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?

Primary Differences:

- **Computational Graph:**
  - **TensorFlow:** Uses a **static computational graph**. This means the graph is defined *before* execution and then run. This allows for powerful optimizations and deployment to various platforms (mobile, embedded devices).
  - **PyTorch:** Uses a **dynamic computational graph** (also known as "define by run"). The graph is built on the fly as operations are executed. This offers greater flexibility for debugging and more intuitive model building, especially for research.
- **Debugging:**
  - **TensorFlow:** Debugging can be more challenging due to the static graph. Tools like tf.debugger exist, but it's not as straightforward as stepping through Python code.
  - **PyTorch:** Debugging is generally easier because the dynamic graph allows you to use standard Python debugging tools directly.
- **Ease of Use/Learning Curve:**
  - **TensorFlow:** Historically, TensorFlow had a steeper learning curve, especially with its lower-level APIs. TensorFlow 2.0 with Keras integration has significantly improved this.
  - **PyTorch:** Generally considered to have a more Pythonic and intuitive API, making it quicker to learn for those familiar with Python.
- **Deployment:**
  - **TensorFlow:** Has robust deployment options (TensorFlow Serving, TensorFlow Lite, TensorFlow.js) making it very strong for production environments.

    - **PyTorch:** While improving, PyTorch's deployment story was historically less mature than TensorFlow's. Tools like TorchScript and ONNX have significantly enhanced its deployment capabilities.
- **Community and Adoption:**
    - **TensorFlow:** Backed by Google, it has a very large and mature community, extensive documentation, and widespread industry adoption, particularly in large-scale production systems.
    - **PyTorch:** Backed by Facebook (Meta AI), it has rapidly gained popularity, especially in the research community, due to its flexibility and ease of use. Its community is also growing rapidly.

When to Choose One Over the Other:

- **Choose TensorFlow if:**
    - **You prioritize large-scale deployment and production:** Its robust deployment ecosystem (TensorFlow Serving, TF Lite) is a major advantage.
    - **You need strong multi-platform support:** Deploying to mobile, web, or embedded devices is a key requirement.
    - **You are working on a project that benefits from static graph optimizations:** Especially for very large models or distributed training where these optimizations can provide significant performance gains.
    - **You prefer a more established, industry-standard framework for certain applications.**
- **Choose PyTorch if:**
    - **You are a researcher or prefer rapid prototyping and experimentation:** The dynamic graph makes iterative development and debugging much faster.
    - **You value a more "Pythonic" and intuitive API:** If you're comfortable with Python, PyTorch's learning curve might be shallower.
    - **You frequently need to modify model architectures on the fly or work with complex, dynamic models.**
    - **You are just starting out with deep learning and want an easier entry point into building and understanding models.**

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

1. **Exploratory Data Analysis (EDA) and Data Preprocessing:** Jupyter Notebooks are excellent for the initial stages of AI development. Data scientists can use them to:
    - Load and inspect datasets (e.g., viewing the first few rows of a DataFrame).
    - Visualize data distributions and relationships using libraries like Matplotlib, Seaborn, or Plotly, directly within the notebook.
    - Perform data cleaning operations (handling missing values, outliers), feature engineering, and data transformation steps, seeing the immediate results of each operation.
    - Document their data exploration process with markdown cells, making the analysis reproducible and understandable for others.
2. **Model Prototyping, Training, and Evaluation:** Jupyter Notebooks provide an interactive environment for building and testing AI models. Developers can:
    - Write and execute small code snippets to define model architectures (e.g., a neural network in TensorFlow or PyTorch).
    - Train models incrementally and monitor training progress (loss, accuracy) in real-time.
    - Evaluate model performance using various metrics and visualize results (e.g., confusion matrices, ROC curves).
    - Experiment with different hyperparameters and instantly see the impact on model performance, facilitating rapid iteration and optimization.
    - Present their model's performance and key insights directly within the notebook, combining code, outputs, and explanations.

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

spaCy significantly enhances NLP tasks compared to basic Python string operations by providing a more robust, efficient, and linguistically-aware framework. Here's how:

1. **Linguistic Understanding (Beyond Simple Text Matching):**
    - **Tokenization:** While Python's split() can break text by spaces, spaCy's tokenizer understands language rules. It correctly handles punctuation, contractions ("don't" into "do" and "n't"), and even complex linguistic phenomena, producing meaningful tokens (words, punctuation, symbols).

