# **Q1: Primary Differences Between TensorFlow and PyTorch**

1. Computation Graph

TensorFlow:

- Static graph (define-then-run) by default.
- Supports dynamic graphs via eager execution.
PyTorch:
- Dynamic graph (define-by-run) by default.
- More intuitive for debugging and iterative development.
2. API Design
TensorFlow:
- Multi-layered API (low-level TF ops + high-level Keras).
- Steeper learning curve.
PyTorch:
- Pythonic, object-oriented design.
- Simpler and more intuitive for Python developers.
3. Deployment
TensorFlow:
- Production-ready tools (TF Serving, TF Lite, TF.js).
- Superior for mobile/edge deployment.
PyTorch:
- Relies on TorchServe/ONNX for deployment (rapidly improving).
- Traditionally stronger in research than production.

4. Community & Ecosystem
TensorFlow:
- Industry-dominated (Google, large-scale systems).
- Extensive production tooling (TFX, Kubeflow).
PyTorch:
- Research-dominated (academia, latest papers).
5. Debugging & Visualization
TensorFlow:
- TensorBoard (advanced tracking, profiling).
PyTorch:
- Supports TensorBoard + lightweight alternatives (e.g., Weights & Biases).
- Easier debugging due to dynamic graphs.
- Preferred for rapid prototyping and state-of-the-art models.
When to Choose One Over the Other
Choose TensorFlow if:
- Deploying models to production (web/mobile/edge).
- Building end-to-end ML pipelines (TFX).
- Working with large-scale distributed systems.
Example: Deploying a model to Android via TF Lite.
Choose PyTorch if:

- Rapid research prototyping or academia.

- Prioritizing debugging flexibility (dynamic graphs).
- Leveraging latest research models (e.g., Hugging Face transformers).

Example: Experimenting with a novel neural architecture.

### Q2: Two Use Cases for Jupyter Notebooks in Al Development

- 1. Interactive Prototyping & Debugging:
- Execute code incrementally (cell-by-cell) to test model components.
- Visualize results immediately without rerunning entire scripts.

Example: Debugging a CNN by inspecting feature maps after each convolutional layer.

- 2. Collaborative Documentation & Reproducibility:
- Combine code, visualizations, equations, and text in one document.
- Share notebooks to ensure reproducibility.

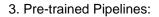
Example: Documenting an NLP pipeline for team review.

### Q3: How spaCy Enhances NLP Tasks vs. Basic Python String Operations

- 1. Linguistic Intelligence:
- spaCy uses statistical models to understand context.

Example: Identifies entities (ORG, PERSON); string ops need complex regex.

- 2. Efficiency & Scalability:
- Built-in tokenization, lemmatization, and parsing with high speed.
- String operations struggle with complex language structures.



- Ready-to-use models for NER, POS tagging, similarity detection.
- String operations require manual implementation.