

Technical Report: Fine-Tuning a Large Language Model

Submissions:

- Jupyter Notebook
- Presentation explaining what LoRA and PEFT
- Presentation explaining the code and project
- Streamlit app

Github Link

<https://github.com/selvintuscano/distilbert-lora-finetune-amazon-polarity>

Please find the code and all presentation on my github link

Recordings Link

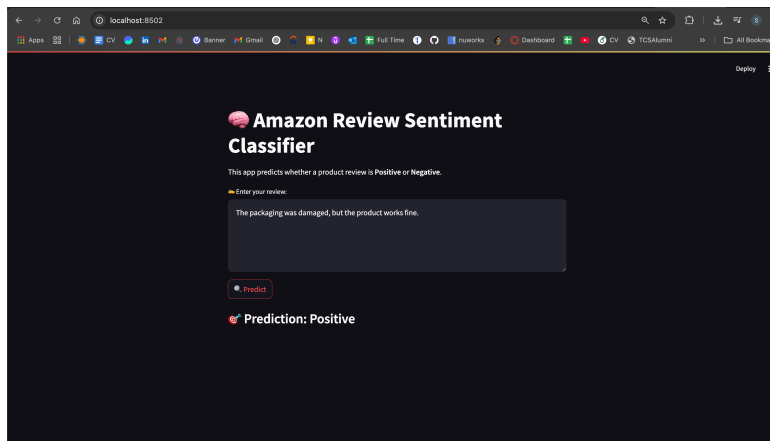
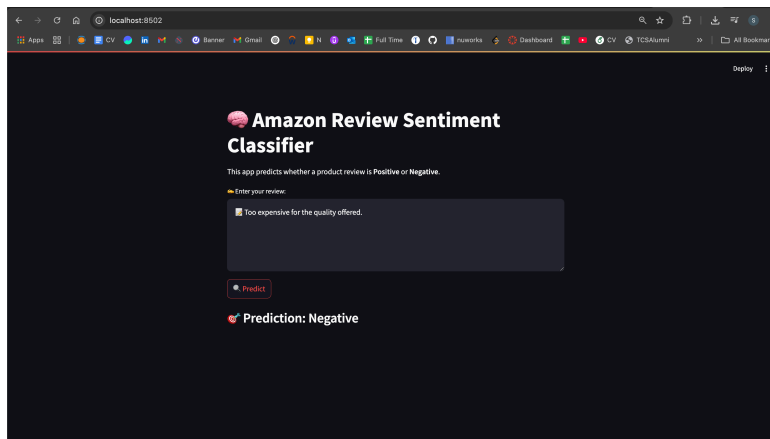
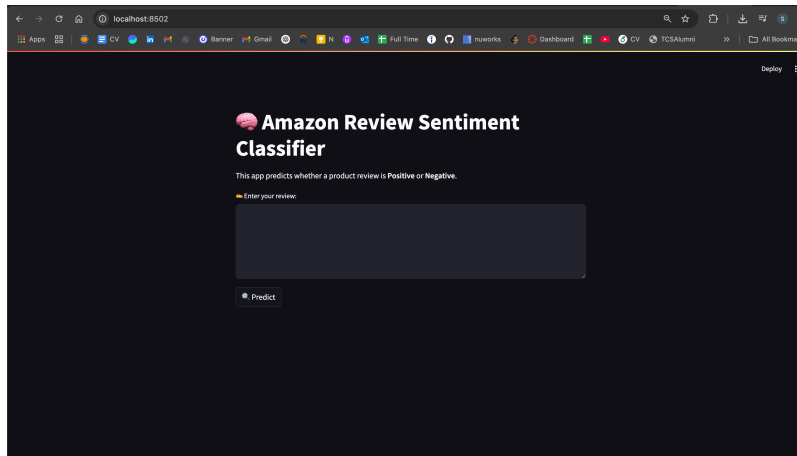
1) Presentation explaining what LoRA and PEFT –

https://drive.google.com/file/d/1G0e2izyMiGLTqED9_qmPLS0LnjILX698/view?usp=drive_sdk

