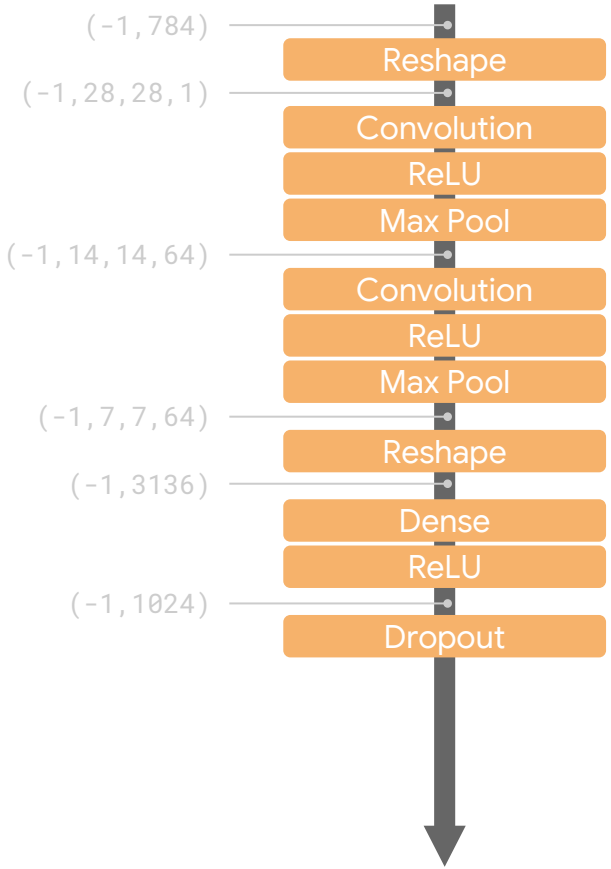


#1 Inference()



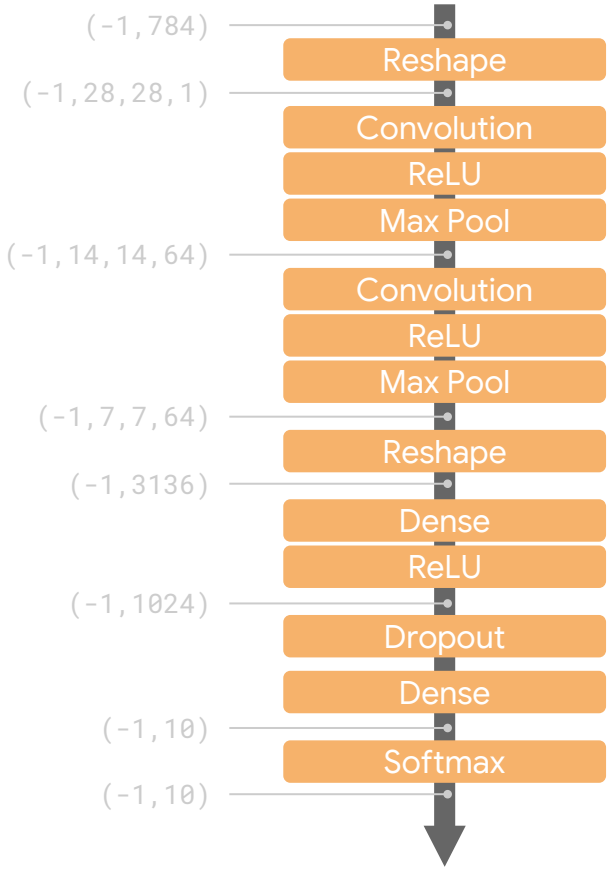

```
def custom_model_fn:
  #1 Inference()
  #2 Calculations
  #3 MODE = PREDICT
  #4 MODE = TRAIN
  #5 MODE = EVAL
```

#1 Inference()




```
def custom_model_fn:
  #1 Inference()
  #2 Calculations
  #3 MODE = PREDICT
  #4 MODE = TRAIN
  #5 MODE = EVAL
```

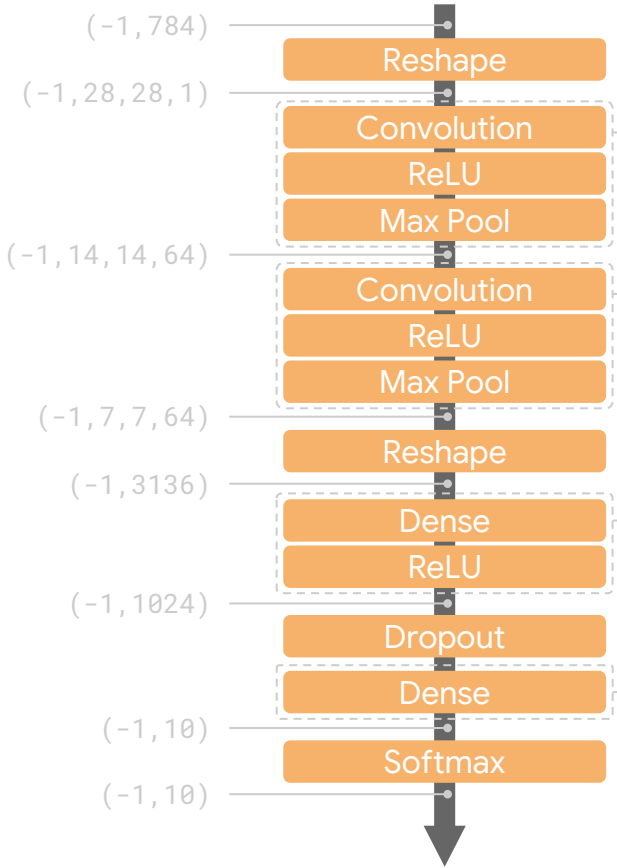
#1 Inference()




```
def custom_model_fn:
  #1 Inference()
  #2 Calculations
  #3 MODE = PREDICT
  #4 MODE = TRAIN
  #5 MODE = EVAL
```

