PyTorch - Practice Sheet

10 conceptual Q&A AI/ML

Q1. What is PyTorch and what AI/ML use case is it designed for?

A: PyTorch is a framework or service targeting a specific AI/ML domain training neural networks, building LLM pipelines, deploying models, or orchestrating agents. Understanding its scope helps you know when to use it vs alternatives.

Q2. What are the core abstractions in PyTorch?

A: PyTorch organizes work around abstractions like models, tensors, pipelines, agents, or embeddings. Learning these core types is the first step to using the framework effectively.

Q3. How does PyTorch handle training or inference?

A: For training, PyTorch defines a model architecture, a loss function, and an optimizer. For inference, a trained model processes input data and returns predictions. Batch processing and hardware acceleration are key performance levers.

Q4. How does PyTorch manage data preprocessing?

A: PyTorch provides data loaders, tokenizers, or pipeline components that transform raw data into the tensor or embedding format the model expects. Proper preprocessing is critical for model correctness and training speed.

Q5. How does PyTorch support GPU/TPU acceleration?

A: PyTorch detects available accelerators and moves computation to them. Specify the device explicitly (cuda, mps, tpu) to avoid accidentally running expensive operations on CPU. Mixed-precision (fp16/bf16) further reduces memory and increases throughput.

Q6. How do you evaluate a model's performance in PyTorch?

A: Evaluation uses a held-out validation set and metrics (accuracy, F1, BLEU, perplexity) appropriate to the task. Monitor both training and validation metrics to detect overfitting. Use a test set only at the very end to report final results.

Q7. How do you deploy a model trained with PyTorch?

A: Export the model to a portable format (ONNX, TorchScript, SavedModel). Serve it via a model server (TorchServe, TF Serving, Triton) or embed it in an API endpoint. Monitor inference latency and drift in production.

Q8. How does PyTorch handle distributed or multi-GPU training?

A: PyTorch provides data-parallel or model-parallel strategies to split work across GPUs. Data parallelism replicates the model on each GPU and averages gradients. Model parallelism splits layers across devices for models too large for one GPU.

Q9. How do you track experiments and versions in PyTorch?

A: Use an experiment tracker (MLflow, Weights & Biases, Comet) alongside PyTorch to log hyperparameters, metrics, and artifacts. Track the code version and data version for full reproducibility.

Q10. What are common pitfalls when using PyTorch in production?

A: Common issues include training-serving skew (different preprocessing), missing input validation, no latency SLA enforcement, models not versioned, and no process to retrain on drifted data distributions.