1. Why would you want to use the Data API?

Ans. An API (Application Programming Interface) is a set of functions that allows applications to access data and interact with external software components, operating systems, or microservices. To simplify, an API delivers a user response to a system and sends the system's response back to a user.

1. What are the benefits of splitting a large dataset into multiple files?

Ans. -Splitting a large dataset into multiple files makes it possible to shuffle it at a coarse level before shuffling it at a finer level using a shuffling buffer.

-It also makes it possible to handle huge datasets that do not fit on a single machine.

-It’s also simpler to manipulate thousands of small files rather than one huge file; for example, it’s easier to split the data into multiple subsets.

-Lastly, if the data is split across multiple files spread across multiple servers, it is possible to download several files from different servers simultaneously, which improves the bandwidth usage.

1. During training, how can you tell that your input pipeline is the bottleneck? What can you do to fix it?

Ans. -You can use TensorBoard to visualize profiling data: if the GPU is not fully utilized then your input pipeline is likely to be the bottleneck.

-You can fix it by making sure it reads and preprocesses the data in multiple threads in parallel, and ensuring it prefetches a few batches.

-If this is insufficient to get your GPU to 100% usage during training, make sure your preprocessing code is optimized. -----You can also try saving the dataset into multiple TFRecord files, and if necessary perform some of the preprocessing ahead of time so that it does not need to be done on the fly during training (TF Transform can help with this).

-If necessary, use a machine with more CPU and RAM, and ensure that the GPU bandwidth is large enough.

Can you save any binary data to a TFRecord file, or only serialized protocol buffers?

Ans. The TFRecord format is a simple format for storing a sequence of binary records.

Protocol buffers are a cross-platform, cross-language library for efficient serialization of structured data.

Protocol messages are defined by .proto files, these are often the easiest way to understand a message type.

The tf.train.Example message (or protobuf) is a flexible message type that represents a {"string": value} mapping. It is designed for use with TensorFlow and is used throughout the higher-level APIs such as TFX.

1. Why would you go through the hassle of converting all your data to the Example protobuf format? Why not use your own protobuf definition?

Ans. -The Example protobuf format has the advantage that TensorFlow provides some operations to parse it (the tf.io.parse\*example() functions) without you having to define your own format. It is sufficiently flexible to represent instances in most datasets.

-However, if it does not cover your use case, you can define your own protocol buffer, compile it using protoc (setting the --descriptor\_set\_out and --include\_imports arguments to export the protobuf descriptor), and use the tf.io.decode\_proto() function to parse the serialized protobufs (see the “Custom protobuf” section of the notebook for an example). It’s more complicated, and it requires deploying the descriptor along with the model, but it can be done.

1. When using TFRecords, when would you want to activate compression? Why not do it systematically?

Ans. -When using TFRecords, you will generally want to activate compression if the TFRecord files will need to be downloaded by the training script, as compression will make files smaller and thus reduce download time.

-But if the files are located on the same machine as the training script, it’s usually preferable to leave compression off, to avoid wasting CPU for decompression.

1. Data can be preprocessed directly when writing the data files, or within the tf.data pipeline, or in preprocessing layers within your model, or using TF Transform. Can you list a few pros and cons of each option?

Ans. Pros:

-If you preprocess the data when creating the data files, the training script will run faster, since it will not have to perform preprocessing on the fly.

-In some cases, the preprocessed data will also be much smaller than the original data, so you can save some space and speed up downloads.

-It may also be helpful to materialize the preprocessed data, for example to inspect it or archive it.

Cons:

-it’s not easy to experiment with various preprocessing logics if you need to generate a preprocessed dataset for each variant.

-if you want to perform data augmentation, you have to materialize many variants of your dataset, which will use a large amount of disk space and take a lot of time to generate. -Lastly, the trained model will expect preprocessed data, so you will have to add preprocessing code in your application before it calls the model.