   - **Part-of-Speech Tagging (POS Tagging):** spaCy can identify the grammatical role of each word (noun, verb, adjective, etc.). Basic string operations can't infer this. This is crucial for understanding sentence structure and for tasks like lemmatization.
   - **Lemmatization:** spaCy reduces words to their base form (e.g., "running," "ran," "runs" all become "run"). Python string operations would treat these as distinct. This helps in analyzing word meanings irrespective of their inflections.
   - **Named Entity Recognition (NER):** spaCy can identify and classify "named entities" in text, such as persons, organizations, locations, dates, etc. This is impossible with basic string operations, which can only search for exact string matches.
   - **Dependency Parsing:** spaCy can reveal the grammatical relationships between words in a sentence (e.g., which words modify which others). This provides a deep syntactic understanding that string operations cannot.
2. **Efficiency and Performance:**
   - spaCy is implemented in Cython, making it significantly faster and more memory-efficient than performing complex NLP operations purely with Python string methods and regex, especially on large text corpuses.
   - It's designed for production use, offering optimized pipelines for common NLP tasks.
3. **Pre-trained Models and Pipelines:**
   - spaCy comes with highly optimized, pre-trained statistical models for various languages. These models encapsulate years of NLP research and training, allowing users to perform complex tasks like POS tagging, NER, and dependency parsing out-of-box with high accuracy. Basic string operations require you to build all logic from scratch.
4. **Semantic Understanding (Word Vectors):**
   - Many spaCy models include word vectors (word embeddings), which are numerical representations of words that capture their semantic meaning. This allows for tasks like finding similar words or performing semantic similarity comparisons, which are far beyond the scope of basic string manipulations.
5. **Robustness and Handling Edge Cases:**

- Basic string operations often struggle with variations, misspellings, or complex sentence structures. spaCy's linguistic models are trained on vast amounts of data and are more robust to these real-world text complexities.

2. Comparative Analysis: Scikit-learn vs. TensorFlow

Here's a comparison of Scikit-learn and TensorFlow:

Feature/Aspect	**Scikit-learn**	**TensorFlow**
Target Applications	Primarily **Classical Machine Learning**.	Primarily **Deep Learning**.
	- **Supervised Learning:** Classification (e.g., Logistic Regression, SVM, Decision Trees, Random Forests), Regression (e.g., Linear Regression, Ridge, Lasso).	- **Neural Networks:** Convolutional Neural Networks (CNNs) for image processing, Recurrent Neural Networks (RNNs) for sequential data (text, time series), Transformers.
	- **Unsupervised Learning:** Clustering (e.g., K-Means, DBSCAN), Dimensionality Reduction (e.g., PCA, t-SNE).	- **Complex Architectures:** Generative Adversarial Networks (GANs), Reinforcement Learning, Large Language Models (LLMs).
	- **Model Selection & Preprocessing:** Cross-validation, hyperparameter tuning, feature scaling, imputation.	- **Large-scale Data & Distributed Training:** Designed to handle massive datasets and training across multiple GPUs or machines.

	- Ideal for structured, tabular data, and problems where traditional statistical methods are effective.	- Ideal for unstructured data (images, video, audio, text), problems requiring feature learning, and highly complex patterns.
Ease of Use for Beginners	**Very High.** - **Simple API:** Designed with a consistent and intuitive API (.fit(), .predict(), .transform()) across all models, making it easy to learn and apply different algorithms. - **Less Overhead:** Focuses purely on algorithms, abstracting away low-level computation details. No need to manage computational graphs or GPU memory explicitly. - **Excellent Documentation:** Comprehensive and well-organized documentation with numerous examples.	**Moderate to High (TensorFlow 2.0 with Keras).** - **TensorFlow 1.x:** Had a steeper learning curve due to its static graph philosophy and more verbose API. - **TensorFlow 2.0:** Significantly improved ease of use by making Keras its high-level API. This allows for more intuitive, eager execution and a more Pythonic experience, making it much more approachable for beginners interested in deep learning. - Still requires a conceptual understanding of neural networks, tensors, and computational graphs, which can be more complex than classical ML algorithms. Setting up GPU environments can also add complexity for beginners.

