**Problem**:

Making and training the machine learning model is not difficult nowadays. However, due to the increased applications of AI in daily life. The focus is shifted towards the deployment of machine learning models to production.

**Model Serving Options**

The easiest and common way to deploy a trained ML model is by saving it into the *binary format* of the appropriate tool then wrap it in a service such as the Flask, and then use it for the frontend. This approach has several cons for instance: as the number of models grows, the number of services grows and so the number of failure points, latency and the list go on which is difficult to manage. Another more recent approach is to standardize the model format so it can be used programmatically using any programming language, so you don’t have to wrap it in a microservice. This is especially useful for data processing streams where latency and error management are an issue. This approach is known as *Model as Data*. Nowadays Model as Data is the most common way to deploy an ML model.

*For this Task, we will use Model as Data Serving Option with Flask*

Some of the tools focused on *Model as Data* are Seldon, Clipper, and TensorFlow Serving.

The pros of the ‘Model as Data’ approach are:

* Easier to develop.
* Data Engineers do not need to take care of maintenance production and monitoring. SREs can manage these services.
* Helps in automation. The tools such as AWS SageMaker take care of deploying the services. In other words, tools like Kubeflow or SageMaker take care of all the aspects from training to scoring.
* One more advantage is that one can keep the model state in the application with metrics and other metadata.

**Model as Data**

A more recent approach is to standardize the models as data so they can be read in any programming language. Currently, TensorFlow (TF) has emerged as the de-facto standard, the new SavedModel format contains a complete TensorFlow program, including weights and computation. It does not require the original model building code to run, which makes it useful for sharing.

**Working of Model Serving**

**Model Serving (Model as Data) in TensorFlow**

The serving lifecycle in TensorFlow starts when TF Serving successfully identifies a model on the disk. It oversees the file system to intercept the arrival of the new version of the model. The moment a new model is identified, it proceeds by creating a Loader for that version of the model. The Loader has the metadata in it. This metadata contains essential information about the ML model such as the procedures to calculate GPU memory, RAM, and other resources.

**Export Model for Serving**

TensorFlow offers SavedModel class to export the model incorrect format. (SavedModel works for all types of TensorFlow models to save them). The SavedModel has the feature to save more than one meta-graph to a single Saved Model. It offers to have different graphs for the different tasks.

For example, if the model has been trained. To perform inference, the graph does not need some training-specific operations. These operations may include the optimizer’s variables, learning rate(alpha) tensors, or other preprocessing operations.

Graph

Graph

Meta Graph

Meta Graph

Saved Model

In other words, the SavedModels allows the ML engineers to save the graphs with different configurations. Here is a twist, if we save three graphs, all three graphs will use the same set of variables. Which results in *memory efficiency.*

**Easy Deployment:**

A few years back, during the deployment of the TF model on the mobile device. The programmer must know the names of the input and output tensors for feeding and fetching the data to/from the model. This was a very difficult task for most of the programmers.

To assist the programmers, TF offers SignatureDefs. In simple words, SignatureDefs determine the proper input and output tensors for the computational graph.

**Serving APIs in TF**

Now there are the following three APIs for Serving.

1. Classification
2. Predict
3. Regression

Each SignatureDefs matches a specific RPC API. The Regression SignatureDef is used for Regression RPC API. The Classification SignatureDef is used to classify RPC API and the same method is repeated for others.