Theoretical Understanding

Q1: Explain the primary differences between TensorFlow and PyTorch. When would you choose one over the other?

Answer:
The primary difference is in how computation graphs are handled: TensorFlow (especially v1) uses static graphs, while PyTorch uses dynamic (eager) execution, making it more intuitive and Pythonic.

- Choose PyTorch for research, experimentation, or education, due to its flexibility and ease of debugging.
- Choose TensorFlow for production deployment, especially on mobile or in distributed systems, due to tools like TensorFlow Serving and TensorFlow Lite.

Q2: Describe two use cases for Jupyter Notebooks in AI development.

Answer:

1. Prototyping and Experimentation: Jupyter allows for quick testing and visualization of models, data preprocessing steps, and evaluation metrics interactively.
2. Data Exploration and Visualization: It's ideal for analyzing datasets with graphs, tables, and plots using libraries like `matplotlib`, `seaborn`, and `pandas`.

Q3: How does spaCy enhance NLP tasks compared to basic Python string operations?

Answer:
spaCy provides linguistically informed processing like tokenization, POS tagging, named entity recognition, and dependency parsing, which go far beyond basic string methods (e.g., `.split()` or `.find()`). This allows for accurate and contextual understanding of text, essential for building robust NLP applications.

Comparing scikit-learn, tensorflow and pytorch

1. Architecture

Feature	TensorFlow	PyTorch	Scikit-learn
Core Design	Static computation graph (eager by default in TF 2.x)	Dynamic computation graph (define-by-run)	No computation graph; traditional ML pipeline
Language Support	Python, C++, JavaScript (via TensorFlow.js)	Primarily Python (C++ backend)	Pure Python
Deployment	TensorFlow Serving, TFLite, TensorFlow.js	TorchScript, ONNX	Not focused on deployment
Hardware Support	CPU, GPU, TPU	CPU, GPU	CPU (limited GPU support via joblib/sklearnex)

---

2. Syntax & Programming Style

Feature	TensorFlow	PyTorch	Scikit-learn
Ease of Use	Moderate to steep learning curve	Pythonic, intuitive	Very beginner-friendly
Model Definition	Using `tf.keras.Model`, `Sequential`, functional API	Use `torch.nn.Module` and forward method	Simple API (`fit`, `predict`, `score`)
Debugging	Harder in TF 1.x, better in TF 2.x (eager mode)	Easier due to dynamic graph	Straightforward
Training Loop	Can use `model.fit()` or custom loops	Typically custom loops	Handled internally

---

3. Use Cases

Use Case	TensorFlow	PyTorch	Scikit-learn
Deep Learning (CNNs, RNNs)	✅ Industry standard	✅ Preferred in research	❌ Not suitable
Production/Deployment	✅ Optimized for production	⚠️ Possible but less mature	❌ Not designed for production
Research & Experimentation	⚠️ Good with TF 2.x	✅ Excellent	⚠️ Limited to classical ML
Classical Machine Learning	⚠️ Some support via `tf.estimator`	⚠️ Basic support	✅ Full support (SVMs, Trees, Clustering, etc.)
Mobile/Embedded AI	✅ via TFLite	⚠️ Possible with optimization	❌

---

4. When to Use Which

Scenario	Use This Tool	Why
You want to build a **deep learning model** for production use	**TensorFlow**	Strong deployment tools (TFLite, TensorFlow Serving)
You're doing **research or prototyping** with deep learning	**PyTorch**	Flexible, intuitive, dynamic graph model
You're building **classical ML models** (e.g., SVMs, Random Forests, KNN)	**Scikit-learn**	Simple, mature, and optimized for classical ML workflows
You want to **deploy AI to mobile devices**	**TensorFlow (TFLite)**	Excellent support for on-device inference
You're teaching or learning ML basics	**Scikit-learn**	Clean, minimal code with well-documented algorithms
You need **explainable ML** with small datasets	**Scikit-learn**	Easier to explain and visualize decision boundaries
You want to convert models to **ONNX** for interoperability	**PyTorch or TensorFlow**	Both support ONNX export with some effort

---

Summary Table

Feature	TensorFlow	PyTorch	Scikit-learn
Best For	Production DL	Research DL	Classical ML
Learning Curve	Moderate/Steep	Moderate	Easy
Model Types	Deep Learning	Deep Learning	Traditional ML
Production Support	Excellent	Good (improving)	Minimal
GPU Support	Yes	Yes	Limited

Results/Outputs

Ethical Considerations

Potential Biases

In MNIST:

- **Class Imbalance**: Some digits may be underrepresented, leading to lower accuracy for those classes.
- **Overfitting on writing style**: The model might generalize poorly to handwriting styles not present in the training data.

In Amazon Reviews:

- **Sentiment bias**: The model may learn bias from skewed review distributions (e.g., more positive reviews).
- **Demographic bias**: Reviews might reflect cultural, gender, or product-related biases that are not explicit in the data.