Open Source Project Documentation

Date: 2-Dec-2022

To : Sir. Jawad Cheema

Written By:

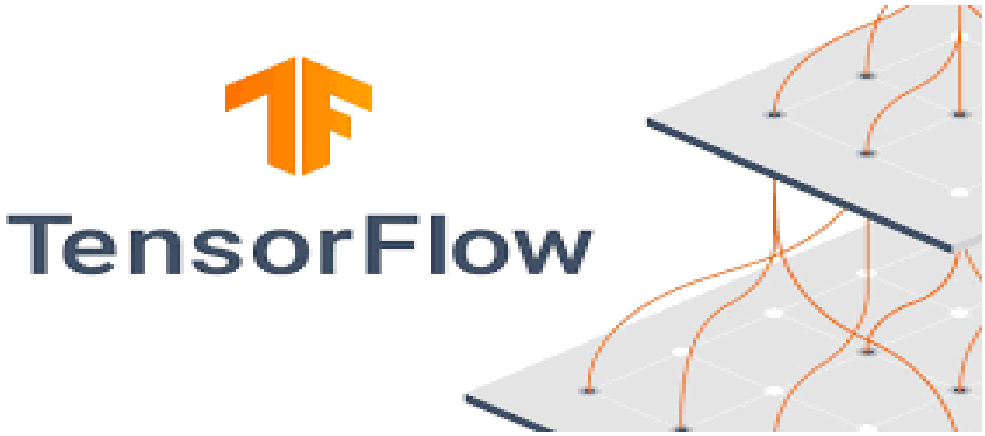
(Testers)

Abdullah Ahmad(20L-1326)

Hassan Arif(20L-1262)

Asad Ghouri(20L-1334)

TENSOR FLOW



1)What is Tensor flow?

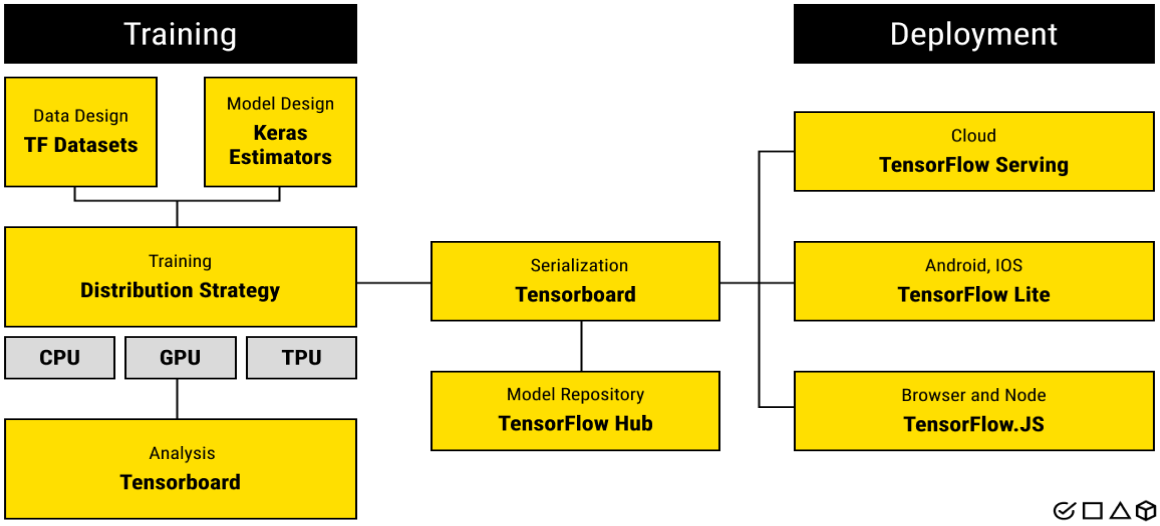
TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow makes it easy for beginners and experts to create machine learning models for desktop, mobile, web, and cloud.

1)What is Core of Tensor flow?

The TensorFlow Core APIs **provide a set of comprehensive, composable, and extensible low-level APIs for high-performance (distributed & accelerated) computation**, primarily aimed at building machine learning (ML) models as well as authoring ML workflow tools and frameworks within the TensorFlow platform.

Documentation

Modularity Requirements:



Modular TensorFlow:

TensorFlow is a very successful open source project. Since it has been open sourced, 1800+ contributors have submitted code into TF from outside Google. However, as more and more developers contribute, it becomes more and more difficult to manage contributions in the single repository.