2) Presentation explaining the code and project –

https://drive.google.com/file/d/1GHThc9iHC9uYcRtAeAyXo9LrPFqAThQc/view?usp=drive_link

Streamlit App



Have Explained everything in detail in the recordings

Also have performed all the required parameters for the assignment

Step 1: Introduction and Objective

In this project, we fine-tuned a pre-trained transformer model, DistilBERT, to perform sentiment classification on Amazon product reviews using the Amazon Polarity dataset. Given that large language models (LLMs) are typically trained on broad corpora, they often require task-specific adaptation to perform optimally in real-world settings.

To achieve efficient adaptation, we used Low-Rank Adaptation (LoRA), a parameter-efficient fine-tuning method, and integrated it using the Hugging Face `peft` library. This method allowed us to update less than 1% of the model's weights while achieving high performance.

Step 2: Dataset Preparation

The dataset used was the **Amazon Polarity** dataset, a binary sentiment dataset derived from millions of Amazon product reviews. For this project:

- We used **1,000 reviews for training** and **1,000 for testing**, a small subset for faster experimentation.
- Each review has a title, content, and a binary label (0 = Negative, 1 = Positive).
- Preprocessing steps included:
 - Renaming "content" to "text"
 - Shuffling with a fixed seed
 - Tokenizing with `distilbert-base-uncased`
 - Truncating to max length 512 tokens

Step 3: Model Selection - DistilBERT

We selected **DistilBERT** due to its:

- 60% faster inference time
- 40% fewer parameters compared to BERT
- Retention of ~97% downstream performance

DistilBERT was trained using knowledge distillation from BERT. The student model learns to mimic the teacher's (BERT's) outputs, optimizing both cross-entropy and KL divergence losses.

It contains 6 encoder layers instead of 12 and omits the NSP objective. This architecture makes it ideal for fine-tuning on medium-sized tasks under limited hardware

Step 4: LoRA - Low-Rank Adaptation

Traditional fine-tuning modifies all weights in a model — an expensive approach. **LoRA** instead freezes the base model and adds small trainable matrices to linear layers in the attention mechanism:

$$W' = W + \Delta W = W + AB \quad W' = W + \Delta W = W + AB$$

This results in a significant reduction in trainable parameters.

For example, for a 768x768 weight matrix, using $r=4$, the update size drops from 589,824 to just 6,144 parameters.

Step 5: PEFT - Efficient Fine-Tuning with Hugging Face

Hugging Face's `peft` library simplifies the injection of LoRA into transformer models. It allows developers to specify:

- Task type (e.g., sequence classification)
- Rank (r)
- Dropout
- Target modules (e.g., `q_lin`)

With just a few lines of code, the base model is wrapped with adapter layers, enabling lightweight training without full reparameterization.

Step 6: Fine-Tuning and Hyperparameter Tuning

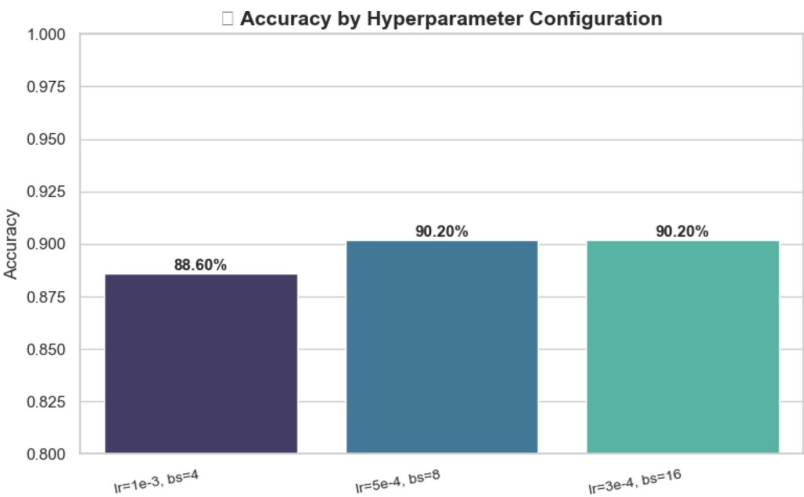
We fine-tuned the model using 3 sets of hyperparameters:

