Q1: What does a SavedModel contain? How do you inspect its content?

Ans: A SavedModel contains a trained TensorFlow model's variables and graph definition. You can inspect its content using the saved\_model\_cli command-line interface or by loading it into a Python script and exploring its contents.

Q2: When should you use TF Serving? What are its main features? What are some tools you can use to deploy it?

Ans: You should use TF Serving when you need to deploy a TensorFlow model in a production environment. Its main features include support for multiple models and versions, scalable and flexible deployment options, and monitoring and metrics. Some tools you can use to deploy it include Docker, Kubernetes, and Apache Beam.

Q3: How do you deploy a model across multiple TF Serving instances?

Ans: To deploy a model across multiple TF Serving instances, you can use a load balancer or a cluster management system like Kubernetes. You can also configure TF Serving to automatically load balance requests across replicas.

Q4.: When should you use the gRPC API rather than the REST API to query a model served by TF Serving?

Ans: You should use the gRPC API when you need low-latency, high-throughput communication with a model served by TF Serving. The REST API may be more appropriate for simpler use cases or when interoperability with other systems is a priority.

Q5.: What are the different ways TFLite reduces a model’s size to make it run on a mobile or embedded device?

Ans: TFLite reduces a model's size by quantizing its weights and activations, pruning its parameters, and using post-training quantization techniques like dynamic range quantization and full integer quantization.

Q6: What is quantization-aware training, and why would you need it?

Ans: Quantization-aware training is a technique for training models that are optimized for deployment on hardware with low precision arithmetic, like mobile and embedded devices. It involves simulating the effects of quantization during training by using low-precision arithmetic and adding noise to the weights.

Q7.: What are model parallelism and data parallelism? Why is the latter generally recommended?

Ans: Model parallelism and data parallelism are two techniques for training models across multiple devices or servers. Model parallelism involves splitting the model across devices, while data parallelism involves splitting the data across devices. Data parallelism is generally recommended because it is simpler to implement and more efficient for most models.

Q7.: When training a model across multiple servers, what distribution strategies can you use? How do you choose which one to use?

Ans: When training a model across multiple servers, you can use various distribution strategies like MirroredStrategy, ParameterServerStrategy, and CentralStorageStrategy. The choice of strategy depends on factors like the size and complexity of the model, the number of servers available, and the communication and synchronization overhead.