IIT Kharagpur | IIT Madras | IIT Goa | IIT Palakkad

APPLIED ACCELERATED ARTIFICIAL INTELLIGENCE

TensorFlow

Dr. Satyajit Das
Assistant Professor
Data Science
Computer Science and Engineering
IIT Palakkad

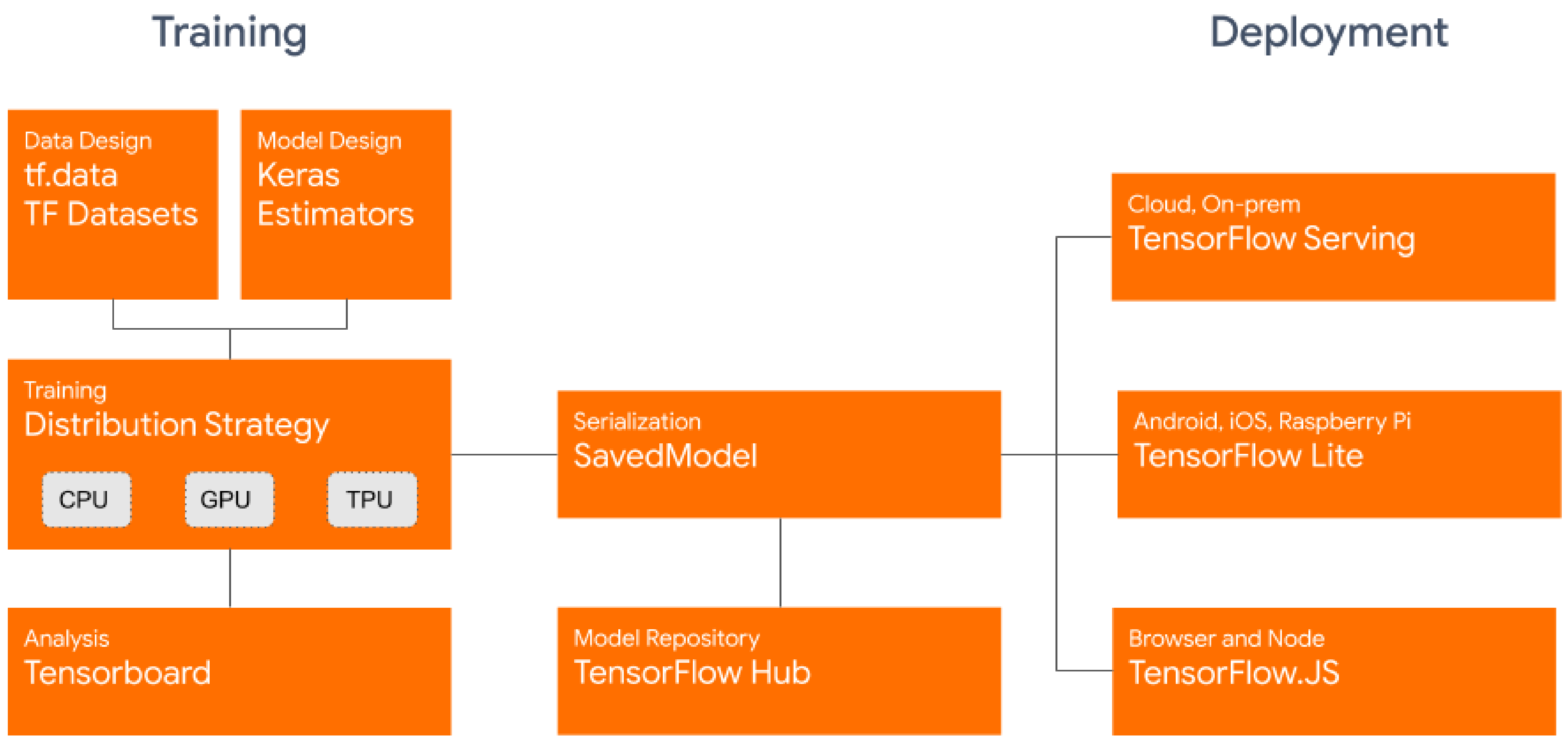
National Supercomputing Mission

Centre for Development of Advanced Computing

TensorFlow 1.x/2.x

- An open source Deep Learning library
  - >1,800 contributors worldwide
  - Apache 2.0 license
  - Released by Google in 2015