Hyperparameter	Values Tested	Why We Chose It
Learning Rate	1e-3, 5e-4, 3e-4	Controls how quickly the model learns. We tried a range from aggressive (1e-3) to conservative (3e-4) to balance convergence speed and stability.
Batch Size	4, 8, 16	Affects memory usage and model generalization. Smaller sizes like 4 can lead to noisy updates; larger sizes like 16 stabilize training.
Epochs	5	Based on visual inspection of training loss; sufficient for convergence on our small dataset.
LoRA Settings	r=4, lora_alpha=32, dropout=0.01	These settings reduce training cost while preserving performance. Lower r reduces parameters; higher alpha stabilizes learning.

- Learning Rates: 1e-3, 5e-4, 3e-4
- Batch Sizes: 4, 8, 16
- Epochs: 5 (fixed)

The best configuration was:

- lr = 5e-4, batch_size = 16
- Final test accuracy: 90.2%

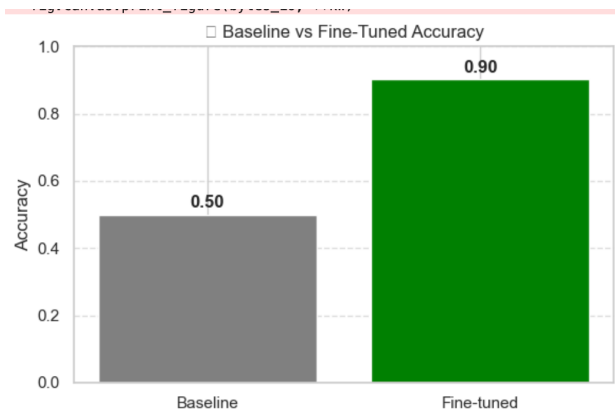


We used the `Trainer` class from Hugging Face Transformers with LoRA-wrapped model.

Step 7: Evaluation & Results

Accuracy

- Before Fine-Tuning: ~50%
- After Fine-Tuning: 90.2%

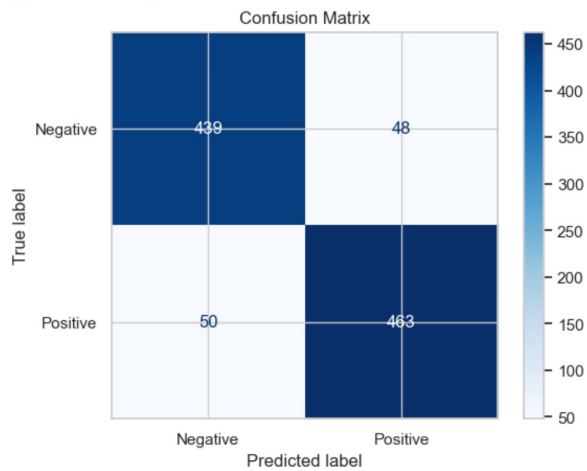


Classification Report

Class	Precision	Recall	F1-Score
Negative	0.90	0.90	0.90
Positive	0.91	0.90	0.90

Confusion Matrix Summary

✓ Manual Accuracy: 0.9020



- **TP:** 463 | **TN:** 439
- **FP:** 48 | **FN:** 50

Error Analysis

Common misclassifications were due to:

- Mixed sentiment in reviews
- Domain-specific phrases or sarcasm
- Summaries with multiple opinions

