Part 1: Theoretical Understanding (40%)
1. Short Answer Questions

Q1: Explain the primary differences between TensorFlow and PyTorch. When would you choose one over the other?
Imagine you're building a house. **PyTorch** is like working with clay—it's flexible, easy to shape, and lets you tweak things as you go. **TensorFlow**, on the other hand, is like working with LEGO bricks—it's structured, optimized for large projects, and great for scaling up.

Some of the Key Differences Include:
- ✓ PyTorch feels more intuitive, like writing Python code naturally while TensorFlow used to be more rigid, but TensorFlow 2.x made it easier to work with.
- ✓ PyTorch lets you test and modify things dynamically, making it great for research while TensorFlow originally required defining everything upfront, but now it supports eager execution.
- ✓ TensorFlow shines in production—think mobile apps, web services, and large-scale AI systems while PyTorch is catching up but is still more popular in research labs.
- ✓ TensorFlow scales better across multiple GPUs, making it ideal for massive datasets. PyTorch is more flexible but may need extra optimization for large-scale training.

When then do we Choose Each
- ✓ **Use PyTorch** if you're experimenting, researching, or want an easy-to-debug framework
- ✓ **Use TensorFlow** if you're deploying AI models in real-world applications or need enterprise-level scalability.

Q2: Describe two use cases for Jupyter Notebooks in AI development.
Jupyter Notebooks are like interactive workspaces for AI developers, making experimentation and visualization seamless. Here are two key use cases:

1. Exploratory Data Analysis (EDA)
Before training a model, AI developers need to understand their data. Jupyter Notebooks allow them to:
- ✓ Load datasets and visualize trends using matplotlib and seaborn.
- ✓ Perform preprocessing steps like handling missing values and feature engineering.
- ✓ Run interactive queries to refine insights dynamically.

2. Model Experimentation & Hyperparameter Tuning
Instead of running entire scripts repeatedly, Jupyter Notebooks let developers:
- ✓ Test different architectures and hyperparameters in separate cells.
- ✓ Compare results side by side without restarting the whole process.
- ✓ Use AI-powered tools like JupyterAI to generate code and optimize models

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

SpaCy is like an experienced language expert compared to Python's simple text operations. Instead of just breaking sentences apart with .split(), spaCy **understands** the structure and meaning of words. It knows that "New York" is a place, "Tesla" is a company, and "run" can mean exercise or a software command.

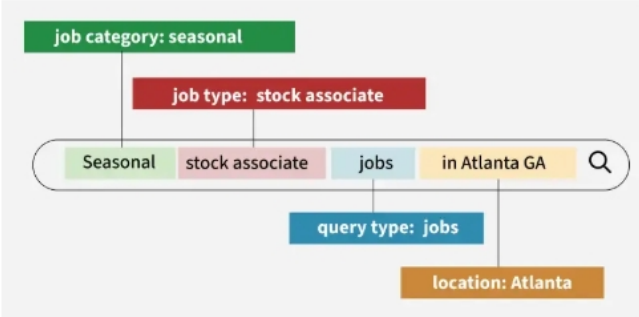
How Then does SpaCy Make NLP Smarter

- ✓ **Pretrained Pipelines**: Built-in tools for tokenization, named entity recognition (NER), part-of-speech tagging, and dependency parsing.
- ✓ **Efficiency**: Written in Cython, making text processing incredibly fast compared to looping through strings in Python.
- ✓ **Context Awareness**: Instead of blindly searching for words, spaCy understands relationships—so "bank" in "river bank" won't be confused with a financial institution.

Python | Named Entity Recognition (NER) using spaCy

Last Updated : 03 Apr, 2025

Named Entity Recognition (NER) is used in Natural Language Processing (NLP) to identify and classify important information within unstructured text. These "named entities" include proper nouns like people, organizations, locations and other meaningful categories such as dates, monetary values and products. By tagging these entities, we can transform raw text into structured data that can be analyzed, indexed or used in applications.



Representation of Named Entity Recognition

2. 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.

