**1. What does a SavedModel contain? How do you inspect its content?**

A **SavedModel** in TensorFlow is a format that contains:

📌 **Model Architecture** → Graph of operations (for TensorFlow execution).  
📌 **Trained Weights** → Serialized tensors.  
📌 **Signatures** → Defines input/output tensor shapes.  
📌 **Assets** → Extra files (like vocabulary for NLP models).  
📌 **Metadata** → Training info, optimizers, etc.

**How to Inspect a SavedModel?**

✅ **Check Model Structure:**

saved\_model\_cli show --dir saved\_model\_directory --all

✅ **Inspect with Python:**

import tensorflow as tf

model = tf.saved\_model.load("saved\_model\_directory")

print(list(model.signatures.keys())) # Check available functions

**2. When to Use TF Serving? Features & Deployment Tools**

🛠 **Use TensorFlow Serving when:**  
✅ You need to **deploy ML models in production**.  
✅ You require **high performance & scalability**.  
✅ You need **real-time inference with minimal latency**.

📌 **Main Features:**

* Supports **model versioning**.
* Handles **batching & multi-threading** automatically.
* Provides **REST & gRPC APIs** for model queries.

📌 **Deployment Tools:**  
✅ **Docker** → tensorflow/serving container.  
✅ **Kubernetes** → For large-scale deployment.  
✅ **TensorFlow Model Server** → Native binary for running TF models.

**3. Deploying a Model Across Multiple TF Serving Instances**

🚀 **Steps:**  
1️⃣ **Start multiple instances** of TF Serving on different servers.  
2️⃣ Use a **load balancer** (e.g., Kubernetes, Nginx) to distribute requests.  
3️⃣ If using **Kubernetes**, deploy using a tf-serving.yaml file.

Example **Kubernetes Deployment:**

apiVersion: apps/v1

kind: Deployment

metadata:

name: tf-serving

spec:

replicas: 3 # Multiple instances

template:

spec:

containers:

- name: tensorflow-serving

image: tensorflow/serving

**4. When to Use gRPC API Instead of REST API?**

✅ **Use gRPC when:**

* You need **faster, low-latency communication** (binary protocol).
* You want **streaming support** (better for real-time apps).
* You are working with **mobile/embedded devices** with limited resources.

✅ **Use REST when:**

* You need **human-readable** JSON responses.
* Your system already uses **RESTful APIs**.
* You are making **simple, occasional requests**.

💡 **gRPC is better for high-performance, low-latency ML inference.**

**5. How TFLite Reduces Model Size for Mobile/Embedded Devices**

📌 **Techniques Used:**  
✅ **Quantization** → Convert float32 weights to int8 to reduce size.  
✅ **Weight Pruning** → Remove redundant weights.  
✅ **Weight Clustering** → Group similar weights to reduce precision loss.  
✅ **Operator Fusion** → Combine multiple ops into one to speed up inference.

💡 These techniques help deploy models on **low-power devices** like Raspberry Pi, Edge AI chips, etc.

**6. What is Quantization-Aware Training? Why Use It?**

📌 **Quantization-Aware Training (QAT):**

* **During training**, the model simulates the effect of lower-precision calculations.
* Helps models **learn to be robust to quantization errors**.

✅ **Why Use QAT?**

* Standard **post-training quantization** can **degrade model accuracy**.
* QAT ensures the model retains accuracy **even after conversion to int8**.
* Essential for **edge devices (IoT, mobile, self-driving cars, etc.)**.

**7. Model Parallelism vs. Data Parallelism? Why Prefer Data Parallelism?**

📌 **Model Parallelism:**

* **Split model across multiple devices** (e.g., ResNet on 2 GPUs).
* Used when a model **is too large for a single GPU**.
* Hard to implement (needs careful layer placement).

📌 **Data Parallelism:**

* **Same model on multiple GPUs**, each processing different batches.
* Faster training, **easier to scale**.
* Recommended for **most deep learning tasks**.

💡 **Data parallelism is preferred** unless the model is **too large to fit into one GPU**.

**8. Training a Model Across Multiple Servers**

🚀 **Distributed Training Strategies:**  
✅ **Mirrored Strategy** → For GPUs in one machine.  
✅ **MultiWorkerMirroredStrategy** → For GPUs across multiple machines.  
✅ **TPU Strategy** → If using TPUs for training.

📌 **Example: Multi-GPU Training with TF Strategy**

import tensorflow as tf

strategy = tf.distribute.MultiWorkerMirroredStrategy()

with strategy.scope():

model = create\_model()

model.compile(optimizer="adam", loss="categorical\_crossentropy")

💡 This enables **scalable training across multiple nodes**.