PART 1 — Theoretical Understanding

Q1: TensorFlow vs PyTorch

Feature	TensorFlow	PyTorch
Developer	Google	Facebook (Meta)
Computation Graph	Uses *static* computation graphs (definethen-run).	Uses *dynamic* computation graphs (define-by-run).
Ease of Debugging	Harder to debug due to static graph.	Easier debugging using Python control flow and print statements.
Deployment	Excellent for production (via TensorFlow Serving, TensorFlow Lite, and TensorFlow.js).	Great for research; deployment improving via TorchServe and ONNX.
Ecosystem	Integrates tightly with Keras (high-level API).	More flexible for custom deep learning architectures.
When to Choose	- For large-scale deployment or mobile apps.- When you need pretrained models in TF Hub.	- For research and experimentation.- When you want more Pythonic, flexible code.

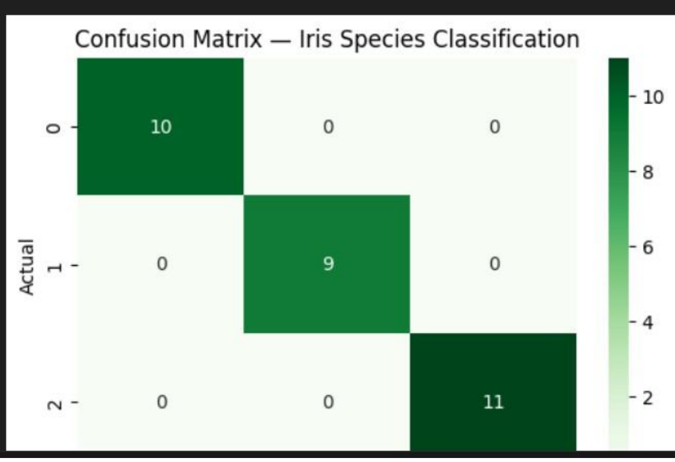
Q2: Two Use Cases for Jupyter Notebooks in AI

1. **Interactive Data Exploration:**
    - Helps visualize datasets using `matplotlib`, `seaborn`, and `plotly`.
    - Example: Checking class distributions in the Iris dataset.
2. **Experiment Tracking & Documentation:**
    - You can mix code, results, and markdown documentation in one place. ○ Useful for comparing ML models and sharing research findings.

Q3: How spaCy Enhances NLP vs Basic Python Strings

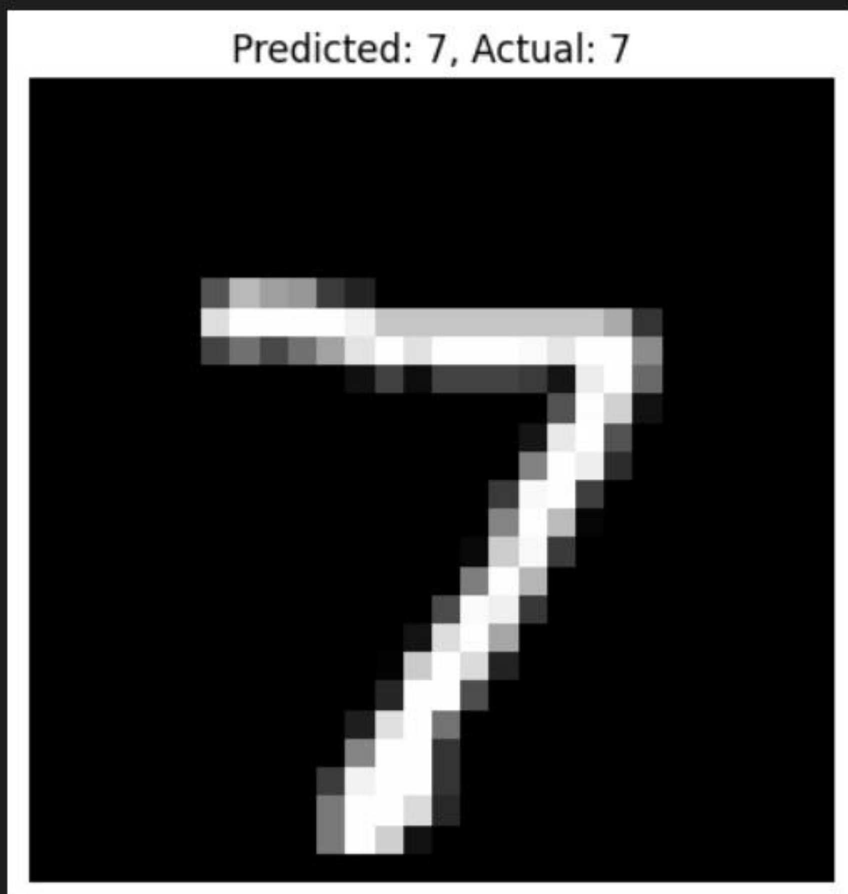
Task	Basic Python	spaCy
Tokenization	Uses `.split()` — poor for punctuation.	Uses linguistic models — accurate tokenization.
POS Tagging / NER	Not supported.	Built-in models for part-of-speech tagging and named entity recognition.
Lemmatization	Manual rules.	Uses language models to get word lemmas.
Efficiency	Slow for large texts.	Cython-based — very fast and optimized.