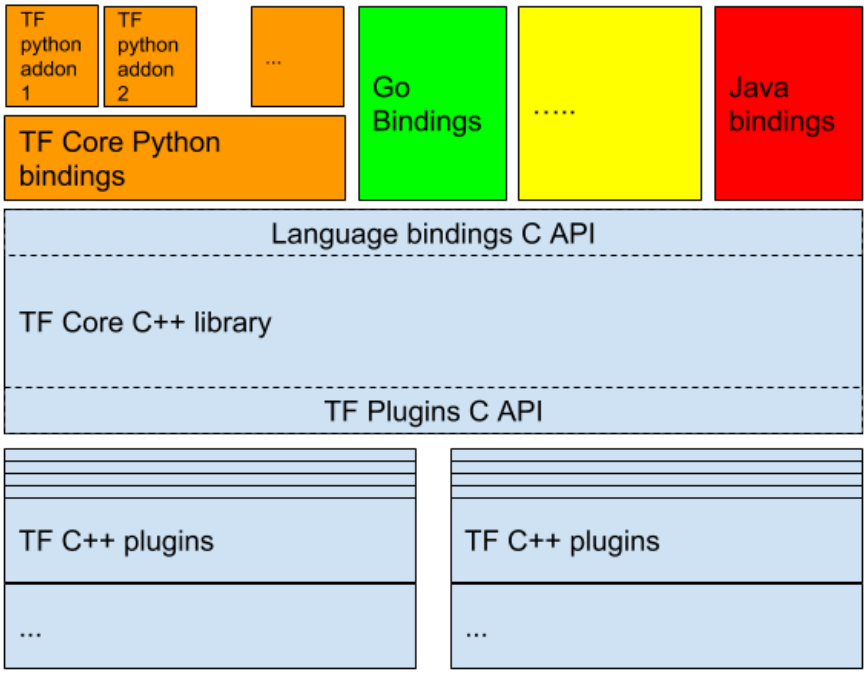
This project aims to split the TensorFlow codebase into smaller, more focused, repositories that can be released and managed separately. These modules will talk to each other using well defined APIs. Thanks to the module APIs, these modules are now managed/owned/released independently.

Overview:

This project aims to split the TensorFlow codebase into smaller, more focused, repositories that can be released and managed separately. These modules will talk to each other using well defined APIs that will evolve over time. Thanks to these APIs, these modules will be managed/owned/released independently. There will be

different strategies to break apart pieces based on the languages, but below summarizes the approach for C++ and Python:

1) Core TF functionality will be implemented in C++
2) Core TF functionality can be extended using shared objects.
3) On top of the core C++ libraries, we will have the language bindings (Using the C API)
4) There can be more functionality built on top of the core TF bindings in different languages, which can be maintained and distributed separately.
5) All different pieces need to use well defined public APIs.



Modules:

To do machine learning in TensorFlow, you are likely to need to define, save, and restore a model.

A model is, abstractly:

- A function that computes something on tensors (a **forward pass**)
- Some variables that can be updated in response to training

Defining Models and layers in Tensor Flow:

Most models are made of layers. Layers are functions with a known mathematical structure that can be reused and have trainable variables. In TensorFlow, most high-level implementations of layers and models, such as Keras or Sonnet, are built on the same foundational class: tf.Module.

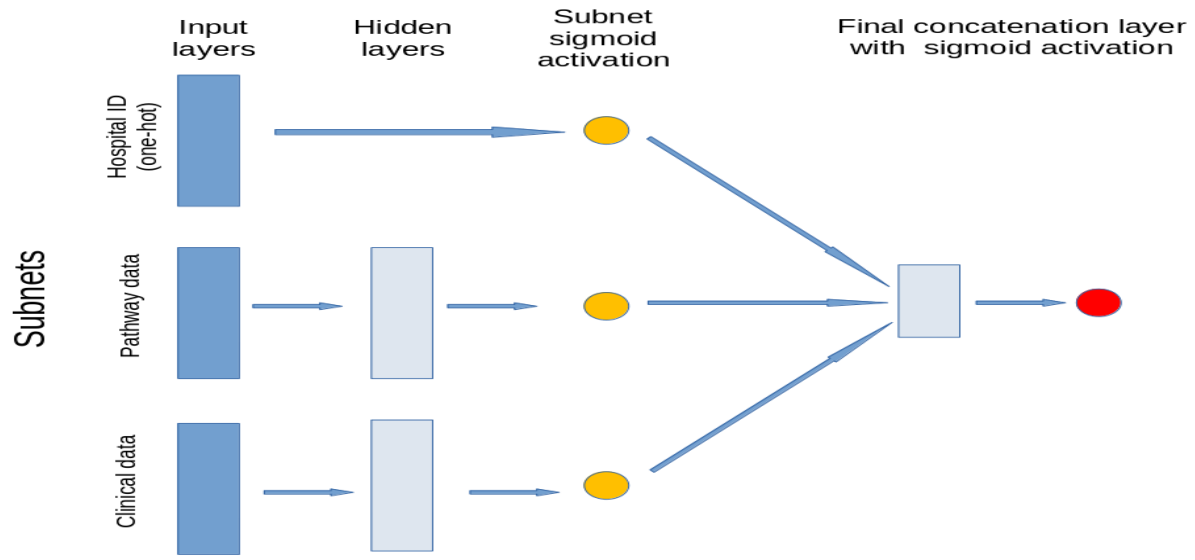
Code:

```
 class SimpleModule(tf.Module):
 def __init__(self, name=None):
 super().__init__(name=name)
 self.a_variable = tf.Variable(5.0, name="train_me")
self.non_trainable_variable = tf.Variable(5.0,
trainable=False, name="do_not_train_me")
 def __call__(self, x):
 return self.a_variable * x + self.non_trainable_variable
simple_module = SimpleModule(name="simple")
simple_module(tf.constant(5.0))
```

Modules and, by extension, layers are deep-learning terminology for "objects": they have internal state, and methods that use that state.

Models and layers can be loaded without actually making an instance of the class that created it. This is desired in situations where you do not have (or want) a Python interpreter, such as serving at scale or on an edge device, or in situations where the original Python code is not available or practical to use.

Modular Tensor flow with 2D embedding



Reusability:

Tensor flow provide facility of reuse parts of machine learning modules. By a module, we mean a self-contained piece of a TensorFlow graph, along with its weights, that can be reused across other, similar tasks. By reusing a module, a developer can train a model using a smaller dataset, improve generalization, or simply speed up training. Let’s look at a couple examples to make this concrete.