#1 Inference()

```
def custom_model_fn:
  #1 Inference()
  #2 Calculations
  #3 MODE = PREDICT
  #4 MODE = TRAIN
  #5 MODE = EVAL
```




```
def _conv(x,kernel,name,log):
  with tf.name_scope(name):
    W = tf.Variable(tf.truncated_normal(shape=kernel,stddev=0.01),name='W')
    b = tf.Variable(tf.constant(0.0,shape=[kernel[3]]),name='b')
    conv = tf.nn.conv2d(x, W, strides=[1,1,1,1],padding='SAME')
    activation = tf.nn.relu(tf.add(conv,b))
    pool = tf.nn.max_pool(activation,ksize=[1,2,2,1],
                          strides=[1,2,2,1],padding='SAME')
    if log==True:
      tf.summary.histogram("weights",W)
      tf.summary.histogram("biases",b)
      tf.summary.histogram("activations",activation)
    return pool

def _dense(x,size_in,size_out,name,log,relu=False):
  with tf.name_scope(name):
    flat = tf.reshape(x,[-1,size_in])
    W = tf.Variable(tf.truncated_normal(
                    [size_in,size_out],stddev=0.1),name='W')
    b = tf.Variable(tf.constant(0.0,shape=[size_out]),name='b')
    activation = tf.add(tf.matmul(flat,W),b)
    if relu==True:
      activation = tf.nn.relu(activation)
    if log==True:
      tf.summary.histogram("weights",W)
      tf.summary.histogram("biases",b)
      tf.summary.histogram("activations",activation)
    return activation
```

#1 Inference() and #2 Calculations and Metrics

```
def custom_model_fn:
  #1 Inference()
  #2 Calculations
  #3 MODE = PREDICT
  #4 MODE = TRAIN
  #5 MODE = EVAL
```

```
#### 1 INFERENCE MODEL

input_layer = tf.reshape(features["x"], [-1, 28, 28, 1])
conv1 = _conv(input_layer,kernel=[5,5,1,64],name='conv1',log=params['log'])
conv2 = _conv(conv1,kernel=[5,5,64,64],name='conv2',log=params['log'])
dense = _dense(conv2,size_in=7*7*64,size_out=params['dense_units'],
               name='Dense',relu=True,log=params['log'])
if mode==tf.estimator.ModeKeys.TRAIN:
    dense = tf.nn.dropout(dense,params['drop_out'])
logits = _dense(dense,size_in=params['dense_units'],
                size_out=10,name='Output',relu=False,log=params['log'])

#### 2 CALCULATIONS AND METRICS

predictions = {"classes": tf.argmax(input=logits,axis=1),
               "logits": logits,
               "probabilities": tf.nn.softmax(logits,name='softmax')}
export_outputs = {'predictions': tf.estimator.export.PredictOutput(predictions)}
if (mode==tf.estimator.ModeKeys.TRAIN or mode==tf.estimator.ModeKeys.EVAL):
    loss = tf.losses.sparse_softmax_cross_entropy(labels=labels,logits=logits)
    accuracy = tf.metrics.accuracy(
        labels=labels, predictions=tf.argmax(logits,axis=1))
    metrics = {'accuracy':accuracy}
    tf.summary.scalar('accuracy',accuracy[1])
```

#1 Inference() and #2 Calculations and Metrics

```
def custom_model_fn:
  #1 Inference()
  #2 Calculations
  #3 MODE = PREDICT
  #4 MODE = TRAIN
  #5 MODE = EVAL
```

```
#### 1 INFERENCE MODEL

input_layer = tf.reshape(features["x"], [-1, 28, 28, 1])
conv1 = _conv(input_layer,kernel=[5,5,1,64],name='conv1',log=params['log'])
conv2 = _conv(conv1,kernel=[5,5,64,64],name='conv2',log=params['log'])
dense = _dense(conv2,size_in=7*7*64,size_out=params['dense_units'],
               name='Dense',relu=True,log=params['log'])
if mode==tf.estimator.ModeKeys.TRAIN:
    dense = tf.nn.dropout(dense,params['drop_out'])
logits = _dense(dense,size_in=params['dense_units'],
                size_out=10,name='Output',relu=False,log=params['log'])

#### 2 CALCULATIONS AND METRICS

predictions = {"classes": tf.argmax(input=logits,axis=1),
               "logits": logits,
               "probabilities": tf.nn.softmax(logits,name='softmax')}
export_outputs = {'predictions': tf.estimator.export.PredictOutput(predictions)}
if (mode==tf.estimator.ModeKeys.TRAIN or mode==tf.estimator.ModeKeys.EVAL):
    loss = tf.losses.sparse_softmax_cross_entropy(labels=labels,logits=logits)
    accuracy = tf.metrics.accuracy(
        labels=labels, predictions=tf.argmax(logits,axis=1))
    metrics = {'accuracy':accuracy}
    tf.summary.scalar('accuracy',accuracy[1])
```

#3 Predict, #4 Train, and #5 Eval

```
def custom_model_fn:
  #1 Inference()
  #2 Calculations
  #3 MODE = PREDICT
  #4 MODE = TRAIN
  #5 MODE = EVAL
```

```
#### 3 MODE = PREDICT

 if mode == tf.estimator.ModeKeys.PREDICT:
     return tf.estimator.EstimatorSpec(
         mode=mode, predictions=predictions, export_outputs=export_outputs)

 #### 4 MODE = TRAIN

 if mode == tf.estimator.ModeKeys.TRAIN:
     learning_rate = tf.train.exponential_decay(
         params['learning_rate'],tf.train.get_global_step(),
         decay_steps=100000,decay_rate=0.96)
     optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
     if params['replicate']==True:
         optimizer = tf.contrib.estimator.TowerOptimizer(optimizer)
     train_op = optimizer.minimize(loss=loss,global_step=tf.train.get_global_step())
     tf.summary.scalar('learning_rate', learning_rate)
     return tf.estimator.EstimatorSpec(
         mode=mode, loss=loss, train_op=train_op)

 #### 5 MODE = EVAL

 if mode == tf.estimator.ModeKeys.EVAL:
     return tf.estimator.EstimatorSpec(
         mode=mode,loss=loss,eval_metric_ops=metrics)
```