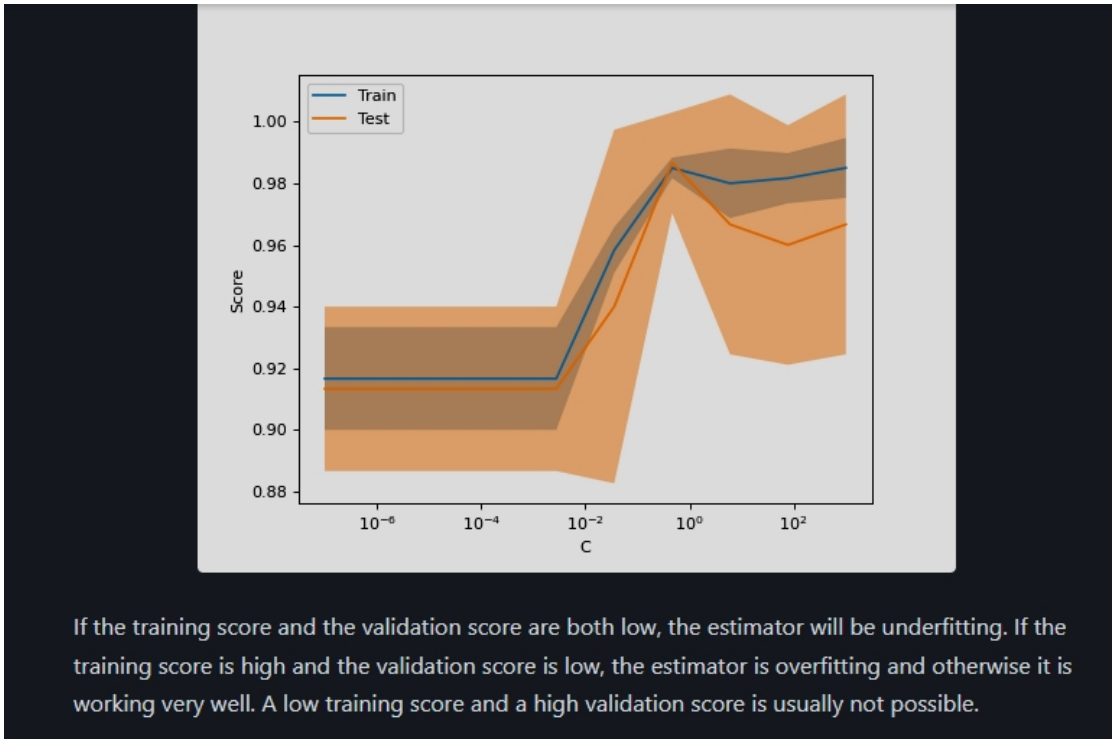
Feature	Scikit-learn	TensorFlow
Target Applications	Classical machine learning (e.g., regression, clustering, decision trees)	Deep learning (e.g., neural networks, image recognition, NLP)
Ease of Use for Beginners	Simple API, great for quick implementation	More complex, requires understanding of tensors and neural networks
Community Support	Strong support for traditional ML, widely used in academia	Large deep learning community, backed by Google, extensive documentation
Model Outputs	Accuracy metrics, confusion	Loss curves, accuracy graphs,

Feature	Scikit-learn	TensorFlow
	matrices, feature importance	neural network visualizations

Model Outputs

Scikit-learn: Accuracy Graph for a Classification Model

Scikit-learn provides accuracy metrics and confusion matrices to evaluate model performance.



TensorFlow tracks loss reduction over epochs, helping optimize neural networks

Ethical Reflection

AI models must prioritize **fairness, accountability, and transparency**:

- **Bias & Fairness:** Scikit-learn models are often more interpretable, reducing bias risks. TensorFlow's deep learning models can inherit biases from training data.
- **Explainability:** Scikit-learn's decision trees and linear models are easier to interpret than deep neural networks.
- **Privacy Concerns:** TensorFlow's deep learning applications raise concerns about data security, especially in facial recognition and surveillance AI.