TensorFlow Hub hosts Saved Models for TensorFlow 2, among other assets. They can be loaded back into a Python program with obj = hub .load (url). The returned obj is the result of tf. saved model. load(). This object can have arbitrary attributes that are tf. functions, tf. Variables (initialized from their pre-trained values), other resources and, recursively, more such objects.

An interface to be implemented by the loaded obj in order to be *reused* in a TensorFlow Python program. Saved Models conforming to this interface are called ***Reusable* Saved Models**.

Reusing means building a larger model around obj, including the ability to fine-tune it. Fine-tuning means further training of the weights in the loaded obj as part of the surrounding model. The loss function and the optimizer are determined by the surrounding model; obj only defines the mapping of input to output activations (the "forward pass"), possibly including techniques such as dropout or batch normalization.

The TensorFlow Hub team recommends you to implementing the Reusable Saved Model interface in all Saved Models that are meant to be reused in the above sense. Many utilities from the tensor flow_ hub library, notably hub. Keras _ layer, require Saved Models to implement it.

Relation to Model-Building Libraries:

A Reusable Saved Model uses only TensorFlow 2 primitives, independent of any particular model-building library like Keras or Sonnet. This facilitates reuse across model-building libraries, free from dependencies on the original model-building code.

Architecture:

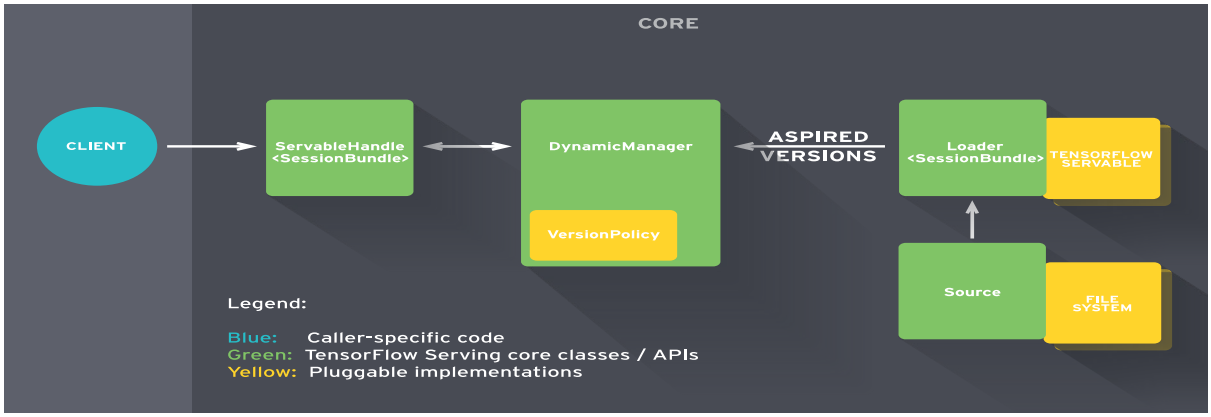
TensorFlow Serving is a flexible, high-performance serving system for machine learning models, designed for production environments. TensorFlow Serving makes it easy to deploy new algorithms and experiments, while keeping the same server architecture and APIs. TensorFlow Serving provides out of the box integration with TensorFlow models, but can be easily extended to serve other types of models.

Servables:

Servables are the central abstraction in TensorFlow Serving. Servables are the underlying objects that clients use to perform computation.

The size and granularity of a Servable is flexible. A single Servable might include anything from a single shard of a lookup table to a single model to a tuple of inference models. Servables can be of any type and interface, enabling flexibility and future improvements such as:

- streaming results
- experimental APIs
- asynchronous modes of operation



Models:

TensorFlow Serving represents a **model** as one or more servables. A machine-learned model may include one or more algorithms (including learned weights) and lookup or embedding tables.

You can represent a **composite model** as either of the following:

- multiple independent servables
- single composite servable

Extensibility:

TensorFlow Serving provides several extension points where you can add new functionality.

TensorFlow Serving includes two policies that accommodate most known use-cases.

1) Availability Preserving Policy

2) Resource Preserving Policy

Simple usage:

For simple usage of TensorFlow Serving where the serving availability of a model is important and the resource costs low, the Availability Preserving Policy will ensure that the new version is loaded and ready before unloading the old one.

Sophisticated usage:

For sophisticated usage of TensorFlow Serving, for example managing versions across multiple server instances, the Resource Preserving Policy requires the least resources (no extra buffer for loading new versions).

Source:

New Sources could support new filesystems, cloud offerings and algorithm backends. TensorFlow Serving provides some common building blocks to make it easy & fast to create new sources. For example, TensorFlow Serving includes a utility to wrap polling behavior around a simple source. Sources are closely related to Loaders for specific algorithms and data hosting servables.

Loader:

Loaders are the extension point for adding algorithm and data backends. TensorFlow is one such algorithm backend. For example, you would implement a new Loader in order to load, provide access to, and unload an instance of a new type of servable machine learning model.

Batcher:

Batching of multiple requests into a single request can significantly reduce the cost of performing inference, especially in the presence of hardware accelerators such as GPUs. TensorFlow Serving includes a request batching widget that lets clients easily batch their type-specific inferences across requests into batch requests that algorithm systems can more efficiently process.