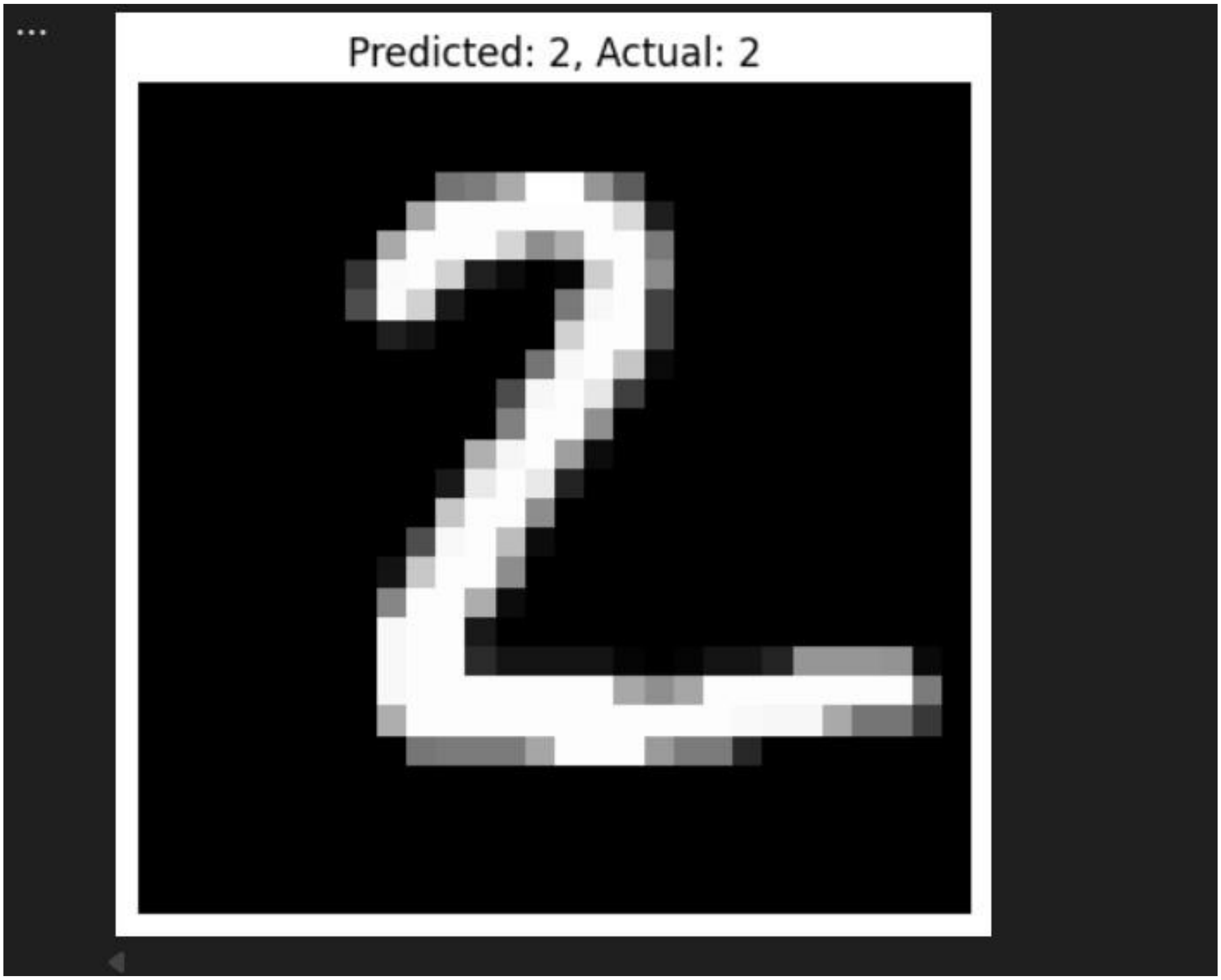
```
...  Accuracy: 1.0
     Precision: 1.0
     Recall: 1.0

     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

...
```