#3 Predict, #4 Train, and #5 Eval

```
def custom_model_fn:
  #1 Inference()
  #2 Calculations
  #3 MODE = PREDICT
  #4 MODE = TRAIN
  #5 MODE = EVAL
```

```
#### 3 MODE = PREDICT

 if mode == tf.estimator.ModeKeys.PREDICT:
     return tf.estimator.EstimatorSpec(
         mode=mode, predictions=predictions, export_outputs=export_outputs)

 #### 4 MODE = TRAIN

 if mode == tf.estimator.ModeKeys.TRAIN:
     learning_rate = tf.train.exponential_decay(
         params['learning_rate'],tf.train.get_global_step(),
         decay_steps=100000,decay_rate=0.96)
     optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
     if params['replicate']==True:
         optimizer = tf.contrib.estimator.TowerOptimizer(optimizer)
     train_op = optimizer.minimize(loss=loss,global_step=tf.train.get_global_step())
     tf.summary.scalar('learning_rate', learning_rate)
     return tf.estimator.EstimatorSpec(
         mode=mode, loss=loss, train_op=train_op)

 #### 5 MODE = EVAL

 if mode == tf.estimator.ModeKeys.EVAL:
     return tf.estimator.EstimatorSpec(
         mode=mode,loss=loss,eval_metric_ops=metrics)
```

#3 Predict, #4 Train, and #5 Eval

```
def custom_model_fn:
  #1 Inference()
  #2 Calculations
  #3 MODE = PREDICT
  #4 MODE = TRAIN
  #5 MODE = EVAL
```

```
#### 3 MODE = PREDICT

 if mode == tf.estimator.ModeKeys.PREDICT:
     return tf.estimator.EstimatorSpec(
         mode=mode, predictions=predictions, export_outputs=export_outputs)

 #### 4 MODE = TRAIN

 if mode == tf.estimator.ModeKeys.TRAIN:
     learning_rate = tf.train.exponential_decay(
         params['learning_rate'],tf.train.get_global_step(),
         decay_steps=100000,decay_rate=0.96)
     optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
     if params['replicate']==True:
         optimizer = tf.contrib.estimator.TowerOptimizer(optimizer)
     train_op = optimizer.minimize(loss=loss,global_step=tf.train.get_global_step())
     tf.summary.scalar('learning_rate', learning_rate)
     return tf.estimator.EstimatorSpec(
         mode=mode, loss=loss, train_op=train_op)

 #### 5 MODE = EVAL

 if mode == tf.estimator.ModeKeys.EVAL:
     return tf.estimator.EstimatorSpec(
         mode=mode,loss=loss,eval_metric_ops=metrics)
```

TFRecords and tf.data

(Inefficient) Pipeline

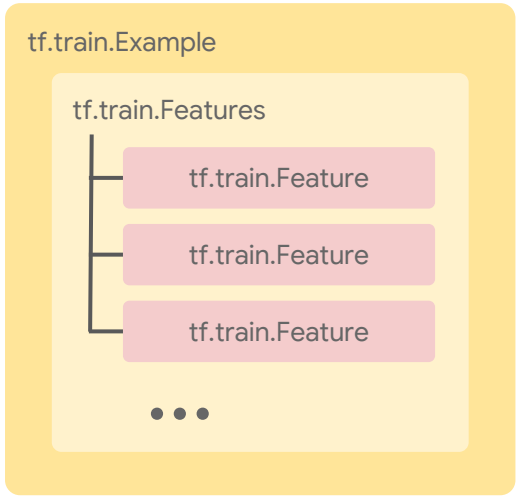
```
# Load Data
mnist = tf.contrib.learn.datasets.load_dataset("mnist")
train_data = mnist.train.images
train_labels = np.asarray(mnist.train.labels, dtype=np.int32)
eval_data = mnist.test.images
eval_labels = np.asarray(mnist.test.labels, dtype=np.int32)

# Create Input Functions
train_input_fn = tf.estimator.inputs.numpy_input_fn(
    x={"x": train_data},y=train_labels,batch_size=100,num_epochs=None,shuffle=True)
eval_input_fn  = tf.estimator.inputs.numpy_input_fn(
    x={"x": eval_data},y=eval_labels,num_epochs=1,shuffle=False)
pred_input_fn  = tf.estimator.inputs.numpy_input_fn(
    x={"x": eval_data},num_epochs=1,shuffle=False)
```

TFRecord


tf.train.Example
tf.train.Example
tf.train.Example
tf.train.Example
tf.train.Example
train004.tfrecords

TFRecord

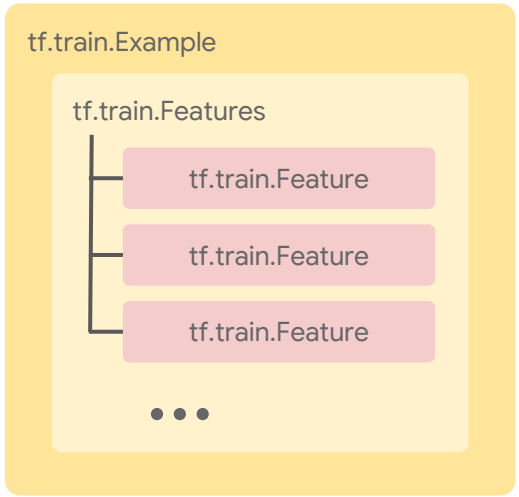



```
writer = tf.python_io.TFRecordWriter('train004.tfrecords')

def _int64_feature(value):
  return tf.train.Feature(
    int64_list=tf.train.Int64List(value=[value]))
def _bytes_feature(value):
  return tf.train.Feature(
    bytes_list=tf.train.BytesList(value=[value]))

# Parallelize:
e = tf.train.Example(features=tf.train.Features(feature={
    'idx'      : _int64_feature(idx),
    'label'    : _int64_feature(label),
    'image'    : _bytes_feature(image.tostring())
    }))
writer.write(e.SerializeToString())
```

tf.data




```
# Deserialize Example
def parse_tfrecord(example):
  feature={'idx'   :tf.FixedLenFeature((),tf.int64),
           'label':tf.FixedLenFeature((),tf.int64),
           'Image':tf.FixedLenFeature((),tf.string,default_value='')}
  parsed = tf.parse_single_example(example,feature)
  image = tf.decode_raw(parsed['image'],tf.float32)
  image = tf.reshape(image,[64,64,3])
  return image, parsed['label']

# Mean subtraction/feature scaling:
def image_scaling(x):
  return tf.image.per_image_standardization(x)

# Data augmentation:
def distort(x):
  x = tf.image.resize_image_with_crop_or_pad(x,80,80)
  x = tf.random_crop(x,[64,64,3])
  x = tf.image.random_flip_left_right(x)
  return x
```