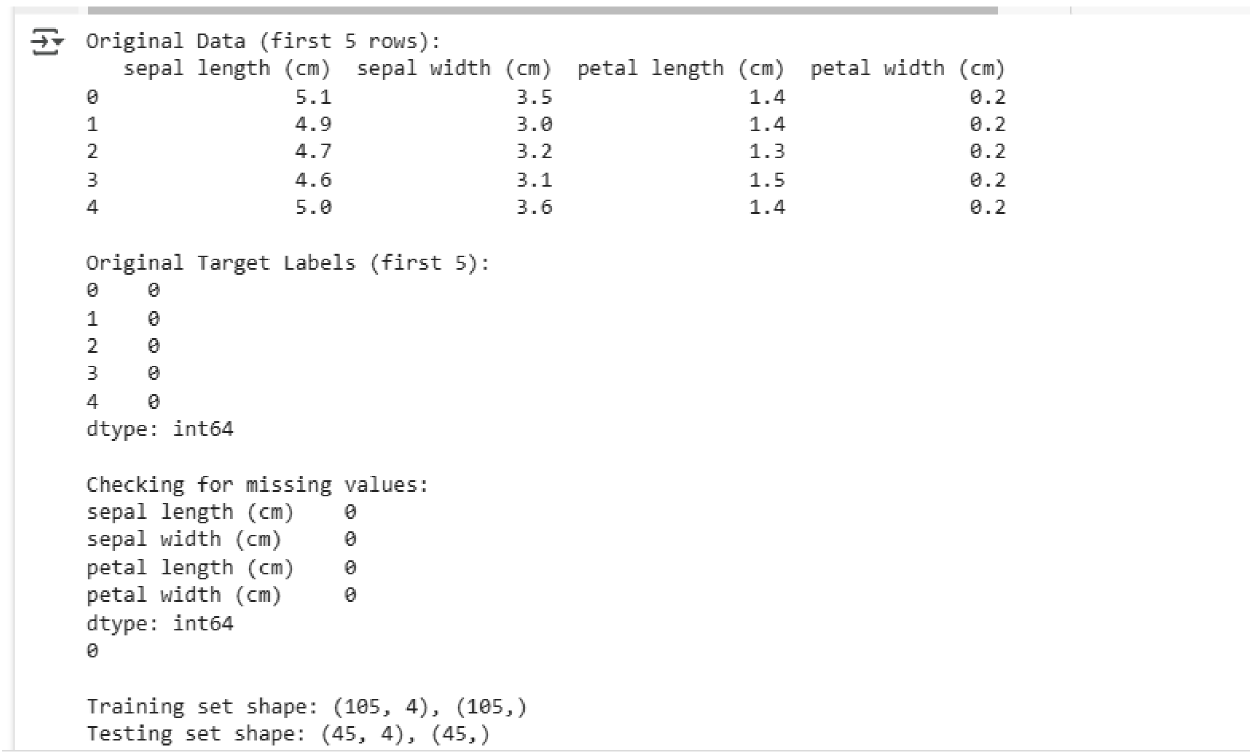
	- Good for quick prototyping of classical ML models.	- Can be more challenging for complete programming beginners due to the complexity of deep learning concepts themselves, even with a user-friendly API like Keras.
Community Support	**Excellent and Mature.**	**Excellent and Massive.**
	- **Active Development:** Continuously maintained and updated by a dedicated community of developers and researchers.	- **Backed by Google:** Benefits from significant resources and contributions from Google and a vast global community of researchers and engineers.
	- **Widespread Adoption:** Widely used in academia and industry for classical ML tasks, leading to a large user base.	- **Industry Standard:** One of the two leading deep learning frameworks (alongside PyTorch) used extensively in both research and production environments.
	- **Extensive Resources:** Abundance of tutorials, articles, Stack Overflow answers, and online courses.	- **Rich Ecosystem:** TensorFlow Hub (pre-trained models), TensorBoard (visualization), TensorFlow Lite (mobile/edge deployment), TensorFlow.js (web deployment) demonstrate a very comprehensive ecosystem.
	- The community focuses on practical applications of well-established ML algorithms.	- The community spans from fundamental deep learning research to cutting-edge production deployments,

with constant innovation and new architectures emerging

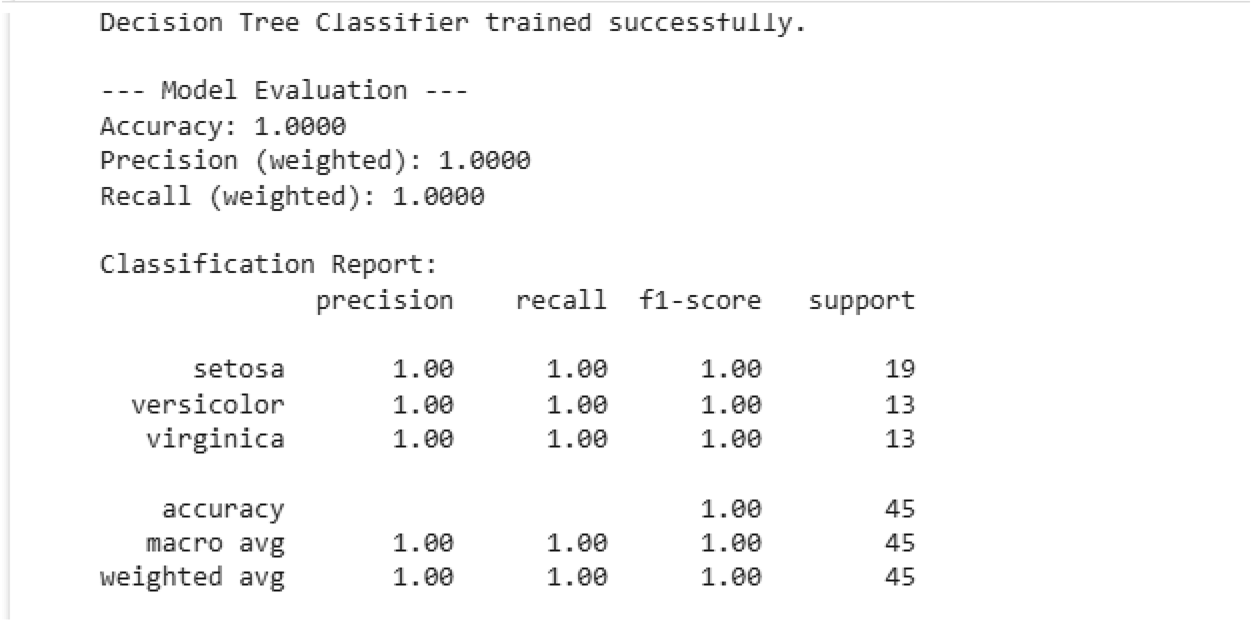
Part 2: Practical Implementation (50%)

