Part 1: Theoretical Understanding

1. Short Answer Questions

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

TensorFlow and PyTorch are leading deep learning frameworks, differing in their computational approaches and use cases:

- ***Computational Graph:*** TensorFlow uses a static computational graph (define-and-run), where the graph is built first and then executed, offering optimization for production environments. PyTorch employs a dynamic computational graph (define-by-run), allowing real-time modifications, which is intuitive for experimentation.
- ***Ease of Use:*** PyTorch has a more Python-like interface, making it easier for rapid prototyping and debugging. TensorFlow's API is more structured but has a steeper learning curve, though TensorFlow 2.x with Keras simplifies this.
- ***Scalability:*** TensorFlow excels in large-scale deployment, supporting distributed training and mobile/edge devices via TensorFlow Lite. PyTorch is catching up with tools like TorchServe but is less mature in production settings.
- ***Community and Ecosystem:*** TensorFlow has broader industry adoption and tools like TensorFlow Serving, while PyTorch dominates in academia due to its flexibility.

When to Choose:

★ ***TensorFlow:*** Choose for production-grade applications, such as deploying models in mobile apps or large-scale systems requiring distributed training (e.g. recommendation systems in e-commerce).

★ ***PyTorch:*** Opt for research, prototyping, or projects requiring dynamic model changes (e.g. experimental NLP models).

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

Jupyter Notebooks are widely used in AI development for their interactivity and versatility. Two key use cases include:

❖ ***Model Prototyping and Experimentation:*** Jupyter Notebooks allow data scientists to write, test, and modify code incrementally, making them ideal for building and iterating on machine learning models. For example, a data scientist can load a dataset, preprocess it, train a model (e.g. a neural network in PyTorch), and visualize results in a single notebook, facilitating quick experimentation with hyperparameters.

❖ ***Data Visualization and Analysis:*** Notebooks support inline visualizations using libraries like Matplotlib or Seaborn, enabling real-time exploration of datasets. For instance, in a churn prediction project, a data scientist can plot feature distributions or model performance metrics (e.g. ROC curves) to gain insights and share findings with stakeholders.

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

spaCy is a powerful NLP library that significantly enhances natural language processing tasks compared to basic Python string operations:

- ***Pre-trained Models:*** spaCy provides pre-trained models for tasks like named entity recognition (NER), part-of-speech tagging, and dependency parsing, which are far more accurate and efficient than manual string manipulation (e.g. regex-based tokenization). For example, spaCy's NER can identify entities like "Apple" as an organization, while string operations require complex, error-prone rules.
- ***Efficiency and Scalability:*** spaCy is optimized for performance, processing large text corpora quickly using Cython-based implementations, whereas string operations like `*split()*` or `*find()*` are slow and limited for large datasets.
- ***Linguistic Features:*** spaCy offers advanced features like word embeddings and sentence boundary detection, enabling sophisticated tasks like sentiment analysis or text classification, which are impractical with basic string operations.
- ***Ease of Use:*** spaCy's pipeline simplifies complex NLP workflows into a few lines of code (e.g. `*nlp(text).ents*` for entity extraction), compared to the tedious, error-prone coding required for string-based solutions.

2. Comparative Analysis

Task 1: Compare Scikit-learn and TensorFlow in terms of:
- Target applications (e.g. classical ML vs. deep learning).
- Ease of use for beginners.
- Community support.

Criteria	Scikit-learn	TensorFlow
Target Applications	Best suited for classical machine learning tasks, such as linear regression, decision trees, SVMs, and clustering. Ideal for structured data tasks like customer segmentation or fraud detection.	Designed for deep learning tasks, such as building neural networks (e.g. CNNs, RNNs) for image classification, NLP, or time-series forecasting. Also supports classical ML via Keras but excels in large-scale neural network applications.

Ease of Use for Beginners	Highly beginner-friendly due to its simple, consistent API and extensive documentation. For example, training a model requires minimal code (e.g. `model.fit(X, y)`). Minimal setup is needed, as it runs on CPU.	Steeper learning curve, especially for beginners, due to its complexity and need for understanding concepts like computational graphs. TensorFlow 2.x with Keras simplifies usage, but GPU setup and debugging remain challenging.
Community Support	Strong community with extensive tutorials, Stack Overflow support, and integration with other Python libraries (e.g., Pandas). Less frequent updates but stable for classical ML tasks.	Robust community with widespread industry adoption, active GitHub contributions, and tools like TensorFlow Hub and TensorFlow Extended (TFX). Frequent updates and support for cutting-edge AI research.

Summary:

Scikit-learn is ideal for quick prototyping of classical ML models on structured data, offering simplicity and accessibility for beginners.
TensorFlow is better for complex deep learning projects and production deployment, with a broader ecosystem but requiring more expertise.