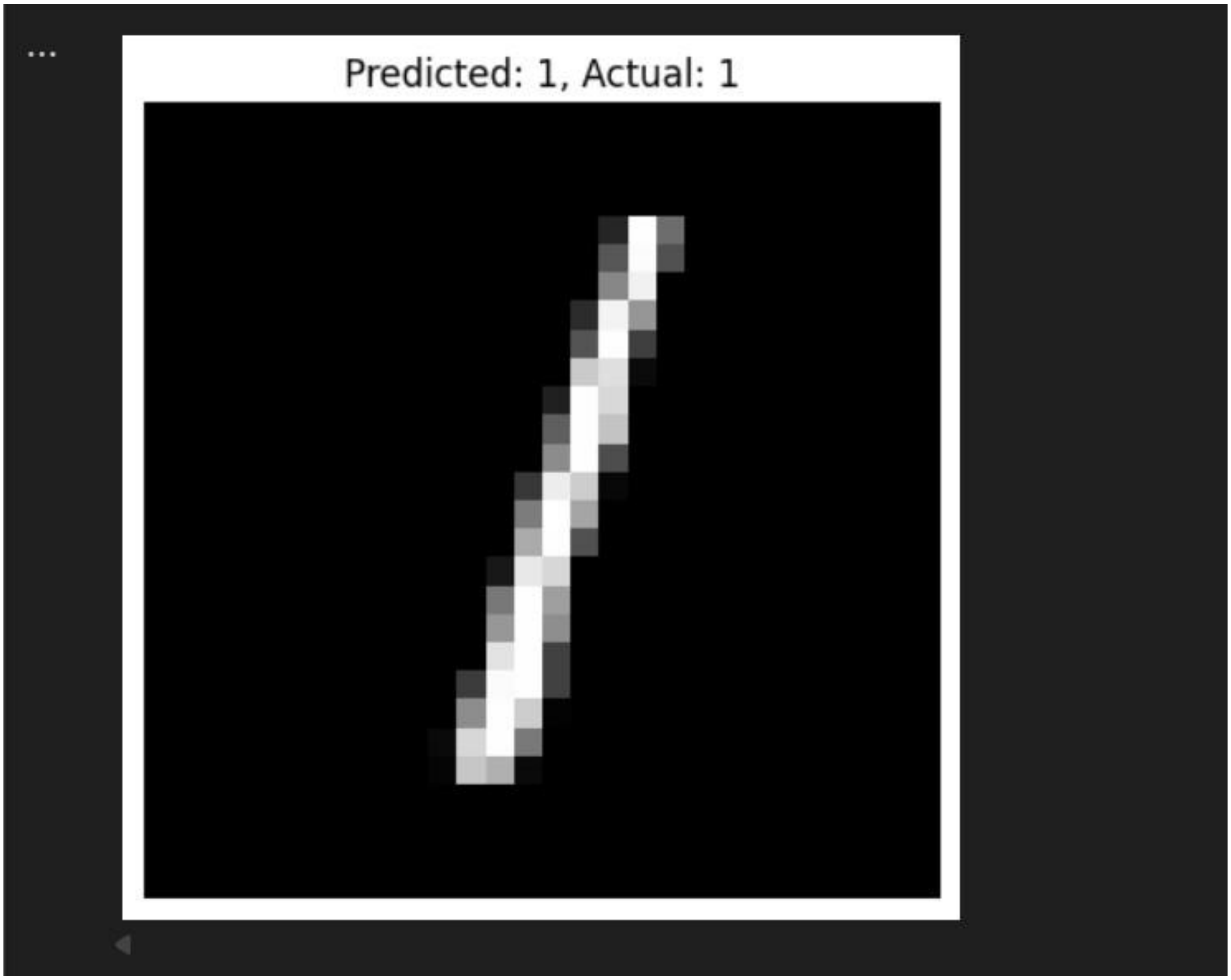


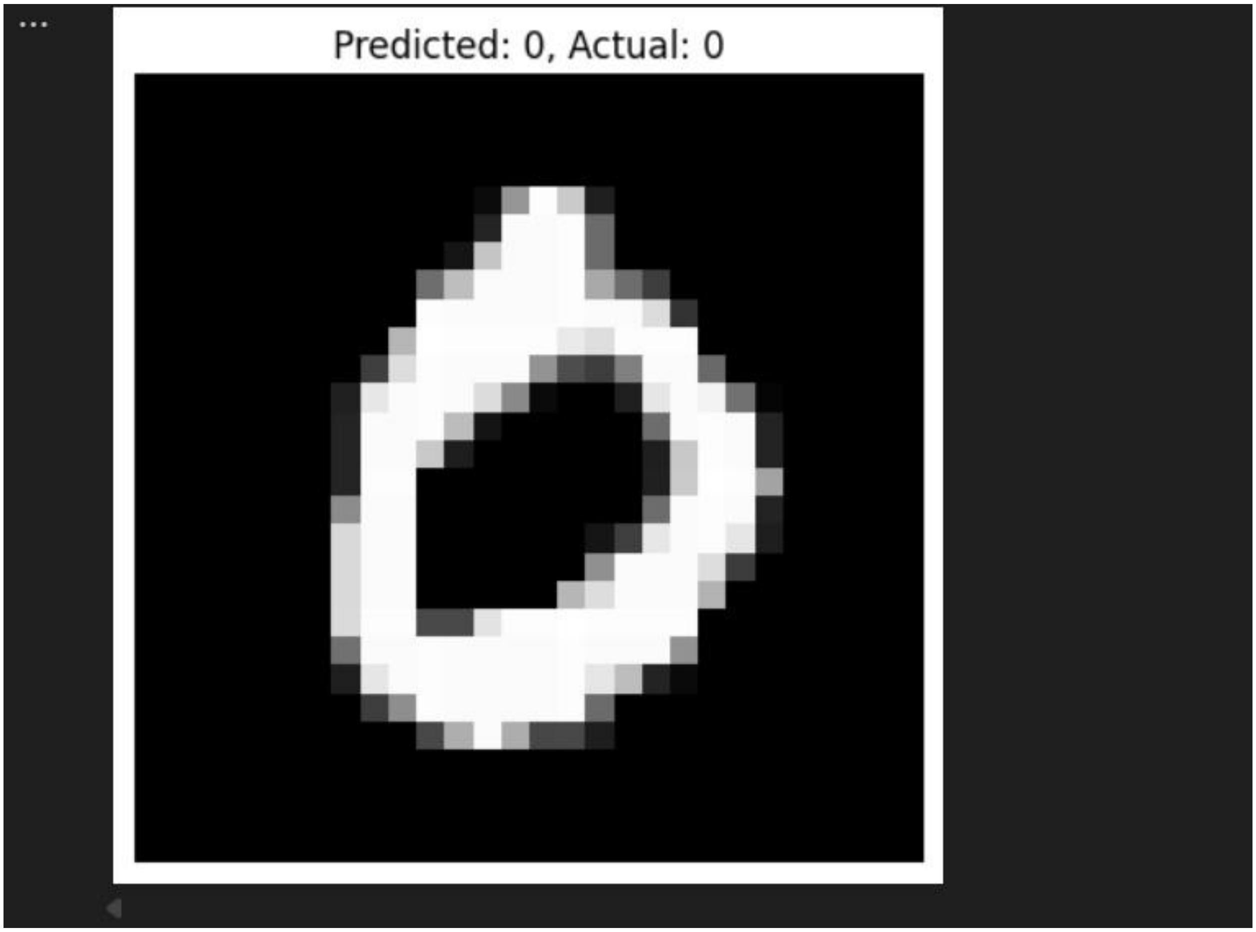
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 6s 0us/step
C:\Users\Admin\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 32s 17ms/step - accuracy: 0.9585 - loss: 0.1362 - val_accuracy: 0.9850 - val_loss: 0.0522
Epoch 2/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 29s 17ms/step - accuracy: 0.9856 - loss: 0.0458 - val_accuracy: 0.9898 - val_loss: 0.0387