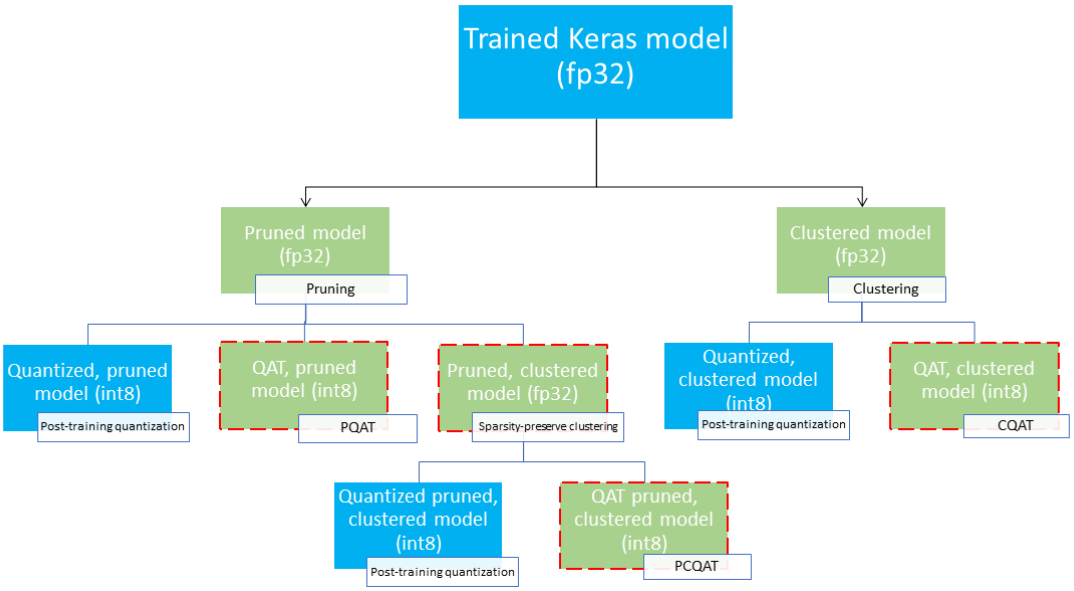
Optimization Requirements:

Optimization:

The TensorFlow Model Optimization Toolkit is a suite of tools for optimizing ML models for deployment and execution. Among many uses, the toolkit supports techniques used to:

1) Reduce latency and inference cost for cloud and edge devices (e.g. mobile, IoT).
2) Deploy models to edge devices with restrictions on processing, memory, power-consumption, network usage, and model storage space.
3) Enable execution on and optimize for existing hardware or new special purpose accelerators.

Grappler is the default graph optimization system in the TensorFlow runtime. Grappler applies optimizations in graph mode to improve the performance of your TensorFlow computations through graph simplifications and other high-level optimizations such as in lining function bodies to enable inter-procedural optimizations.



(Tensor Flow Optimization Toolkit)

Available graph optimizers:

Grappler performs graph optimizations through a top-level driver called the Meta Optimizer. The following graph optimizers are available with TensorFlow:

Constant folding optimizer: Statically infers the value of tensors when possible by folding constant nodes in the graph and materializes the result using constants.

Arithmetic optimizer: Simplifies arithmetic operations by eliminating common subexpressions and simplifying arithmetic statements.

Layout optimizer: Optimizes tensor layouts to execute data format dependent operations such as convolutions more efficiently.

Remapper optimizer: Remaps subgraphs onto more efficient implementations by replacing commonly occurring subgraphs with optimized fused monolithic kernels.

Memory optimizer: Analyzes the graph to inspect the peak memory usage for each operation and inserts CPU-GPU memory copy operations for swapping GPU memory to CPU to reduce the peak memory usage.

Dependency optimizer: Removes or rearranges control dependencies to shorten the critical path for a model step or enables other optimizations. Also removes nodes that are effectively no-ops such as Identity.

Pruning optimizer: Prunes nodes that have no effect on the output from the graph. It is usually run first to reduce the size of the graph and speed up processing in other Grappler passes.

Function optimizer: Optimizes the function library of a TensorFlow program and in lines function bodies to enable other inter-procedural optimizations.

Shape optimizer: Optimizes subgraphs that operate on shape and shape related information.

Autoparallel optimizer: Automatically parallelizes graphs by splitting along the batch dimension. This optimizer is turned OFF by default.

Loop optimizer: Optimizes the graph control flow by hoisting loop-invariant subgraphs out of loops and by removing redundant stack operations in loops. Also optimizes loops with statically known trip counts and removes statically known dead branches in conditionals.

Scoped allocator optimizer: Introduces scoped allocators to reduce data movement and to consolidate some operations.

Pin to host optimizer: Swaps small operations onto the CPU. This optimizer is turned OFF by default.

Auto mixed precision optimizer: Converts data types to float16 where applicable to improve performance. Currently applies only to GPUs.

Debug stripper: Strips nodes related to debugging operations from the graph. This optimizer is turned OFF by default.

The TensorFlow Model Optimization Toolkit minimizes the complexity of optimizing machine learning inference.

Inference efficiency is a critical concern when deploying machine learning models because of latency, memory utilization, and in many cases power consumption. Particularly on edge devices, such as mobile and Internet of Things (IoT), resources are further constrained, and model size and efficiency of computation become a major concern.

USE CASES:

Model optimization is useful, among other things, for:

1) Reducing latency and cost for inference for both cloud and edge devices (e.g. mobile, IoT).
2) Deploying models on edge devices with restrictions on processing, memory and/or power-consumption.
3) Reducing payload size for over-the-air model updates.
4) Enabling execution on hardware restricted-to or optimized-for fixed-point operations.
5) Optimizing models for special purpose hardware accelerators.