- TensorFlow 2.0
  - Easier to learn and use
  - For beginners and experts
  - Available today



https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html

The workflow

Data Ingestion and Transformation	→	Model Building	→	Training	→	Saving
tf.data Feature Columns		Keras Premade Estimators Custom		Eager Execution Autograph Distribution Strategy Tensorboard		Saved Model

What’s new in TensorFlow 2.x

- Easy model building with **Keras and eager execution** (activated by default in TF2.0).
- Robust model **deployment** in production on any platform.
- Powerful experimentation for research.
- Simplifying the API by **cleaning up deprecated APIs** and reducing duplication (relevant in case you have code developed in TensorFlow 1.X and you need to convert it)
- Load your data using **tf.data**. Training data is read using input pipelines which are created using tf.data.
- Build, train and validate your model with **tf.keras**, or use **Premade Estimators**.
- **TensorFlow Hub**.
- **Run and debug with eager execution**, then use tf.function for the benefits of graphs.
- Use Distribution Strategies for distributed training.
- hardware accelerators like CPUs, GPUs, and TPUs; you can enable training workloads to be distributed to single-node/multi-accelerator as well as multi-node/multi-accelerator configurations, including TPU Pods.
- Export to **SavedModel**. TensorFlow will standardize on SavedModel as an interchange format for TensorFlow Serving, TensorFlow Lite, TensorFlow.js, TensorFlow Hub, and more.
- Tensorflow Datasets

Model Building

- Sequential API + built in layers — New users, simple models
- Functional API + built in layers — Engineers with standard use cases
- Functional API + Custom layers + Custom Metrics + Custom losses — Engineers requiring increasing control
- Subclassing:Write everything yourself from scratch — Researchers

Symbolic vs Imperative APIs

- Symbolic (Keras Sequential)
  - Your model is a graph of layers
  - Any graph you compile will run
  - TensorFlow helps you debug by catching errors at compile time

Symbolic vs Imperative APIs

- Symbolic (Keras Sequential)
  - Your model is a graph of layers
  - Any graph you compile will run
  - TensorFlow helps you debug by catching errors at compile time
- Imperative (Keras Subclassing)
  - Your model is Python bytecode
  - Complete flexibility and control
  - Harder to debug / harder to maintain

Model Training

model.fit()	model.fit()+ Callbacks	model. train_on_batch(), + callbacks	Custom training loop with GradientTape
Quick experiment	Customize your training loop Add checkpointing, early stopping, TensorBoard monitoring, send Slack notifications...	Custom training loop using built-in optimizers and losses e.g. GANs	Complete control e.g. new optimization algorithm; easily modify gradients as you go.

