Theoretical Understanding

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

The primary differences between TensorFlow and PyTorch are in their computation graph approach and typical use cases.

TensorFlow traditionally uses static computation graphs, which are defined before running the model and are optimized for large-scale production and deployment.

PyTorch uses dynamic computation graphs, allowing models to be defined and modified on-the-fly, making it more intuitive and flexible for research and rapid prototyping.

Example:

- **TensorFlow:** Preferred for deploying models at scale in production (e.g., TensorFlow Serving, TensorFlow Lite).
- **PyTorch:** Favoured for rapid prototyping, research, and experimentation due to its flexibility and ease of use.

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

Two common use cases for Jupyter Notebooks in AI development are:

- **Interactive model experimentation and visualization:** Jupyter Notebooks allow developers to write and execute code in real time, visualize data and model outputs, and tweak parameters on the fly making them ideal for rapid prototyping, testing different algorithms, and visualizing results such as accuracy graphs or confusion matrices.
- **Creating reproducible and shareable data science reports:** Notebooks combine executable code, documentation, and visualizations in a single document. This unified format supports reproducibility, enables clear communication of methods and results, and makes it easy to collaborate or share findings with other researchers or stakeholders.

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

spaCy enhances NLP tasks compared to basic Python string operations by providing advanced, pre-trained pipelines that handle complex linguistic analysis such as tokenization, part-of-speech tagging, named entity recognition, and dependency parsing—all with high speed and accuracy. Unlike simple string splits or regular expressions, spaCy understands language

structure and context, enabling it to extract meaningful information (like entities and relationships) from text efficiently and reliably. This results in more accurate and scalable NLP solutions than what can be achieved with basic string manipulation in Python.

Q4: . Comparative Analysis Compare Scikit-learn and TensorFlow in terms of: Target applications (e.g., classical ML vs. deep learning). Ease of use for beginners. Community support.

Comparative Analysis: Scikit-learn vs TensorFlow

Aspect	**Scikit-learn**	**TensorFlow**
Target Applications	Best for classical machine learning (e.g., regression, classification, clustering, SVM, etc.). Handles structured/tabular data and smaller datasets efficiently.	Designed for deep learning and large-scale machine learning (e.g., neural networks, computer vision, NLP, reinforcement learning). Excels with unstructured data and massive datasets.
Ease of Use for Beginners	Highly beginner-friendly due to its simple, consistent API and clear documentation. Ideal for rapid prototyping and experimentation.	Moderate learning curve; more complex setup, especially for custom deep learning models. Keras API (within TensorFlow) improves usability, but still requires deeper understanding.
Community Support	Strong, mature, and supportive community, especially for classical machine learning tasks.	Very strong and active community, particularly for deep learning, with extensive resources and industry backing.