Optimization Technique:

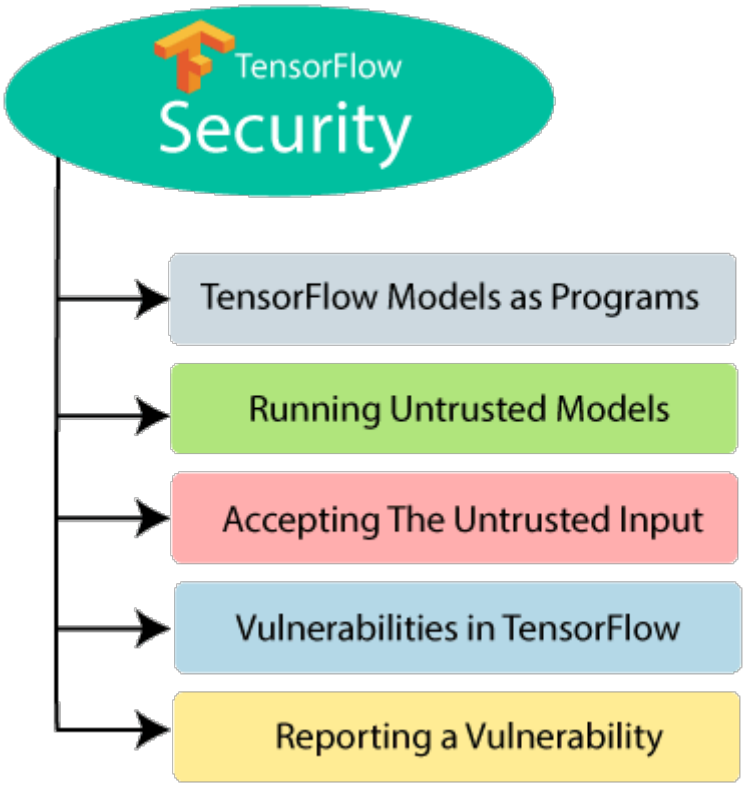
The area of model optimization can involve various techniques:

1) Reduce parameter count with pruning and structured pruning.
2) Reduce representational precision with quantization.
3) Update the original model topology to a more efficient one with reduced parameters or faster execution. For example, tensor decomposition methods and distillation

Security Requirements:

Security:

TensorFlow Security is **determined by the computational graph, whether the user-provided checkpoint is safe or not**. Generally, creating a computational graph with malicious inspections can trigger uncommon and insecure behavior.



TensorFlow Models As Programs:

TensorFlow programs are encoded as computation graphs. The model's parameters are often stored separately in checkpoints.

At runtime, TensorFlow executes the computation graph using the parameters provided. Note that the behavior of the computation graph may change depending on the parameters provided. **TensorFlow itself is not a sandbox**. When executing the computation graph, TensorFlow may read and write files, send and receive data over the network, and even spawn additional processes. All these tasks are performed with the permission of the TensorFlow process. Allowing for this flexibility makes for a powerful machine learning platform, but it has security implications.

Running Untrusted Models:

General Rule:

Always execute untrusted models inside a sandbox.

There are several ways in which a model could become untrusted. Obviously, if an untrusted party supplies TensorFlow kernels, arbitrary code may be executed. The same is true if the untrusted party provides Python code, such as the Python code that generates TensorFlow graphs.

Graphs can contain vulnerabilities of their own. To allow users to provide checkpoints to a model you run on their behalf. You must carefully audit your model, and they recommend you run the TensorFlow process in a sandbox.

Accepting Untrusted Inputs:

It is possible to write models that are secure in the sense that they can safely process untrusted inputs assuming there are no bugs. There are, however, two main reasons to not rely on this.

1) It is easy to write models which must not be exposed to untrusted inputs.
2) There are bugs in any software system of sufficient complexity.

Letting users control inputs could allow them to trigger bugs either in TensorFlow or in dependencies.

In general, it is good practice to isolate parts of any system which is exposed to untrusted (e.g., user-provided) inputs in a sandbox.

Vulnerabilities in TensorFlow:

TensorFlow is a large and complex system. It also depends on a large set of third party libraries. It is possible that TensorFlow or its dependencies may contain vulnerabilities that would allow triggering unexpected or dangerous behavior with specially crafted inputs.

Given TensorFlow's flexibility, it is possible to specify computation graphs which exhibit unexpected or unwanted behavior. The fact that TensorFlow models can perform arbitrary computations means that they may read and write files, communicate via the network, produce deadlocks and infinite loops, or run out of memory. It is only when these behaviors are outside the specifications of the operations involved that such behavior is a vulnerability.

Reporting Vulnerabilities:

Please fill out report about any security related issues you find.

Please use a descriptive title for your report.

In addition, please include the following information along with your report:

1) Your name and affiliation (if any).
2) A description of the technical details of the vulnerabilities. It is very important to let us know how we can reproduce your findings.
3) A minimal example of the vulnerabity.
4) An explanation of who can exploit this vulnerability, and what they gain when doing so -- write an attack scenario. This will help us evaluate your report quickly, especially if the issue is complex.
5) Whether this vulnerability is public or known to third parties. If it is, please provide details.