Epoch 3/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 28s 16ms/step - accuracy: 0.9900 - loss: 0.0311 - val_accuracy: 0.9908 - val_loss: 0.0306
Epoch 4/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 26s 15ms/step - accuracy: 0.9927 - loss: 0.0227 - val_accuracy: 0.9883 - val_loss: 0.0439
Epoch 5/5
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 26s 15ms/step - accuracy: 0.9948 - loss: 0.0159 - val_accuracy: 0.9902 - val_loss: 0.0384
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9902 - loss: 0.0313
```

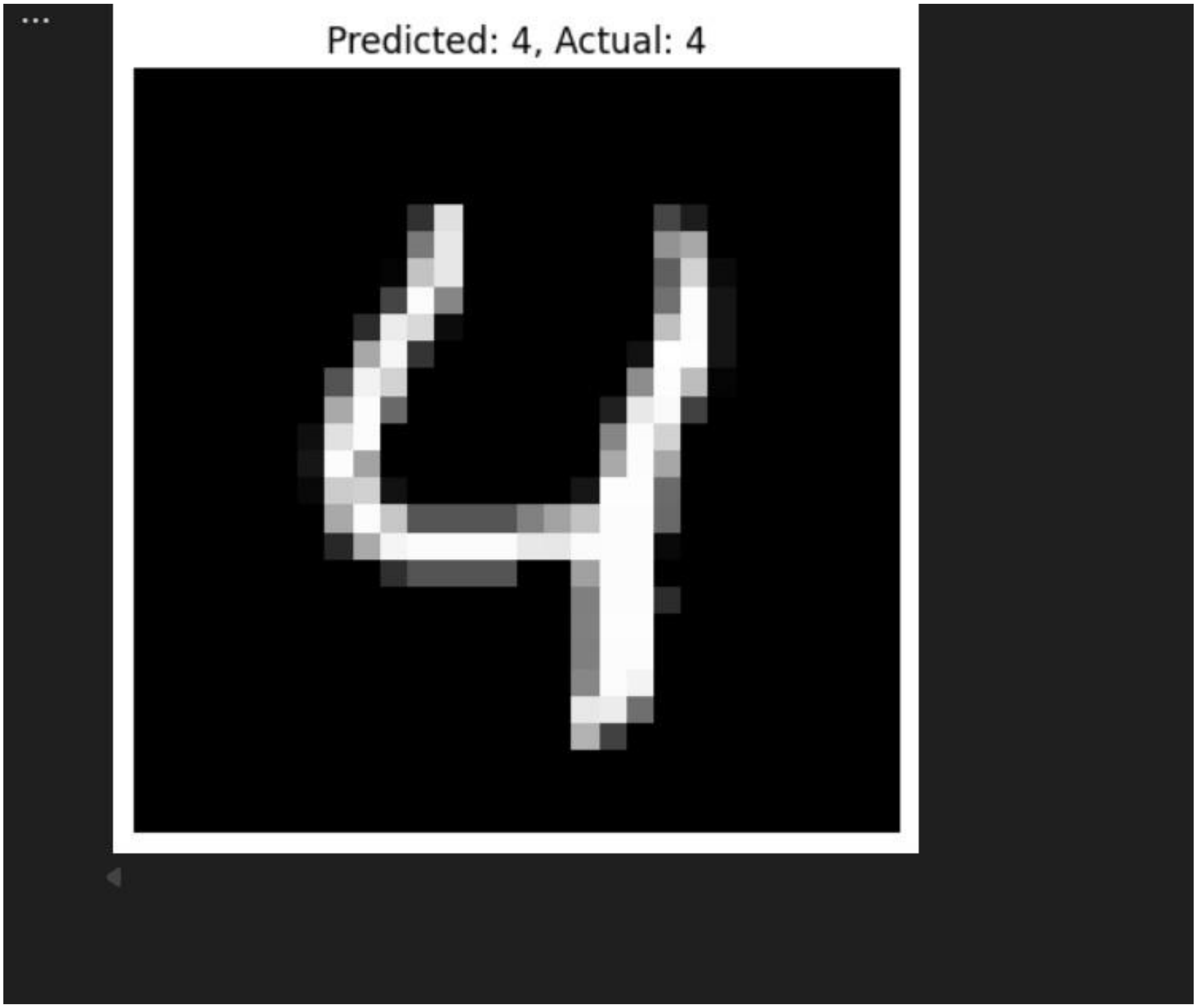
```
✅ Test Accuracy: 0.9902
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step
```




...
Predicted: 2, Actual: 2


Predicted: 1, Actual: 1


Predicted: 0, Actual: 0


Predicted: 4, Actual: 4

