

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

TensorFlow and PyTorch are two of the most popular open-source libraries for building and training deep learning models. While they both serve the same purpose, they have some key differences in their approach and philosophy.

A breakdown in simple terms:

#### Primary Differences

Computational Graphs (How they build the "recipe" for calculations): PyTorch: Dynamic Graphs (Define-by-Run)

Imagine you're cooking and you write down each step as you do it. If you decide to change something mid-way, you just adjust your next step. PyTorch builds the computational graph "on the fly" as you run your code. This means the graph can change with each step or iteration.

Analogy: Flexible, interactive cooking where you can improvise.

Benefits: Easier to debug (you see errors immediately), more flexible for complex and changing model architectures (like recurrent neural networks with varying sequence lengths), and feels more "Pythonic" (like regular Python code).

TensorFlow: Static Graphs (Define-and-Run) - Historically

Imagine you write down the entire recipe before you start cooking, down to every ingredient and every action. You can't change anything once you start following the recipe. Historically, TensorFlow required you to define the entire computational graph first, and then you would "run" data through it.

Analogy: A rigid, pre-planned cooking process where everything is set in stone beforehand.

Benefits: Can be more optimized for performance and deployment in large-scale production environments because the entire graph is known beforehand, allowing for global optimizations. Note: TensorFlow 2.0 introduced "Eager Execution" which brings a dynamic graph capability, making it much more similar to PyTorch in this regard. However, the static graph option is still there for those who prefer it.

Ease of Use and "Pythonic" Feel:

PyTorch: Generally considered more "Pythonic" and intuitive, especially for those already familiar with Python. It feels more like writing regular Python code.

Debugging is often simpler because you can use standard Python debugging tools. TensorFlow: Historically, it had a steeper learning curve, feeling a bit less like traditional Python. However, with TensorFlow 2.0 and the integration of Keras (a high-level API), it has become much more user-friendly and easier to get started with.

Deployment and Production Readiness:

TensorFlow: Has historically had a stronger ecosystem for deploying models into production, especially for large-scale applications. Tools like TensorFlow Serving (for deploying models as APIs), TensorFlow Lite (for mobile and embedded devices), and TensorFlow.js (for web browsers) make it very robust for real-world applications.

PyTorch: Has made significant strides in this area with tools like TorchServe, but TensorFlow is still often seen

as having an edge for enterprise-level production deployments.

Community and Adoption:

PyTorch: Gained significant popularity in the research and academic communities due to its flexibility and ease of use for experimentation and rapid prototyping. Many new research papers often release their code in PyTorch.

TensorFlow: Has a very large and mature community, particularly in industry and large-scale applications, largely due to Google's backing and its early dominance.

When to Choose One Over the Other:

Choose PyTorch if: You're in research or academia: Its dynamic graph and Pythonic nature make it excellent for experimentation, rapid prototyping, and trying out new ideas. You prioritize flexibility and easy debugging: If you need to frequently change your model architecture or want to step through your code line-by-line like a normal Python program, PyTorch is often more convenient.

You prefer a more "Pythonic" coding style: If you're comfortable with Python and want a deep learning framework that feels like a natural extension of it, PyTorch might be a better fit. You're working with smaller to medium-sized projects: While PyTorch can handle large-scale projects, its primary strength lies in its flexibility for rapid development.

Choose TensorFlow if: You're building large-scale production applications: TensorFlow's robust deployment tools (TensorFlow Serving, Lite, etc.) and its strong emphasis on optimization for production environments make it a solid choice for deploying models to users.

You need cross-platform deployment: If you plan to deploy your models on various platforms like mobile devices, web browsers, or in distributed systems, TensorFlow's ecosystem provides more comprehensive support.

You prefer a more structured API (even with Eager Execution): Even with Eager Execution, TensorFlow still offers a more opinionated and structured way of building models, especially with Keras, which can be beneficial for larger teams and long-term maintenance.

You want to leverage Google's ecosystem (e.g., TPUs): TensorFlow has native and highly optimized support for Google's Tensor Processing Units (TPUs), which can significantly speed up training for very large models.

In summary:

Both TensorFlow and PyTorch are powerful and constantly evolving. The "best" choice often comes down to your specific project needs, your team's familiarity with either framework, and your personal preference for how you like to develop. Many organizations and researchers even use both depending on the phase of the project (e.g., PyTorch for research and prototyping, then TensorFlow for production deployment).