Task 1: Classical ML with Scikit-learn

Dataset: Iris Species Dataset **Goal:** Preprocess data, train a Decision Tree, evaluate.



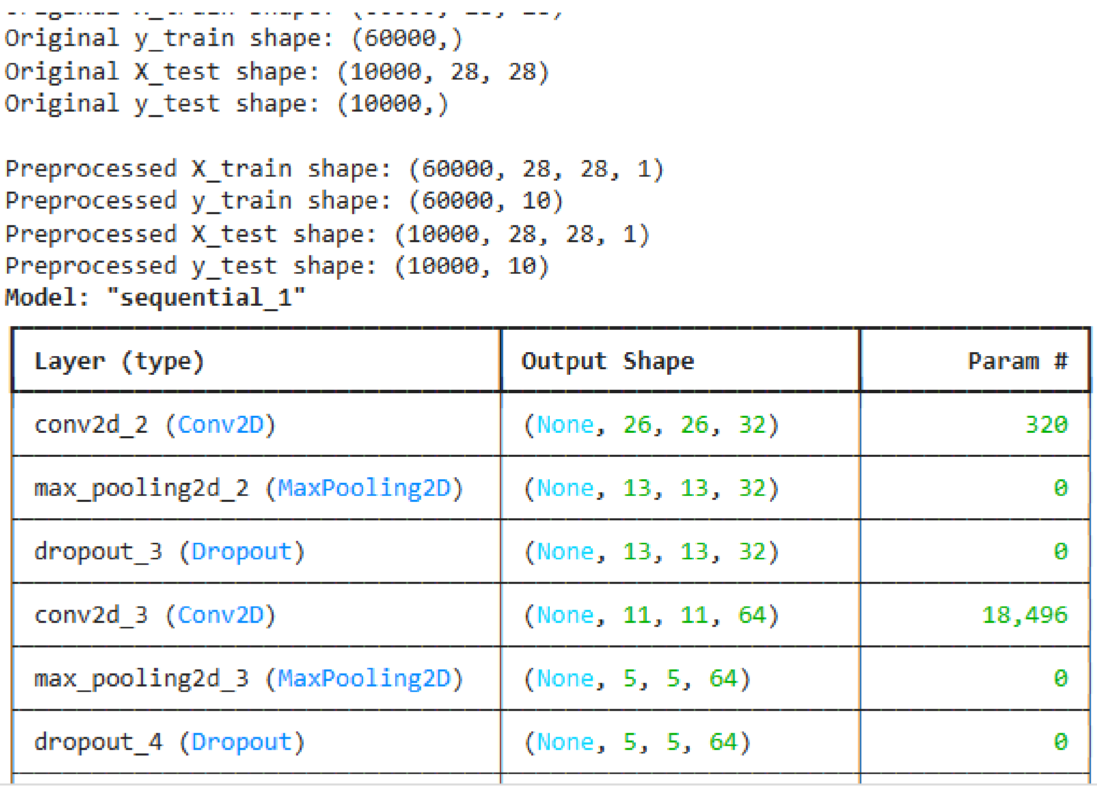
```
Original Data (first 5 rows):
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

Original Target Labels (first 5):
0    0
1    0
2    0
3    0
4    0
dtype: int64

Checking for missing values:
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
dtype: int64
0

Training set shape: (105, 4), (105,)
Testing set shape: (45, 4), (45,)
```