tf.data input_fn()

```
def dataset_input_fn(c):
  dataset = tf.data.TFRecordDataset(c['files'],num_parallel_reads=c['threads'])
  dataset = dataset.map(parse_tfrecord,num_parallel_calls=c['threads'])
  dataset = dataset.map(lambda x,y: (image_scaling(x),y),num_parallel_calls=c['threads'])
  if c['mode']==tf.estimator.ModeKeys.TRAIN:
    dataset = dataset.map(lambda x,y: (distort(x),y),num_parallel_calls=c['threads'])
    dataset = dataset.shuffle(buffer_size=c['shuffle_buff'])
  dataset = dataset.repeat()
  dataset = dataset.batch(c['batch'])
  dataset = dataset.prefetch(2*c['batch'])
  iterator = dataset.make_one_shot_iterator()
  images, labels = iterator.get_next()
  features = {'images': images}
  return features, labels

train_spec = tf.estimator.TrainSpec(input_fn=lambda: dataset_input_fn(train_params))
eval_spec  = tf.estimator.EvalSpec(input_fn=lambda: dataset_input_fn(eval_params))
```

tf.data input_fn()

```
def dataset_input_fn(c):
  dataset = tf.data.TFRecordDataset(c['files'],num_parallel_reads=c['threads'])
  dataset = dataset.map(parse_tfrecord,num_parallel_calls=c['threads'])
  dataset = dataset.map(lambda x,y: (image_scaling(x),y),num_parallel_calls=c['threads'])
  if c['mode']==tf.estimator.ModeKeys.TRAIN:
    dataset = dataset.map(lambda x,y: (distort(x),y),num_parallel_calls=c['threads'])
    dataset = dataset.shuffle(buffer_size=c['shuffle_buff'])
  dataset = dataset.repeat()
  dataset = dataset.batch(c['batch'])
  dataset = dataset.prefetch(2*c['batch'])
  iterator = dataset.make_one_shot_iterator()
  images, labels = iterator.get_next()
  features = {'images': images}
  return features, labels

train_spec = tf.estimator.TrainSpec(input_fn=lambda: dataset_input_fn(train_params))
eval_spec  = tf.estimator.EvalSpec(input_fn=lambda: dataset_input_fn(eval_params))
```

tf.data input_fn()

```
def dataset_input_fn(c):
  dataset = tf.data.TFRecordDataset(c['files'],num_parallel_reads=c['threads'])
  dataset = dataset.map(parse_tfrecord,num_parallel_calls=c['threads'])
  dataset = dataset.map(lambda x,y: (image_scaling(x),y),num_parallel_calls=c['threads'])
  if c['mode']==tf.estimator.ModeKeys.TRAIN:
    dataset = dataset.map(lambda x,y: (distort(x),y),num_parallel_calls=c['threads'])
    dataset = dataset.shuffle(buffer_size=c['shuffle_buff'])
  dataset = dataset.repeat()
  dataset = dataset.batch(c['batch'])
  dataset = dataset.prefetch(2*c['batch'])
  iterator = dataset.make_one_shot_iterator()
  images, labels = iterator.get_next()
  features = {'images': images}
  return features, labels

train_spec = tf.estimator.TrainSpec(input_fn=lambda: dataset_input_fn(train_params))
eval_spec  = tf.estimator.EvalSpec(input_fn=lambda: dataset_input_fn(eval_params))
```

tf.data input_fn()

```
def dataset_input_fn(c):
  dataset = tf.data.TFRecordDataset(c['files'],num_parallel_reads=c['threads'])
  dataset = dataset.map(parse_tfrecord,num_parallel_calls=c['threads'])
  dataset = dataset.map(lambda x,y: (image_scaling(x),y),num_parallel_calls=c['threads'])
  if c['mode']==tf.estimator.ModeKeys.TRAIN:
    dataset = dataset.map(lambda x,y: (distort(x),y),num_parallel_calls=c['threads'])
    dataset = dataset.shuffle(buffer_size=c['shuffle_buff'])
  dataset = dataset.repeat()
  dataset = dataset.batch(c['batch'])
  dataset = dataset.prefetch(2*c['batch'])
  iterator = dataset.make_one_shot_iterator()
  images, labels = iterator.get_next()
  features = {'images': images}
  return features, labels

train_spec = tf.estimator.TrainSpec(input_fn=lambda: dataset_input_fn(train_params))
eval_spec  = tf.estimator.EvalSpec(input_fn=lambda: dataset_input_fn(eval_params))
```

tf.data input_fn()

```
def dataset_input_fn(c):
  dataset = tf.data.TFRecordDataset(c['files'],num_parallel_reads=c['threads'])
  dataset = dataset.map(parse_tfrecord,num_parallel_calls=c['threads'])
  dataset = dataset.map(lambda x,y: (image_scaling(x),y),num_parallel_calls=c['threads'])
  if c['mode']==tf.estimator.ModeKeys.TRAIN:
    dataset = dataset.map(lambda x,y: (distort(x),y),num_parallel_calls=c['threads'])
    dataset = dataset.shuffle(buffer_size=c['shuffle_buff'])
  dataset = dataset.repeat()
  dataset = dataset.batch(c['batch'])
  dataset = dataset.prefetch(2*c['batch'])
  iterator = dataset.make_one_shot_iterator()
  images, labels = iterator.get_next()
  features = {'images': images}
  return features, labels

train_spec = tf.estimator.TrainSpec(input_fn=lambda: dataset_input_fn(train_params))
eval_spec  = tf.estimator.EvalSpec(input_fn=lambda: dataset_input_fn(eval_params))
```