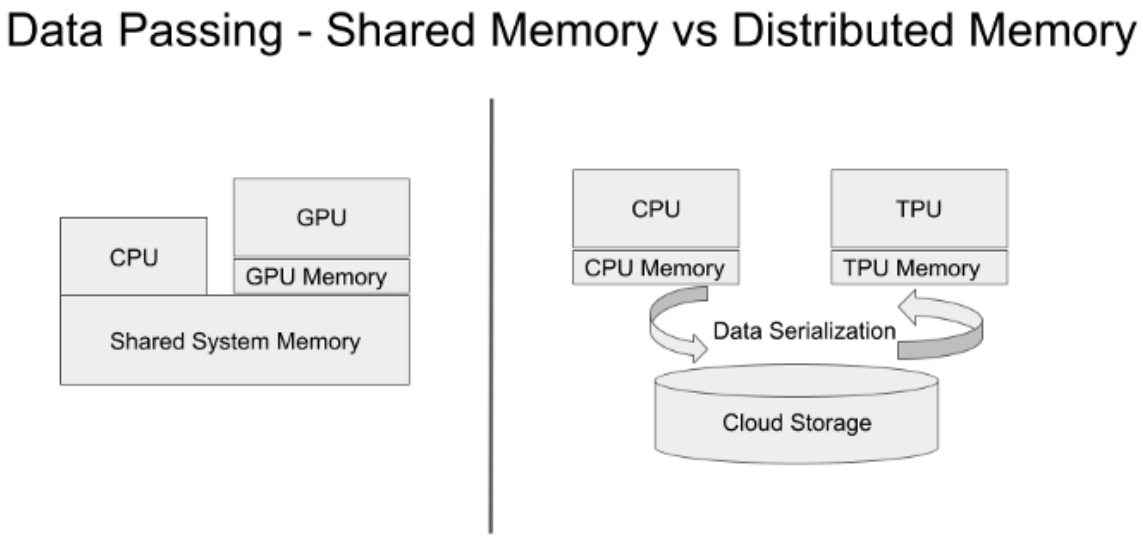
For each vulnerability, we try to ingress it as soon as possible, given the size of the team and the number of reports. Vulnerabilities will, in general, be batched to be fixed at the same time as a quarterly release. An exception to this rule is for high impact vulnerabilities where exploitation of models used for inference in products (i.e., not models created just to showcase a vulnerability) is possible. In these cases, we will attempt to do patch releases within an accelerated timeline, not waiting for the next quarterly release.

Reliability Requirements:

Scalability:

In scalable machine learning, the components of the system have their own work or task which helps the whole system to lead towards the solution of the problem rapidly.

When the size of data becomes very large, the performance of machine learning models becomes a concern. In such situations, the machine learning models need to be scaled which not only helps in saving time and memory but also helps in improving the performance of the model. TensorFlow provides features to scale machine learning models easily and effectively.



Fault Tolerance:

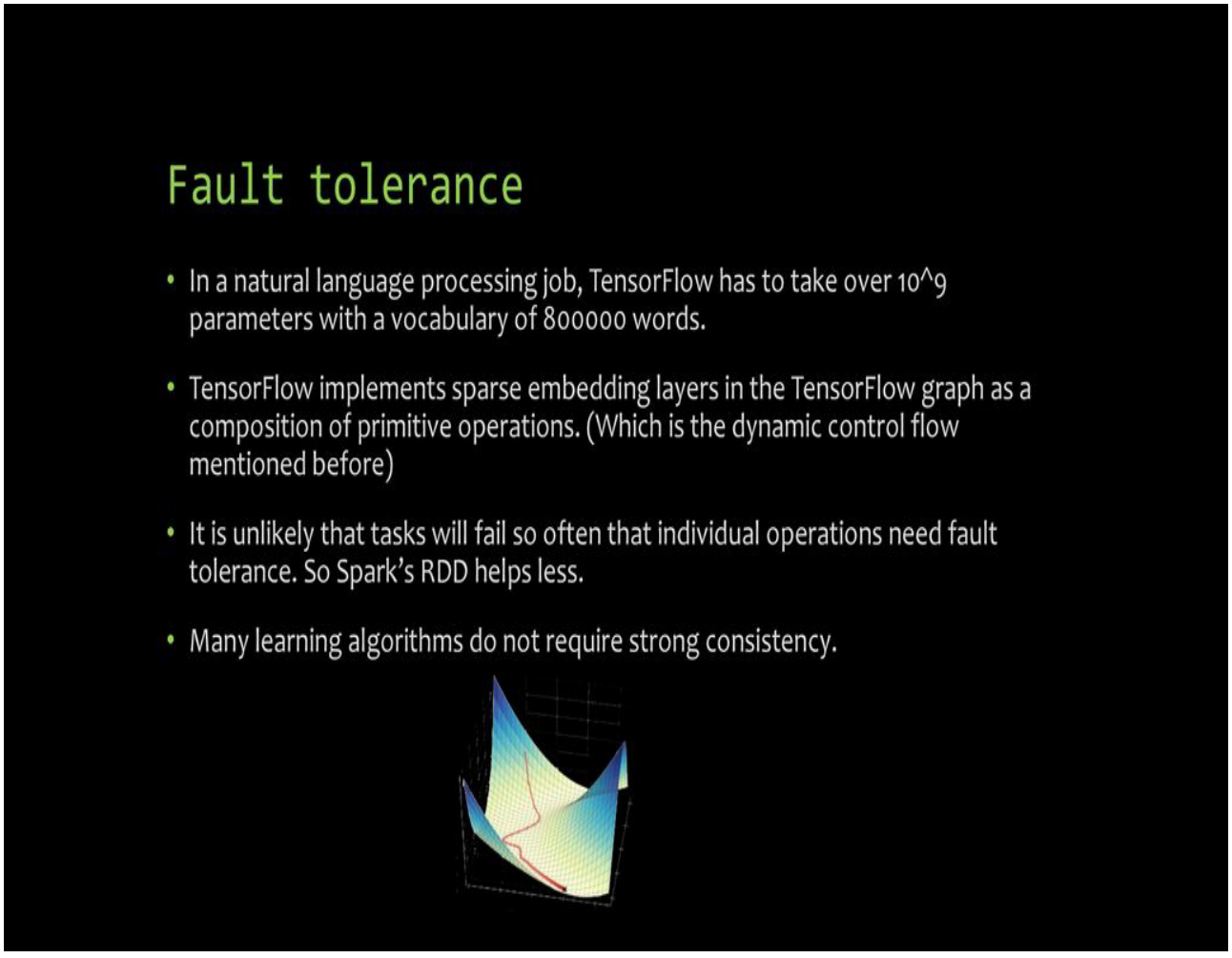
Fault tolerance refers to **a mechanism of periodically saving the states of trackable objects, such as parameters and models**. This enables you to recover them in the event of a program/machine failure during training.