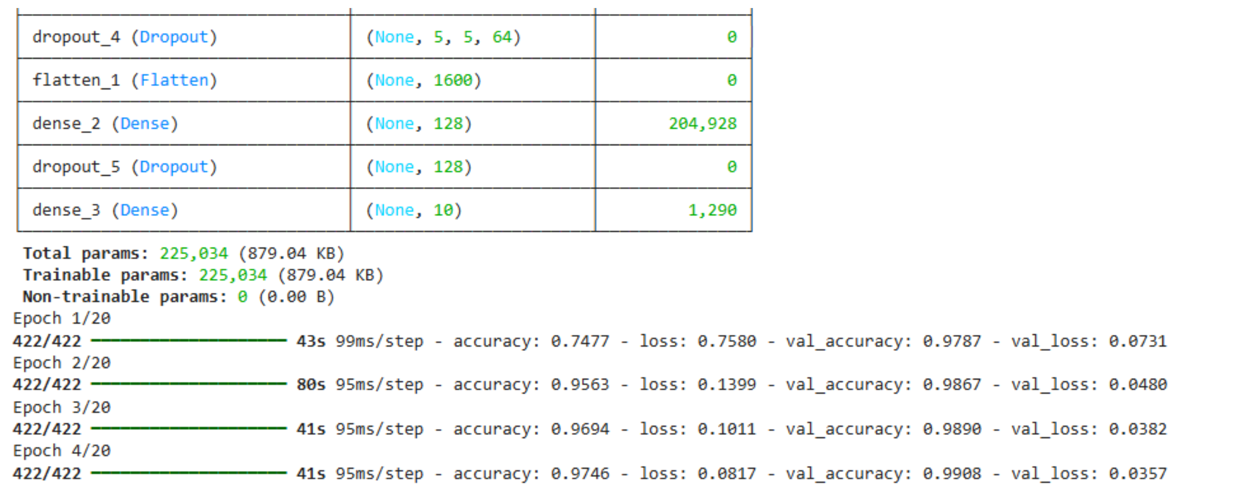
```
Decision Tree Classifier trained successfully.

--- Model Evaluation ---
Accuracy: 1.0000
Precision (weighted): 1.0000
Recall (weighted): 1.0000

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Task 2: Deep Learning with TensorFlow/PyTorch

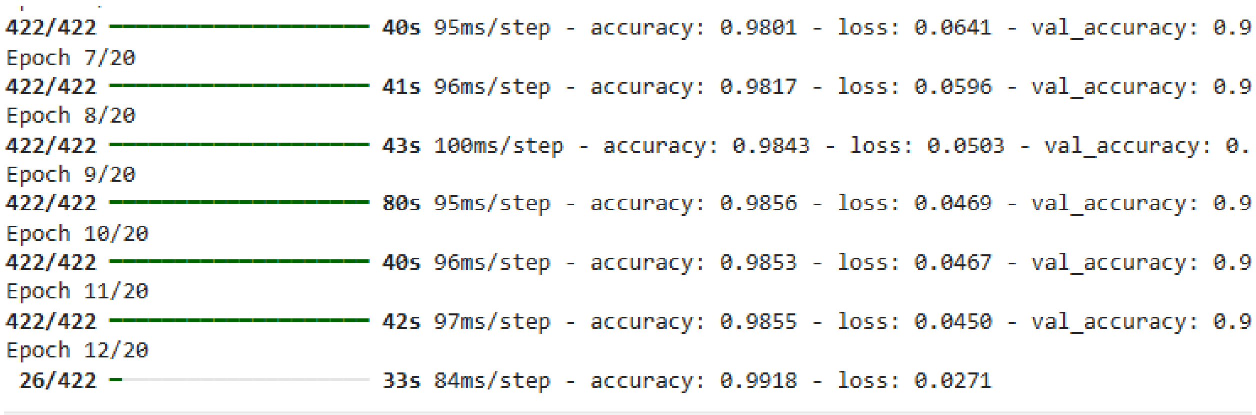
Dataset: MNIST Handwritten Digits **Goal:** Build a CNN, achieve >95% accuracy, visualize predictions.



```
Original y_train shape: (60000,)
Original X_test shape: (10000, 28, 28)
Original y_test shape: (10000,)