```
...
📝 Review: I love my new iPhone 15 Pro! The camera is amazing.
Named Entities:
  - 15 (CARDINAL)
Sentiment: Positive 😀

📝 Review: This Samsung TV broke after a week. Terrible quality.
Named Entities:
  - Samsung (ORG)
  - a week (DATE)
Sentiment: Negative 😞

📝 Review: Sony headphones have incredible sound and comfort.
Named Entities:
  - Sony (ORG)
Sentiment: Positive 😀

📝 Review: The Dell laptop is fast but the battery dies too soon.
Named Entities:
  - Dell (ORG)
Sentiment: Positive 😀
```

Part 3 — Ethics & Optimization

MNIST (image classifier)

Potential biases

- Most MNIST digits are handwritten by a limited demographic — variations in handwriting styles, cultural scripts, or scanner/camera devices may not be represented.
- Model may perform worse on digit styles it has not seen (domain bias).

Mitigations

- **Dataset augmentation**: rotate, scale, add noise to training images so the model generalizes.
- **Collect diverse data**: include samples from different writers, devices, and countries.
- **Evaluate by subgroup**: compute accuracy per subgroup (e.g., by stroke thickness or digit style) to identify disparities.
- **Use fairness tools**: TensorFlow Fairness Indicators can show metric slices (e.g., performance on different user groups). Use them to discover where model underperforms.
- **Human-in-the-loop**: have fallback manual review for low-confidence predictions.

Amazon Reviews (NLP)

Potential biases

- Reviews may overrepresent certain products, brands, demographics, or opinion types (e.g., more negative reviews get posted).
- Language style differences (non-native speakers) may be misinterpreted by sentiment rules.

Mitigations

- **Diverse training data**: include reviewers from different regions and languages (or detect language and apply different models).
- **Debias embeddings**: when using embeddings, apply debiasing techniques (e.g., neutralize certain protected attributes).
- **Rule augmentation**: for rule-based sentiment, expand rules and include negation handling; combine with a learned sentiment model and calibrate with human labels.
- **Confidence thresholding**: flag low-confidence or ambiguous predictions for manual review.

2) How tools can help

- **TensorFlow Fairness Indicators**: visualize model metrics across slices (e.g., per subgroup), detect disparities, and monitor fairness over time.
- **spaCy rule-based systems**: you can write custom patterns to extract product names/brands; these rules are transparent and easy to audit for fairness and biases (compare rule outputs across groups).

3) Troubleshooting Challenge — Common TensorFlow Bugs & Fixes

Bug class: dimension mismatch

Buggy example (common):

```
# Wrong: using softmax + from_logits=True with already softmaxed outputs

logits = model(x)          # model returns softmax probabilities loss =

tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

loss_value = loss(y_true, logits)
```

Fix:

- Either return raw logits (no softmax) from model and keep `from_logits=True`, or keep probabilities and set `from_logits=False`.

  # Option A: model returns logits (no final softmax), keep from_logits=True loss

  = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

  # Option B: model returns probabilities (softmax on final layer), set from_logits=False

loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False) **Bug class: wrong label encoding**

Buggy example:

Using sparse categorical crossentropy but labels are one-hot vectors

y_train_onehot = tf.keras.utils.to_categorical(y_train, 10)

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')

Fix: • If labels are one-hot → use `categorical_crossentropy`.

- If labels are integer class ids → use `sparse_categorical_crossentropy`.

Use categorical_crossentropy for one-hot labels model.compile(loss='categorical_crossentropy', optimizer='adam')

OR convert labels to ints for sparse loss y_train_int = np.argmax(y_train_onehot, axis=1)

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')

Use categorical_crossentropy for one-hot labels model.compile(loss='categorical_crossentropy', optimizer='adam')

OR convert labels to ints for sparse loss

y_train_int = np.argmax(y_train_onehot, axis=1)

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')

Bug class: shape mismatch for input

Buggy example:

```
# Model expects shape (batch, 28,28,1) but gets (batch,28,28)
x_train = x_train.reshape(-1, 28, 28)  # missing channel dim
```

Fix:

```
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32')
```

Bug class: dtype mismatch

Buggy example: `x = x.astype('int')          # ints into a model`

`expecting float`

Fix:

```
x = x.astype('float32') / 255.0
```