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

Jupyter Notebooks are incredibly versatile tools in AI development, acting as interactive environments where you can combine code, text, visualizations, and equations. Here are two prominent use cases:

Use Case 1: Exploratory Data Analysis (EDA) and Preprocessing

Description: Before you can even think about building an AI model, you need to understand your data. This involves exploring its characteristics, identifying patterns, spotting anomalies, and cleaning it up for model

consumption. Jupyter Notebooks are perfectly suited for this iterative process.

**Benefit for AI Development:** A thorough EDA and preprocessing phase in a Jupyter Notebook ensures that the data fed into your AI model is clean, well-understood, and in the correct format, leading to more robust and accurate models. It helps avoid "garbage in, garbage out" scenarios.

## Use Case 2: Model Prototyping and Experimentation

**Description:** Jupyter Notebooks provide an excellent environment for rapidly building, training, and evaluating different AI models. This is where you can quickly test various algorithms, hyperparameter settings, and model architectures without the overhead of setting up complex scripts or projects for each iteration. **Benefit for AI Development:** Jupyter Notebooks enable a fast feedback loop for model development. Data scientists and AI engineers can quickly experiment with different ideas, compare model performances, and refine their approaches, significantly accelerating the research and development phase of AI projects.

How does spaCy enhance NLP tasks compared to basic Python string operations? spaCy offers a vastly superior and more efficient approach for any real-world NLP task.

In essence, while basic Python string operations allow you to manipulate strings, spaCy allows you to understand and process language in a meaningful, efficient, and scalable way. It provides the heavy lifting for linguistic analysis, freeing developers to focus on higher level AI applications rather than reinventing the wheel of linguistic parsing

### Limitations of Basic Python String Operations for NLP:

Basic Python string methods like `split()`, `replace()`, `lower()`, `upper()`, `strip()`, `find()`, and regular expressions (`re` module) are good for:

Simple text cleaning: Removing extra spaces, converting to lowercase, finding simple patterns.

Basic tokenization: Splitting a sentence by spaces to get "words" (but this is often flawed).

Crude keyword matching: Checking if a specific word or phrase exists in a text. However, they fall short dramatically for most NLP tasks because they lack any understanding of language.

No Linguistic Context: String operations treat text as just a sequence of characters. They have no concept of words, sentences, parts of speech, meaning, or grammatical relationships.

Rule-Based and Brittle: You'd have to write endless, complex, and often inaccurate rules (using `if/else` statements and `regex`) to handle the nuances of language. For example, splitting a sentence by periods to get sentences fails with "Dr. Smith" or "U.S.A.".

Laborious for Complex Tasks: Imagine trying to extract all proper nouns, identify verbs and their subjects, or understand sentiment using only string methods. It would be an incredibly tedious, error-prone, and practically impossible task.

Inefficient and Slow: Manually implementing linguistic rules with string operations would be extremely slow, especially for large volumes of text, as they don't leverage optimized algorithms or pre-trained models.

How spaCy Enhances NLP Tasks:

spaCy is an industrial-strength NLP library that provides a comprehensive suite of tools built on robust linguistic models. It goes far beyond simple string manipulation by understanding the structure and meaning of text.

Here are key ways spaCy enhances NLP compared to basic string operations: Named Entity Recognition (NER):

Basic String Ops: You'd need complex regex patterns or hardcoded lists to find names, locations, dates – highly inaccurate and limited.

spaCy: Identifies and classifies "named entities" in text, such as persons, organizations, locations, dates, monetary values, etc., using pre-trained models. This is fundamental for information extraction

Intelligent Tokenization:

Basic String Ops: `text.split(' ')` might split "don't" into "don't", or miss punctuation attached to words.

spaCy: Automatically breaks text into meaningful tokens (words, punctuation, numbers) while intelligently handling contractions (e.g., "don't" becomes "do" and "n't"), hyphens, and other complex cases based on language-specific rules.

Part-of-Speech (POS) Tagging:

Basic String Ops: No way to determine if a word is a noun, verb, adjective, etc. You'd have to manually create lists of words, which is unfeasible and ignores context.

spaCy: Assigns grammatical tags to each token (e.g., NOUN, VERB, ADJ, PROPN for proper noun) based on its context within the sentence, powered by pre-trained statistical models.