Preprocessed X_train shape: (60000, 28, 28, 1)
Preprocessed y_train shape: (60000, 10)
Preprocessed X_test shape: (10000, 28, 28, 1)
Preprocessed y_test shape: (10000, 10)
Model: "sequential_1"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_2 (Conv2D)               │ (None, 26, 26, 32)     │           320 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 13, 13, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_3 (Dropout)             │ (None, 13, 13, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_3 (Conv2D)               │ (None, 11, 11, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_3 (MaxPooling2D)  │ (None, 5, 5, 64)       │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_4 (Dropout)             │ (None, 5, 5, 64)       │             0 │
```



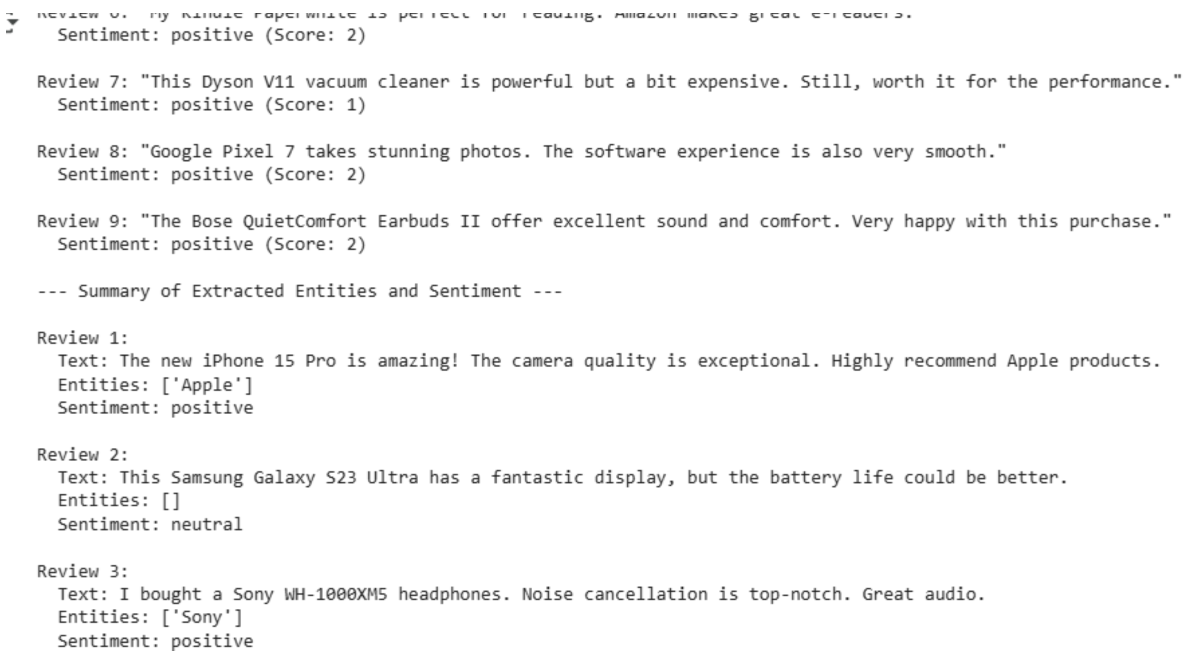
```
│ dropout_4 (Dropout)             │ (None, 5, 5, 64)       │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_1 (Flatten)             │ (None, 1600)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 128)            │       204,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_5 (Dropout)             │ (None, 128)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 10)             │         1,290 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 225,034 (879.04 KB)
 Trainable params: 225,034 (879.04 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/20