Step 8: Inference Pipeline

A simple prediction function was implemented:

```
[138]: def predict_sentiment(text):
        device = torch.device("cpu") # Use "mps" or "cuda" if you want to test those
        model.to(device)

        inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True, max_length=512)
        inputs = {k: v.to(device) for k, v in inputs.items()}

        with torch.no_grad():
            outputs = model(**inputs)
            logits = outputs.logits
            pred = torch.argmax(logits, dim=1).item()

        return "Positive" if pred == 1 else "Negative"

[140]: # Example 1
text1 = "I loved this product! It exceeded my expectations."
print(f"👉 Review: {text1}")
print(f"👉 Prediction: {predict_sentiment(text1)}")

# Example 2
text2 = "Terrible experience. It broke after two days."
print(f"👉 Review: {text2}")
print(f"👉 Prediction: {predict_sentiment(text2)}")
```

```
👉 Review: I loved this product! It exceeded my expectations.
👉 Prediction: Positive

👉 Review: Terrible experience. It broke after two days.
👉 Prediction: Negative
```

Ask user to input text in real-time

```
[145]: # Get user input from the notebook
user_input = input("Type a review to analyze sentiment:\n")
prediction = predict_sentiment(user_input)
print(f"👉 Your Review: {user_input}")
print(f"👉 Predicted Sentiment: {prediction}")
```

```
Type a review to analyze sentiment:
Terrible experience. It broke after two days.

👉 Your Review: Terrible experience. It broke after two days.
👉 Predicted Sentiment: Negative
```

Batch predictions (for multiple reviews)

```
[148]: reviews = [
    "It was just okay. Not great, not terrible.",
    "Worst purchase I've ever made.",
    "Absolutely loved it! Would buy again.",
    "The packaging was damaged, but the product works fine.",
    "Too expensive for the quality offered."
]

for review in reviews:
    print(f"👉 {review}")
    print(f"👉 {predict_sentiment(review)}\n")
```