You will learn how to implement fault tolerance for training in Tensorflow 2 in two ways:

1. If you use the Keras Model.fit API, you can pass the tf.keras.callbacks.BackupAndRestore callback to it.

2. If you use a custom training loop (with tf.GradientTape), you can arbitrarily save checkpoints using the tf.train.Checkpoint and tf.train.CheckpointManager APIs.

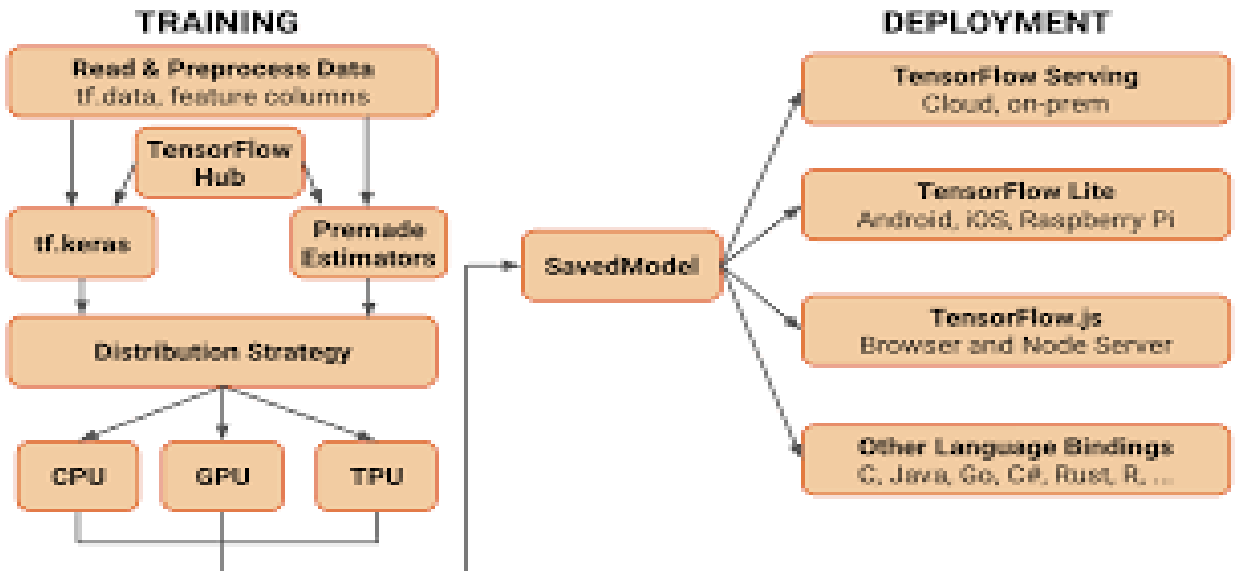
In TensorFlow, if you use the Keras Model.fit API for training, you can provide the tf.keras.callbacks.BackupAndRestore callback to add the fault tolerance functionality.



Failures in a distributed execution can be detected in a variety of places. The main ones we rely on are (a) an error in a communication between a Send and Receive node pair, and (b) periodic health-checks from the master process to every worker process.

When a failure is detected, the entire graph execution is aborted and restarted from scratch. Recall however that Variable nodes refer to tensors that persist across executions of the graph. We support consistent checkpointing and recovery of this state on a restart.

Process Requirements:



Open-source contribution requirements for developers:

The TensorFlow ecosystem can only grow through the contributions of developers.

You can contribute code, make improvements to the TensorFlow API documentation, or add your Jupyter notebooks to the tensorflow/examples repo. This guide provides everything you need to get started. Our most common contributions include *code*, *documentation*, and *community support*.

1. Write code.
2. Improve tests.
3. Improve Documentation.
4. Answer questions on Stack Overflow.
5. Participate in the discussion on our mailing lists.
6. Contribute example notebooks.
7. Investigate bugs and issues on GitHub.
8. Review and comment on pull requests from other developers.
9. Report an issue.
10. Give a “thumbs up” 👍 on issues that are relevant to you.