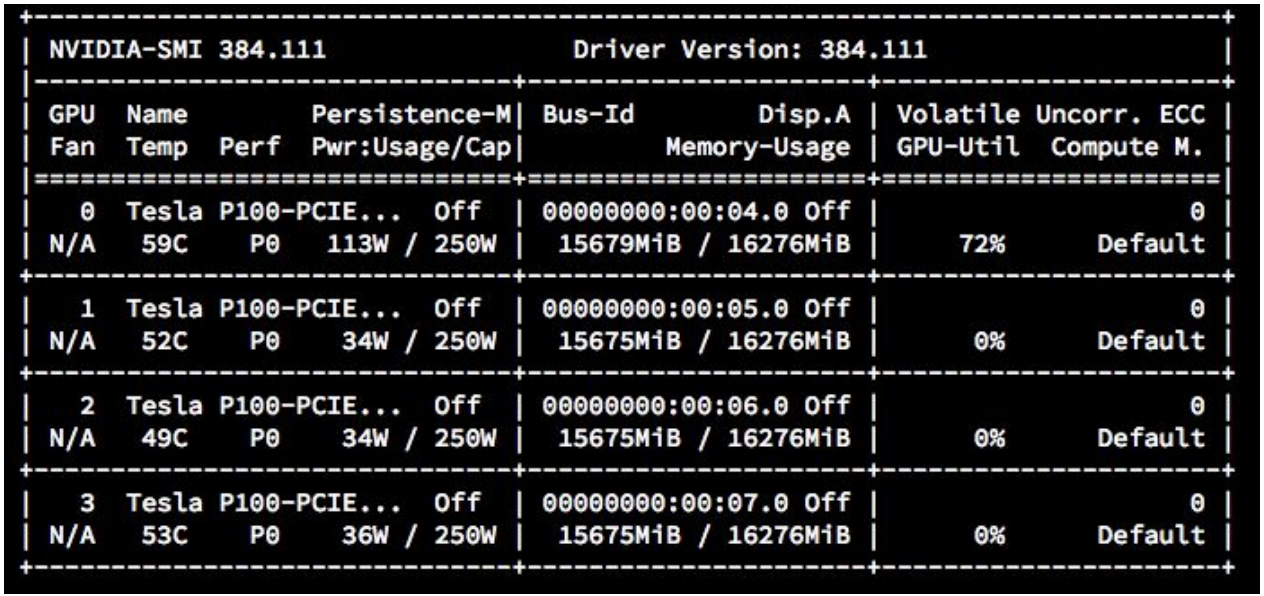
tf.data input_fn()

```
def dataset_input_fn(c):
  dataset = tf.data.TFRecordDataset(c['files'],num_parallel_reads=c['threads'])
  dataset = dataset.map(parse_tfrecord,num_parallel_calls=c['threads'])
  dataset = dataset.map(lambda x,y: (image_scaling(x),y),num_parallel_calls=c['threads'])
  if c['mode']==tf.estimator.ModeKeys.TRAIN:
    dataset = dataset.map(lambda x,y: (distort(x),y),num_parallel_calls=c['threads'])
    dataset = dataset.shuffle(buffer_size=c['shuffle_buff'])
  dataset = dataset.repeat()
  dataset = dataset.batch(c['batch'])
  dataset = dataset.prefetch(2*c['batch'])
  iterator = dataset.make_one_shot_iterator()
  images, labels = iterator.get_next()
  features = {'images': images}
  return features, labels

train_spec = tf.estimator.TrainSpec(input_fn=lambda: dataset_input_fn(train_params))
eval_spec  = tf.estimator.EvalSpec(input_fn=lambda: dataset_input_fn(eval_params))
```