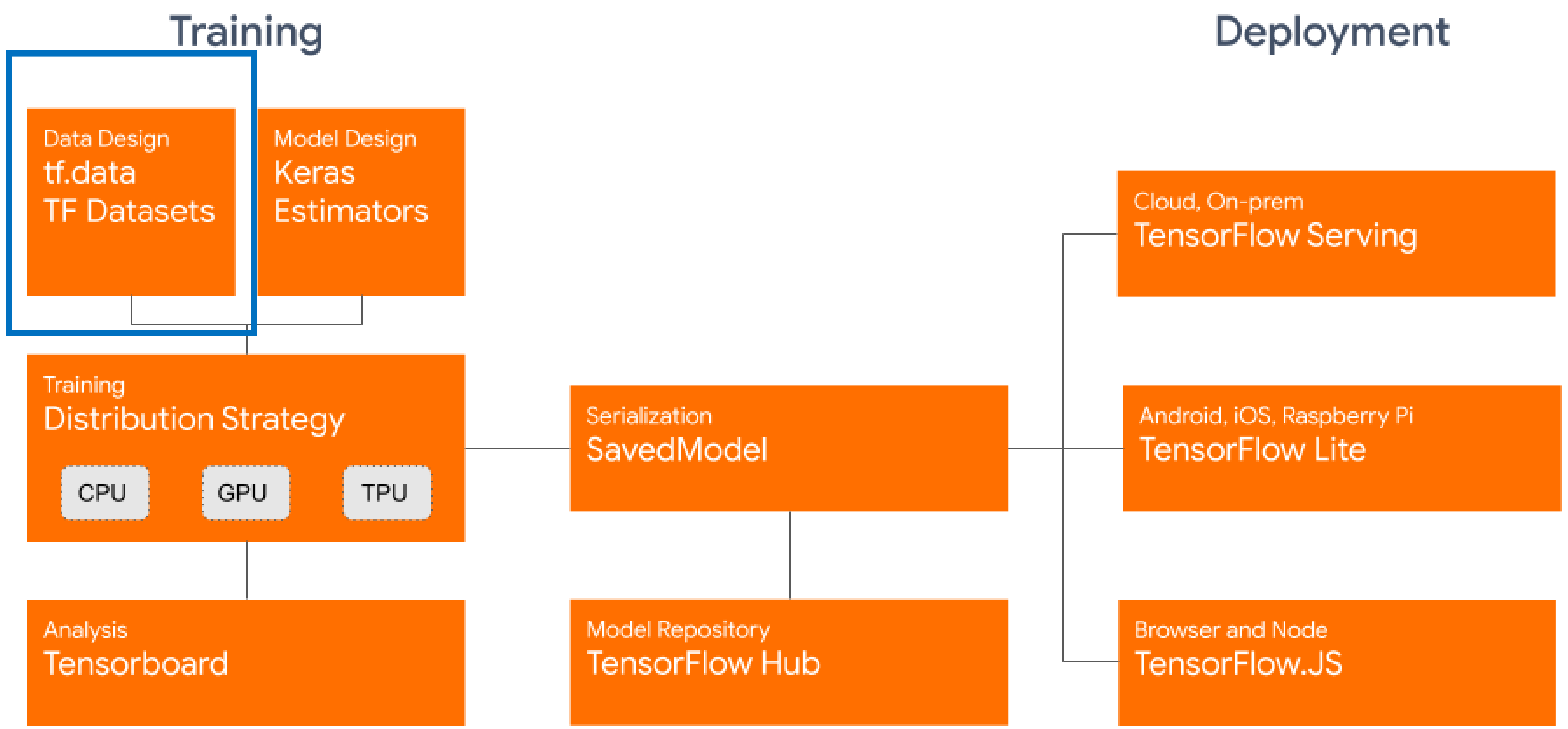
Tensor datatype

- A Tensor has a name, type a rank and a shape
  - The name identifies uniquely the object in the computational graph
  - The type specify the data type, for example tf.float32 or tf.int8.
  - The rank is simply the number of dimensions of the tensor: a scalar has rank 0, a vector has rank 1 and so on.
  - The shape is the number of elements in each dimensions: a scalar has a shape of (), a vector a shape of (d0), a matrix shape of (d0,d1) and so on (with d0 and d1 positive integers).
  - NOTE: the dimension None is allowed and indicates an unknown dimension.

Main tensor types

- tf.Variable* - will change during training; a variable maintains state in the graph across calls to run()
- tf.constant** - will remain constant

https://www.tensorflow.org/guide/tensor



https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html

```
# Keras datasets
from tensorflow.keras import datasets (train_images,
train_labels), \
 (test_images, test_labels) = datasets.cifar10.load_data()

# TensorFlow Datasets
import tensorflow_datasets as tfds
dataset, metadata = tfds.load('cycle_gan/horse2zebra',
                    with_info=True,  as_supervised=True)
```

Helpful reference (on tf.data, loading images, and caching): tensorflow.org/tutorials/load_data/images
List of TensorFlow Datasets: tensorflow.org/datasets/catalog/overview

```
# If you're using TensorFlow Datasets
# Either load your dataset into memory, or
# write a performant input pipeline to load it off disk.
dataset, metadata = tfds.load('mnist',
                              with_info=True,  as_supervised=True,
                              in_memory=True)
```

Credit: Josh Gordon

```
# Caching is important to avoid repeated work
# Use either an in-memory cache, or a cache file
def preprocess(img):

  img = tf.cast(image, tf.float32)  img = (img / 127.5) - 1
  img = tf.image.resize(img, [286, 286]) # ...
  return img

image_ds = image_ds.map(
    preprocess, num_parallel_calls=AUTOTUNE).cache()
```

Note: order is important. Cache before shuffling and batching.

Helpful reference (on tf.data, loading images, and caching): tensorflow.org/tutorials/load_data/images
List of TensorFlow Datasets: tensorflow.org/datasets/catalog/overview

Credit: Josh Gordon

DEMO

[demo1] https://colab.research.google.com/drive/1BbMLpUS5-9vnee3DEBq4DKMD2h-cx3cc#scrollTo=4N7XbNDVY8P3

[demo 2] https://colab.research.google.com/drive/1U1R4fntlQzN93e0WSANgwGtHczwXIbR_#scrollTo=-HJV4JF789aC

[demo 3] https://colab.research.google.com/drive/11AnQ39sHsuUkEC7Lg5__-lT2SLrIElA8#scrollTo=Y04m-jvKRDsJ

Thank You