11. Reference TensorFlow in your blogs, papers, and articles.
12. Talk about TensorFlow on social media.

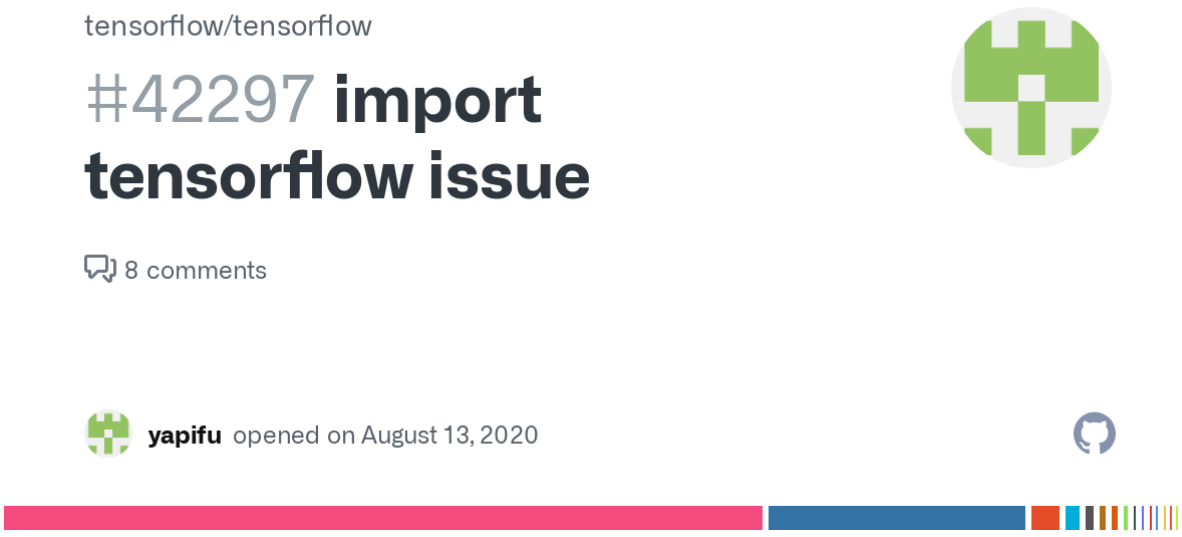
TensorFlow was originally developed by researchers and engineers from the Google Brain team within Google's AI organization. Google open sourced TensorFlow in the hope of sharing technology with the external community and encouraging collaboration between researchers and industry. Since then, TensorFlow has grown into a thriving ecosystem of products, on a wide range of platforms. But there goal is still to make machine learning accessible to anyone, anywhere.

TensorFlow Versions:

Version	Python version	Compiler	Build tools	cuDNN	CUDA
tensorflow-2.3.0	3.5-3.8	GCC 7.3.1	Bazel 3.1.0	7.6	10.1
tensorflow-2.2.0	3.5-3.8	GCC 7.3.1	Bazel 2.0.0	7.6	10.1
tensorflow-2.1.0	2.7, 3.5-3.7	GCC 7.3.1	Bazel 0.27.1	7.6	10.1
tensorflow-2.0.0	2.7, 3.3-3.7	GCC 7.3.1	Bazel 0.26.1	7.4	10.0
tensorflow_gpu-1.15.0	2.7, 3.3-3.7	GCC 7.3.1	Bazel 0.26.1	7.4	10.0
tensorflow_gpu-1.14.0	2.7, 3.3-3.7	GCC 4.8	Bazel 0.24.1	7.4	10.0
tensorflow_gpu-1.13.1	2.7, 3.3-3.7	GCC 4.8	Bazel 0.19.2	7.4	10.0
tensorflow_gpu-1.12.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.11.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.10.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.9.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.11.0	7	9
tensorflow_gpu-1.8.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.10.0	7	9
tensorflow_gpu-1.7.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.9.0	7	9
tensorflow_gpu-1.6.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.9.0	7	9
tensorflow_gpu-1.5.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.8.0	7	9
tensorflow_gpu-1.4.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.5.4	6	8
tensorflow_gpu-1.3.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.5	6	8
tensorflow_gpu-1.2.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.5	5.1	8
tensorflow_gpu-1.1.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.2	5.1	8
tensorflow_gpu-1.0.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.2	5.1	8

TensorFlow gave different version releases based on different problems and needs. They introduce versions that are compatible to each system to fulfill end users requirements.

Issue Life Cycle:



If you open a GitHub Issue, here is our policy:

1. It must be a bug/performance issue or a feature request or a build issue or a documentation issue (for small doc fixes please send a PR instead).
2. Make sure the Issue Template is filled out.
3. The issue should be related to the repo it is created in.

Here's why we have this policy:

We want to focus on the work that benefits the whole community, e.g., fixing bugs and adding features. Individual support should be sought on Stack Overflow or other non-GitHub channels. It helps us to address bugs and feature requests in a timely manner.

Describe the Problem:

Describe the problem clearly here. Be sure to convey here why it's a bug in TensorFlow or a feature request.

Source Code/logs:

Include any logs or source code that would be helpful to diagnose the problem. If including tracebacks, please include the full traceback. Large logs and files should be attached. Try to provide a reproducible test case that is the bare minimum necessary to generate the problem.

Documentation Requirements:

INSTALLATION:

Here is the installation link of document of tensor flow:

https://www.tensorflow.org/api_docs

Tutorials:

Here is the link to official You-tube channel of tensor flow.

https://www.youtube.com/tensorflow

User Manual:

Here is the link of user manual of tensor flow.

https://www.tensorflow.org/guide