Distributed Training

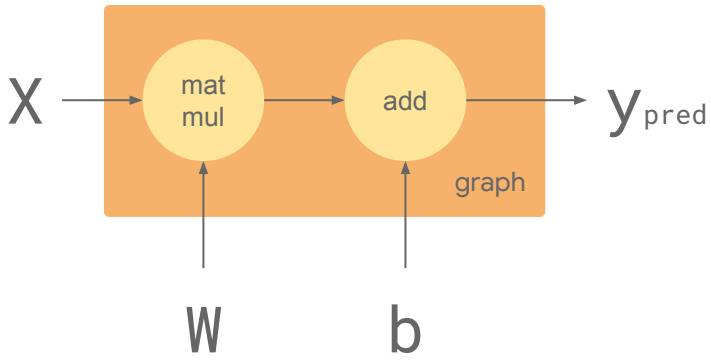
Parallelism is Not Automatic



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.111                 Driver Version: 384.111                  |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   59C    P0   113W / 250W |  15679MiB / 16276MiB |     72%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-PCIE...  Off  | 00000000:00:05.0 Off |                    0 |
| N/A   52C    P0    34W / 250W |  15675MiB / 16276MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla P100-PCIE...  Off  | 00000000:00:06.0 Off |                    0 |
| N/A   49C    P0    34W / 250W |  15675MiB / 16276MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla P100-PCIE...  Off  | 00000000:00:07.0 Off |                    0 |
| N/A   53C    P0    36W / 250W |  15675MiB / 16276MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
```

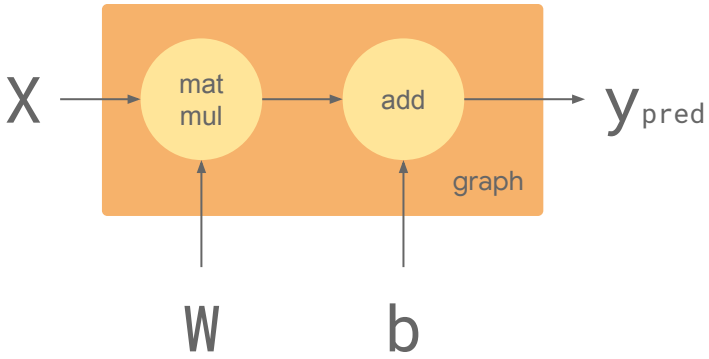
Data and Model Parallelism