422/422 ━━━━━━━━━━━━━━━━━━━━ 43s 99ms/step - accuracy: 0.7477 - loss: 0.7580 - val_accuracy: 0.9787 - val_loss: 0.0731
Epoch 2/20
422/422 ━━━━━━━━━━━━━━━━━━━━ 80s 95ms/step - accuracy: 0.9563 - loss: 0.1399 - val_accuracy: 0.9867 - val_loss: 0.0480
Epoch 3/20
422/422 ━━━━━━━━━━━━━━━━━━━━ 41s 95ms/step - accuracy: 0.9694 - loss: 0.1011 - val_accuracy: 0.9890 - val_loss: 0.0382
Epoch 4/20
422/422 ━━━━━━━━━━━━━━━━━━━━ 41s 95ms/step - accuracy: 0.9746 - loss: 0.0817 - val_accuracy: 0.9908 - val_loss: 0.0357
```



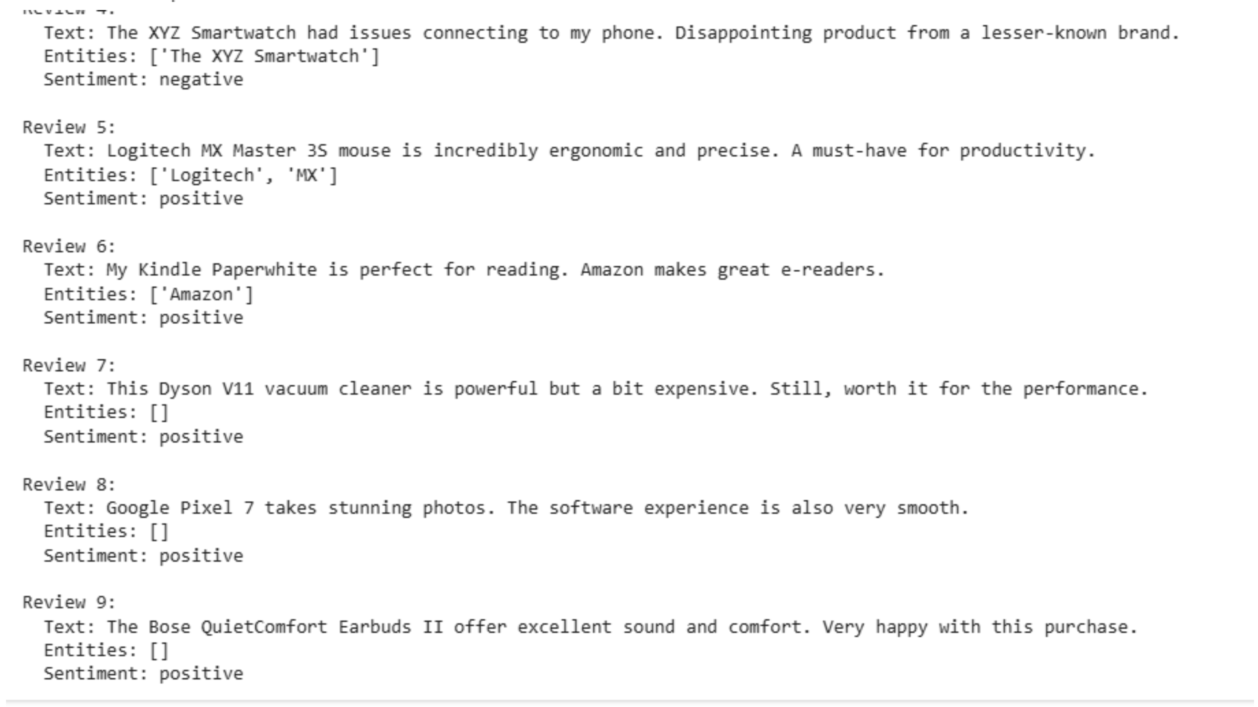
```
422/422 ━━━━━━━━━━━━━━━━━━━━ 40s 95ms/step - accuracy: 0.9801 - loss: 0.0641 - val_accuracy: 0.9
Epoch 7/20
422/422 ━━━━━━━━━━━━━━━━━━━━ 41s 96ms/step - accuracy: 0.9817 - loss: 0.0596 - val_accuracy: 0.9
Epoch 8/20
422/422 ━━━━━━━━━━━━━━━━━━━━ 43s 100ms/step - accuracy: 0.9843 - loss: 0.0503 - val_accuracy: 0.
Epoch 9/20
422/422 ━━━━━━━━━━━━━━━━━━━━ 80s 95ms/step - accuracy: 0.9856 - loss: 0.0469 - val_accuracy: 0.9
Epoch 10/20
422/422 ━━━━━━━━━━━━━━━━━━━━ 40s 96ms/step - accuracy: 0.9853 - loss: 0.0467 - val_accuracy: 0.9
Epoch 11/20
422/422 ━━━━━━━━━━━━━━━━━━━━ 42s 97ms/step - accuracy: 0.9855 - loss: 0.0450 - val_accuracy: 0.9
Epoch 12/20
 26/422 ━━━━━━━━━━━━━━━━━━━━ 33s 84ms/step - accuracy: 0.9918 - loss: 0.0271
```

Task 3: NLP with spaCy

Text Data: User reviews from Amazon Product Reviews. **Goal:** Perform NER, analyze sentiment using a rule-based approach.

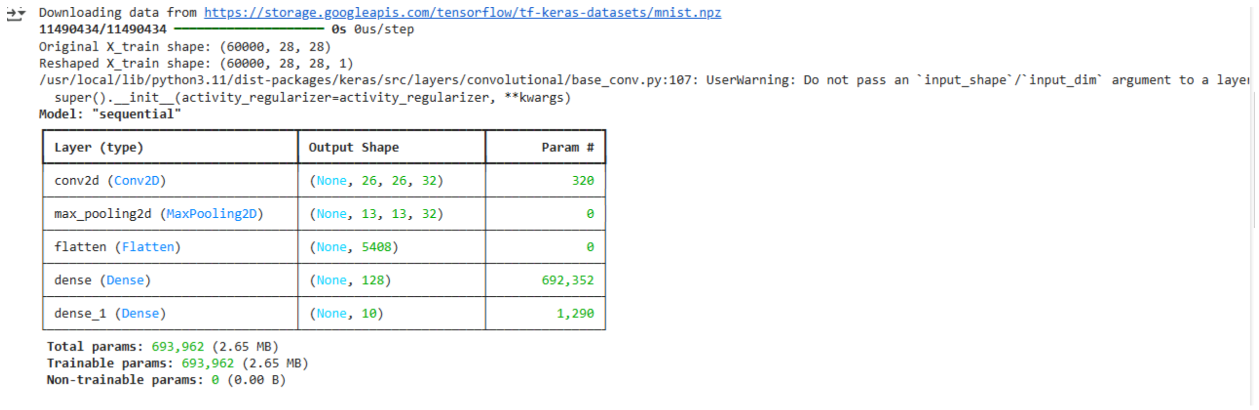