## 2. Comparative Analysis

Compare Scikit-learn and TensorFlow in terms of:

- 1.) Target applications (e.g., classical ML vs. deep learning).
- 2.) Ease of use for beginners.
- 3.) Community support.

Let's compare Scikit-learn and TensorFlow in simple terms, focusing on their key differences:

### 1. Target Applications (What they're best for) Scikit-learn (often shortened to "sklearn"):

Best for: Classical Machine Learning (ML). Think of tasks like predicting house prices, classifying emails as spam or not, grouping similar customers, or finding hidden patterns in data. It's excellent for working with structured, tabular data (like spreadsheets).

Examples of algorithms: Linear Regression, Logistic Regression, Decision Trees, Random Forests, Support Vector Machines (SVMs), K-Means Clustering, Principal Component Analysis (PCA).

Analogy: It's like a well-stocked toolbox for common carpentry tasks. You've got hammers, saws, drills – everything you need for typical building projects.

TensorFlow:

Best for: Deep Learning and Neural Networks. This involves more complex tasks like recognizing objects in images, understanding and generating human language (like chatbots), or building recommendation systems with very large and complex data.

It's designed to handle huge datasets and leverage powerful hardware (like GPUs). Examples of algorithms: Convolutional Neural Networks (CNNs) for images, Recurrent Neural Networks (RNNs) for sequences (text,

time series), Transformers (for advanced NLP).

Analogy: It's like a sophisticated factory for building advanced robots. It has specialized machinery and infrastructure for highly complex, large-scale manufacturing.

## 2.Ease of Use for Beginners Scikit-learn:

Very easy to use for beginners. It has a consistent and straightforward way of doing things (e.g., `model.fit()`, `model.predict()`). You can often get a basic model up and running with just a few lines of code. It abstracts away a lot of the complex math.

Why it's easy: Its API (the way you interact with it) is simple and logical, and it's well-documented with many examples. It doesn't require you to understand the "inside" workings of the algorithms in detail to use them effectively.

Analogy: It's like driving a car with an automatic transmission. You just put it in drive and go.

## TensorFlow:

Can be more complex, but has become much easier. Historically, it had a steeper learning curve because it gave you a lot of control over the "nuts and bolts" of neural networks. However, with the integration of Keras (a high-level API) into TensorFlow 2.0, it's become significantly more user-friendly. You can now build deep learning models with fewer lines of code, similar to Scikit-learn.

Why it can be harder (originally): You often had to define the "computational graph" (the steps of your model) explicitly, which required a deeper understanding of how neural networks work.

Analogy: It's like driving a race car. You can get a lot more performance and customization, but it might take more practice to master, even with some automatic features.

## 3.Community Support Scikit-learn:

Excellent and mature community support. Being around for a long time, it has a vast amount of tutorials, examples, Stack Overflow answers, and active forums. It's widely used in academia and for traditional data science tasks.

Why it's strong: Its stability, clear documentation, and widespread adoption in teaching basic machine learning have fostered a very helpful and comprehensive community.

## TensorFlow:

Massive and highly active community support. Backed by Google, TensorFlow has an enormous global community of developers, researchers, and companies using it. This means tons of tutorials, official documentation, GitHub repositories, and forums. Why it's strong: Its stability, clear documentation, and widespread adoption in teaching basic machine learning have fostered a very helpful and comprehensive community.

## In a nutshell:

Scikit-learn is your go-to for standard, "off-the-shelf" machine learning tasks and a great starting point for beginners.

TensorFlow is for powerful, custom deep learning models, especially when you're working with complex data (images, text) and need to scale up for production.

Many machine learning projects will actually use both – Scikit-learn for initial data preparation and classical models, and TensorFlow (or PyTorch) for more advanced deep learning components

## Part 2: Practical Implementation (50%) Task 1:

Classical ML with Scikit-learn Dataset: Iris

Species Dataset

Goal:

Preprocess the data (handle missing values, encode labels). Train a decision tree classifier to predict iris species.

Evaluate using accuracy, precision, and recall.