$$y_{pred} = WX + b$$

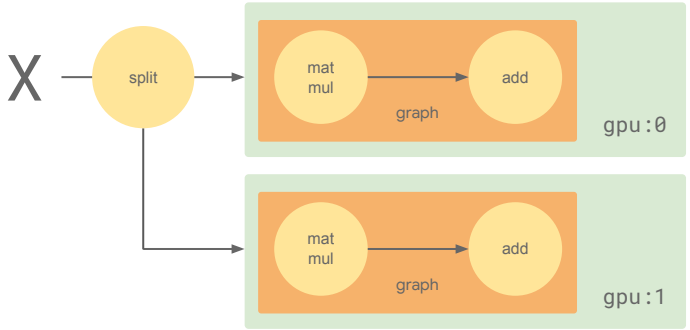


Data and Model Parallelism

$$y_{pred} = WX + b$$

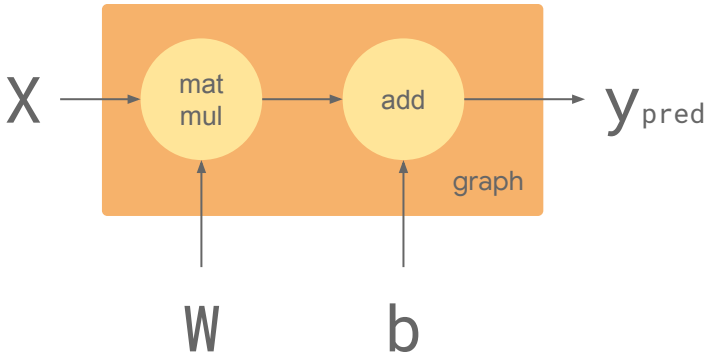


Data Parallelism

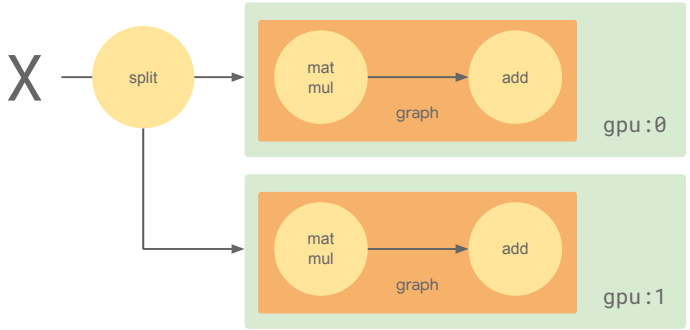


Data and Model Parallelism

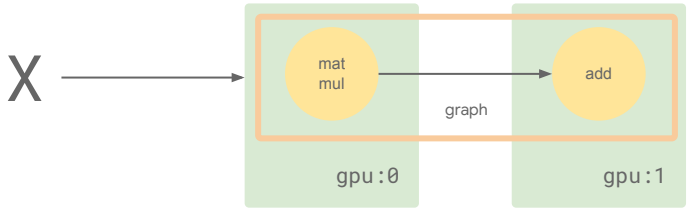
$$y_{pred} = WX + b$$



Data Parallelism

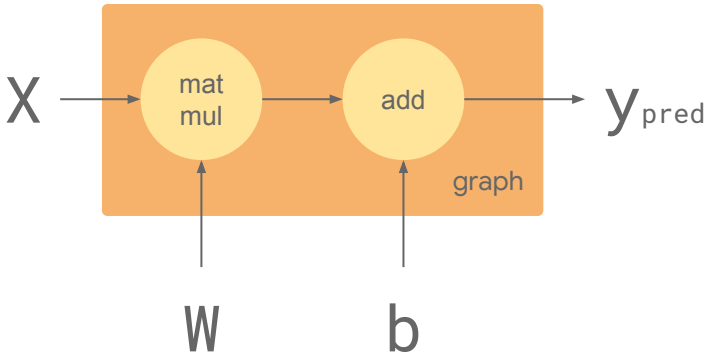


Model Parallelism

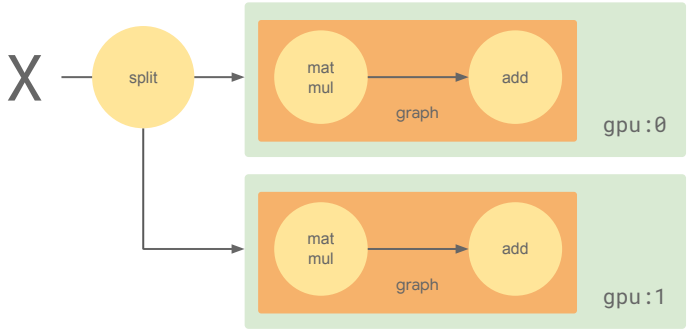


Data and Model Parallelism

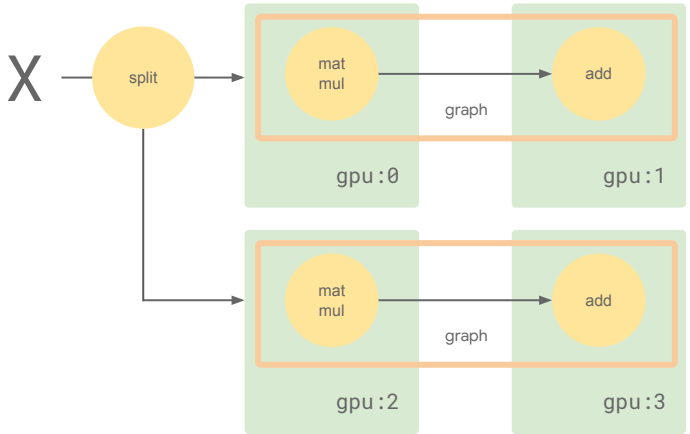
$$y_{pred} = WX + b$$



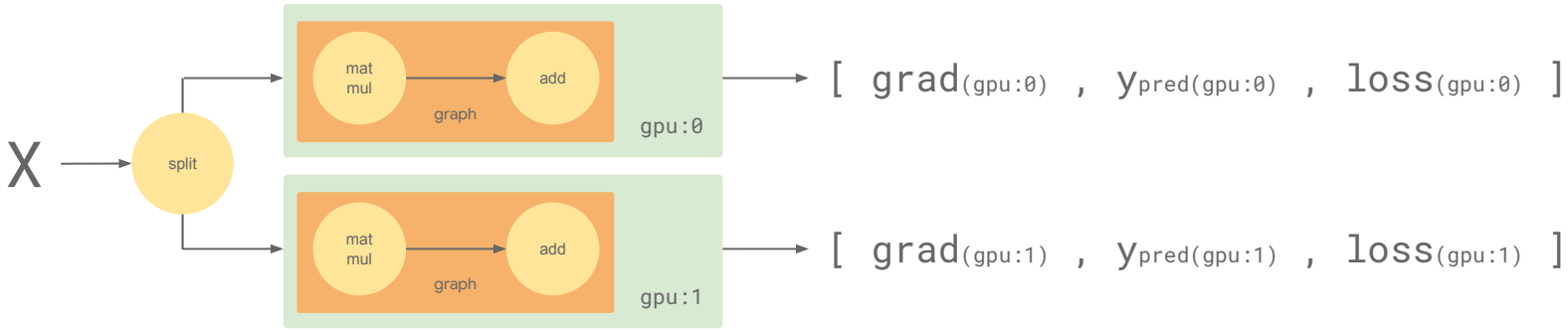
Data Parallelism



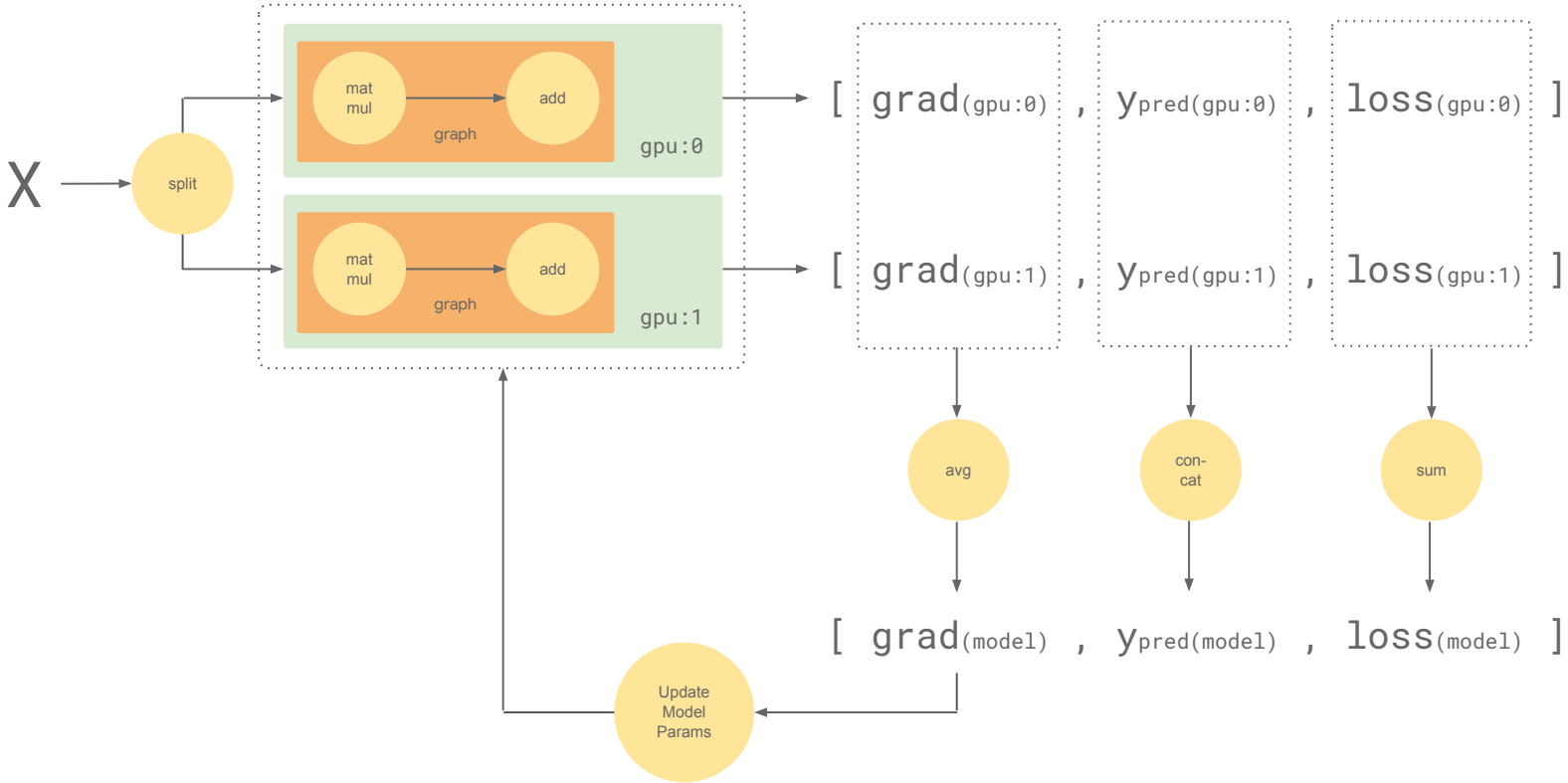
Model Parallelism



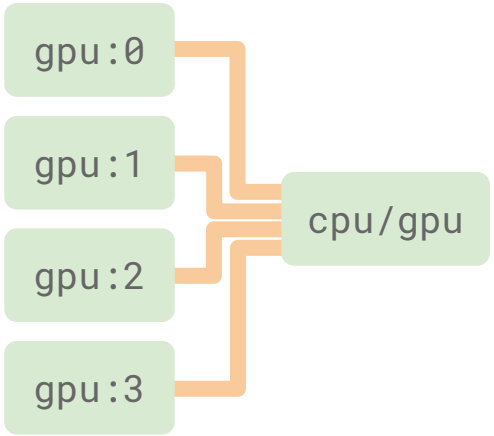
Output Reduction



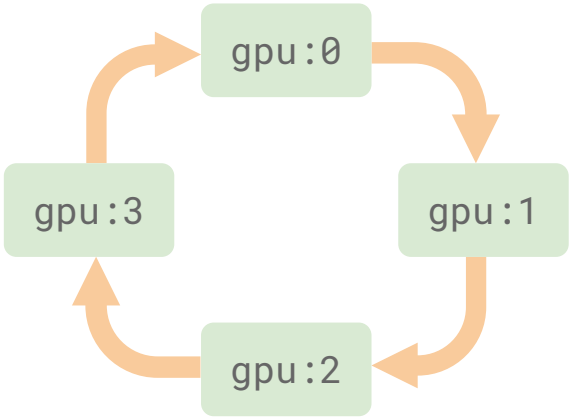
Output Reduction



Output Reduction



Naive Consolidation



Ring All-Reduce

Baidu All-Reduce: Andrew Ng

Distributed Accelerator Options

replicate_model_fn (TF1.5+; Naive Consolidation; Single Server)