```
Review 6: "My Kindle Paperwhite is perfect for reading. Amazon makes great e-readers."
  Sentiment: positive (Score: 2)

Review 7: "This Dyson V11 vacuum cleaner is powerful but a bit expensive. Still, worth it for the performance."
  Sentiment: positive (Score: 1)

Review 8: "Google Pixel 7 takes stunning photos. The software experience is also very smooth."
  Sentiment: positive (Score: 2)

Review 9: "The Bose QuietComfort Earbuds II offer excellent sound and comfort. Very happy with this purchase."
  Sentiment: positive (Score: 2)

--- Summary of Extracted Entities and Sentiment ---

Review 1:
  Text: The new iPhone 15 Pro is amazing! The camera quality is exceptional. Highly recommend Apple products.
  Entities: ['Apple']
  Sentiment: positive

Review 2:
  Text: This Samsung Galaxy S23 Ultra has a fantastic display, but the battery life could be better.
  Entities: []
  Sentiment: neutral

Review 3:
  Text: I bought a Sony WH-1000XM5 headphones. Noise cancellation is top-notch. Great audio.
  Entities: ['Sony']
  Sentiment: positive
```

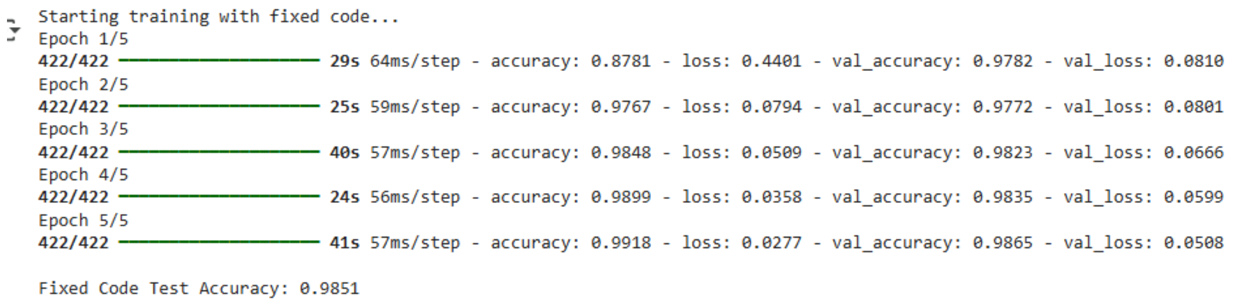


```
Review 4:
  Text: The XYZ Smartwatch had issues connecting to my phone. Disappointing product from a lesser-known brand.
  Entities: ['The XYZ Smartwatch']
  Sentiment: negative

Review 5:
  Text: Logitech MX Master 3S mouse is incredibly ergonomic and precise. A must-have for productivity.
  Entities: ['Logitech', 'MX']
  Sentiment: positive

Review 6:
  Text: My Kindle Paperwhite is perfect for reading. Amazon makes great e-readers.
  Entities: ['Amazon']
  Sentiment: positive

Review 7:
  Text: This Dyson V11 vacuum cleaner is powerful but a bit expensive. Still, worth it for the performance.
  Entities: []
  Sentiment: positive

Review 8:
  Text: Google Pixel 7 takes stunning photos. The software experience is also very smooth.
  Entities: []
  Sentiment: positive

Review 9:
  Text: The Bose QuietComfort Earbuds II offer excellent sound and comfort. Very happy with this purchase.
  Entities: []
  Sentiment: positive
```

Part 3: Ethics & Optimization



```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Original X_train shape: (60000, 28, 28)
Reshaped X_train shape: (60000, 28, 28, 1)
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a laye
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                  │ (None, 26, 26, 32)      │           320 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 13, 13, 32)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ flatten (Flatten)                │ (None, 5408)            │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dense (Dense)                    │ (None, 128)             │       692,352 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dense_1 (Dense)                  │ (None, 10)              │         1,290 │
└──────────────────────────────────┴─────────────────────────┴───────────────┘
 Total params: 693,962 (2.65 MB)
 Trainable params: 693,962 (2.65 MB)
 Non-trainable params: 0 (0.00 B)
```



```
Starting training with fixed code...
Epoch 1/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 29s 64ms/step - accuracy: 0.8781 - loss: 0.4401 - val_accuracy: 0.9782 - val_loss: 0.0810
Epoch 2/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 25s 59ms/step - accuracy: 0.9767 - loss: 0.0794 - val_accuracy: 0.9772 - val_loss: 0.0801
Epoch 3/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 40s 57ms/step - accuracy: 0.9848 - loss: 0.0509 - val_accuracy: 0.9823 - val_loss: 0.0666
Epoch 4/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 24s 56ms/step - accuracy: 0.9899 - loss: 0.0358 - val_accuracy: 0.9835 - val_loss: 0.0599
Epoch 5/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 41s 57ms/step - accuracy: 0.9918 - loss: 0.0277 - val_accuracy: 0.9865 - val_loss: 0.0508

Fixed Code Test Accuracy: 0.9851
```