Deliverable: Python script/Jupyter notebook with comments explaining each step.

```
# Convert to DataFrame for easy viewing
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target
df.head()
```

```
[2]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```

# Make predictions
y_pred = model.predict(X_test)

# Evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision (macro):", precision_score(y_test, y_pred, average='macro'))
print("Recall (macro):", recall_score(y_test, y_pred, average='macro'))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

```

Accuracy: 1.0

Precision (macro): 1.0

Recall (macro): 1.0

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	1.00	1.00	9
2	1.00	1.00	1.00	11
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

## Task 2: Deep Learning with TensorFlow/PyTorch Dataset: MNIST

### Handwritten Digits

Goal:

- 1.) Build a CNN model to classify handwritten digits.
- 2.) Achieve >95% test accuracy.
- 3.) Visualize the model's predictions on 5 sample images.

Deliverable: Code with model architecture, training loop, and evaluation.



```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
1688/1688 — 48s 28ms/step - accuracy: 0.8965 - loss: 0.3439 - val_accuracy: 0.9828 - val_loss: 0.0570
Epoch 2/5
1688/1688 — 45s 27ms/step - accuracy: 0.9835 - loss: 0.0535 - val_accuracy: 0.9857 - val_loss: 0.0462
Epoch 3/5
1688/1688 — 84s 28ms/step - accuracy: 0.9891 - loss: 0.0342 - val_accuracy: 0.9877 - val_loss: 0.0403
Epoch 4/5
1688/1688 — 80s 27ms/step - accuracy: 0.9932 - loss: 0.0216 - val_accuracy: 0.9910 - val_loss: 0.0338
Epoch 5/5
1688/1688 — 81s 26ms/step - accuracy: 0.9947 - loss: 0.0165 - val_accuracy: 0.9902 - val_loss: 0.0384
313/313 — 4s 12ms/step - accuracy: 0.9864 - loss: 0.0400
Test Accuracy: 0.9894999861717224
1/1 — 0s 107ms/step
```

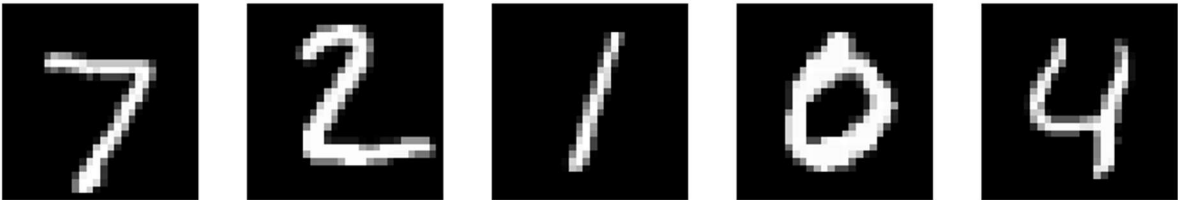
Predicted: 7

Predicted: 2

Predicted: 1

Predicted: 0

Predicted: 4



### Task 3: NLP with spaCy

Text Data: User reviews from Amazon Product Reviews. Goal:

- 1.) Perform named entity recognition (NER) to extract product names and brands.
- 2.) Analyze sentiment (positive/negative) using a rule-based approach.
- 3.) Deliverable: Code snippet and output showing extracted entities and sentiment.

```

    for ent in doc.ents:
        print(f" - {ent.text} ({ent.label_})")
    print("\n")

```

## Named Entity Recognition:

Review: I love the new Apple iPhone 14 - it's fast and sleek!

- Apple (ORG)

Review: The Samsung TV has stunning colors but the sound is average.

Review: Logitech mouse stopped working after two weeks. Very disappointed.

- Logitech (ORG)

- two weeks (DATE)

Review: This Sony camera takes amazing pictures. Worth every penny.

- Sony (ORG)

Review: The Dell laptop heats up quickly and battery life is poor.

```
review_lower = review.lower()
```

```
sentiment = "Neutral"
```

```
if any(word in review_lower for word in positive_keywords):
```

```
    sentiment = "Positive"
```

```
elif any(word in review_lower for word in negative_keywords):
```

```
    sentiment = "Negative"
```

```
print(f"Review: {review}\nSentiment: {sentiment}\n")
```

## Rule-based Sentiment:

Review: I love the new Apple iPhone 14 - it's fast and sleek!

Sentiment: Positive

Review: The Samsung TV has stunning colors but the sound is average.

Sentiment: Positive

Review: Logitech mouse stopped working after two weeks. Very disappointed.

Sentiment: Negative

Review: This Sony camera takes amazing pictures. Worth every penny.

Sentiment: Positive

Review: The Dell laptop heats up quickly and battery life is poor.

Sentiment: Negative