```
optimizer = tf.contrib.estimator.TowerOptimizer(optimizer)  # in model_fn()

cnn_model_fn = tf.contrib.estimator.replicate_model_fn(
  cnn_model_fn, loss_reduction=tf.losses.Reduction.MEAN)
```

distribute.MirroredStrategy (TF1.8+; All-Reduce; Single Server for now)

```
distribution = tf.contrib.distribute.MirroredStrategy(num_gpus=4)

config = tf.estimator.RunConfig(distribute = distribution)

cnn_model = tf.estimator.Estimator(model_fn=model_fn,model_dir=model_dir,params=params,config=config)
```

TPUEstimator (TF1.4+; All-Reduce; 64-TPU pod)

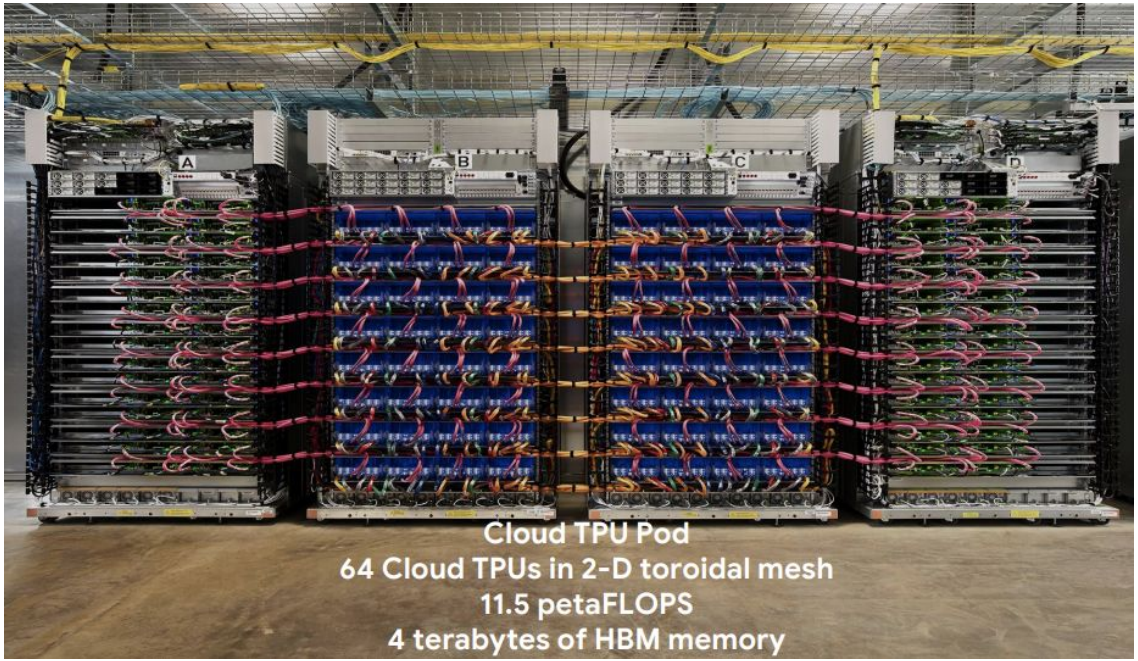
```
optimizer = tf.contrib.tpu.CrossShardOptimizer(optimizer)  # in model_fn()
```

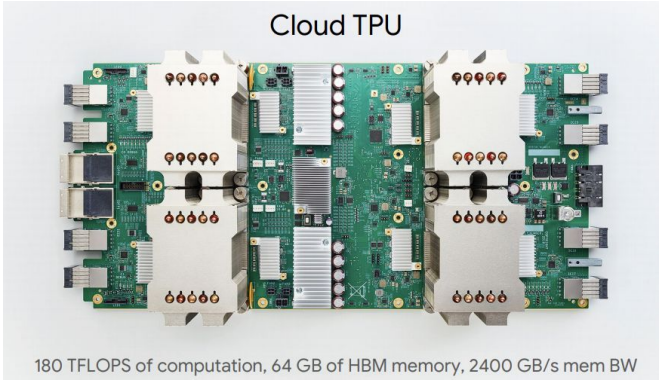
Horovod (TF1.8+; All-Reduce; Multi Server)

multi_gpu_model (Keras; Naive Consolidation; Single Server)

Meet Horovod: Alex Sergeev, Mike Del Balso

TPU / TPU Pod





TPUs	Batch Size	Time to 90 Epochs	Accuracy
1	256	23:22	76.6%
4	1024	05:48	76.3%
16	4096	01:30	76.5%
32	8192	45 min	76.1%
64	16384	22 min	75.0%
64	8192 ->16384	29.5 min	76.1%

ImageNet is the New MNIST

Summary

tf.estimator

- Premade Estimators for quick and dirty jobs (5-part recipe)
- Custom models w/o the plumbing (5-part function)
- Keras Functional API for Inference() in custom model_fn()
- Use Checkpoints/Preemptible VMs/Cloud Storage to reduce cost
- Scalable model building from laptop to GPUs to TPU Pods

TFRecords & tf.data for pipeline performance

- Parallelize creation of TFRecord files ~ 100-150MB
- Use tf.data input_fn()

Distributed Accelerator options

- Input pipeline before bigger/faster/more accelerators
- Scale Up before Out

Thank you.

goto; chicago

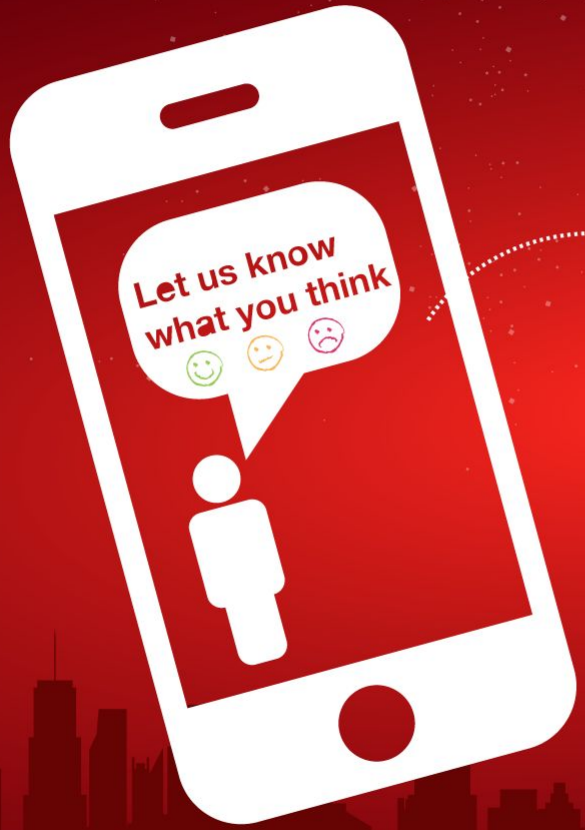
Please

Remember to rate this session

Thank you!

follow us @gotochgo

goto; chicago



Click ‘Rate Session’

Rate **5** sessions to get the supercool GOTO reward

follow us @gotochgo