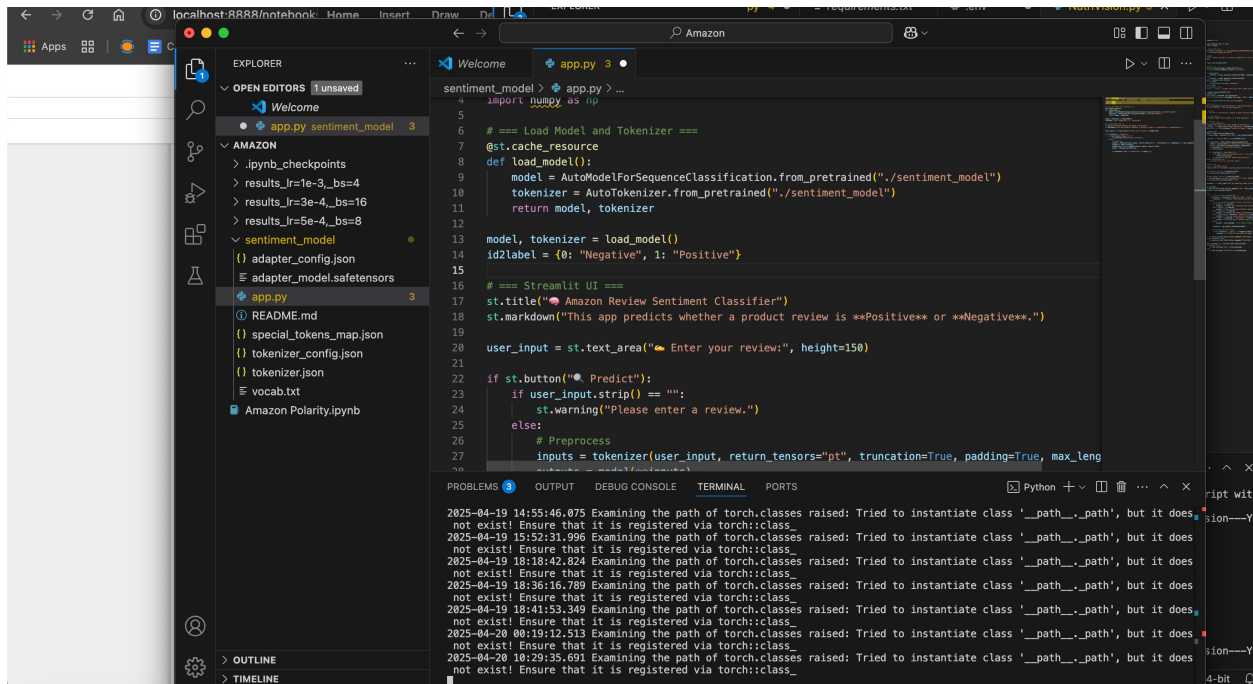
```
👉 It was just okay. Not great, not terrible.
👉 Negative

👉 Worst purchase I've ever made.
👉 Negative

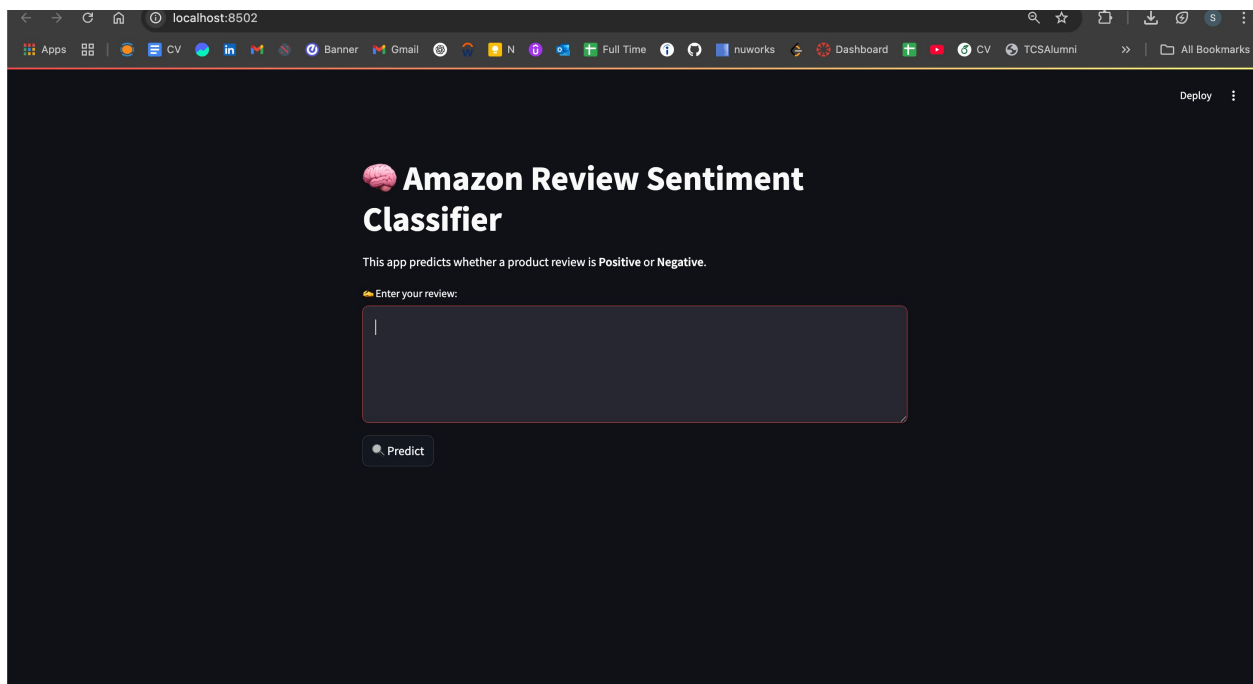
👉 Absolutely loved it! Would buy again.
👉 Positive
```

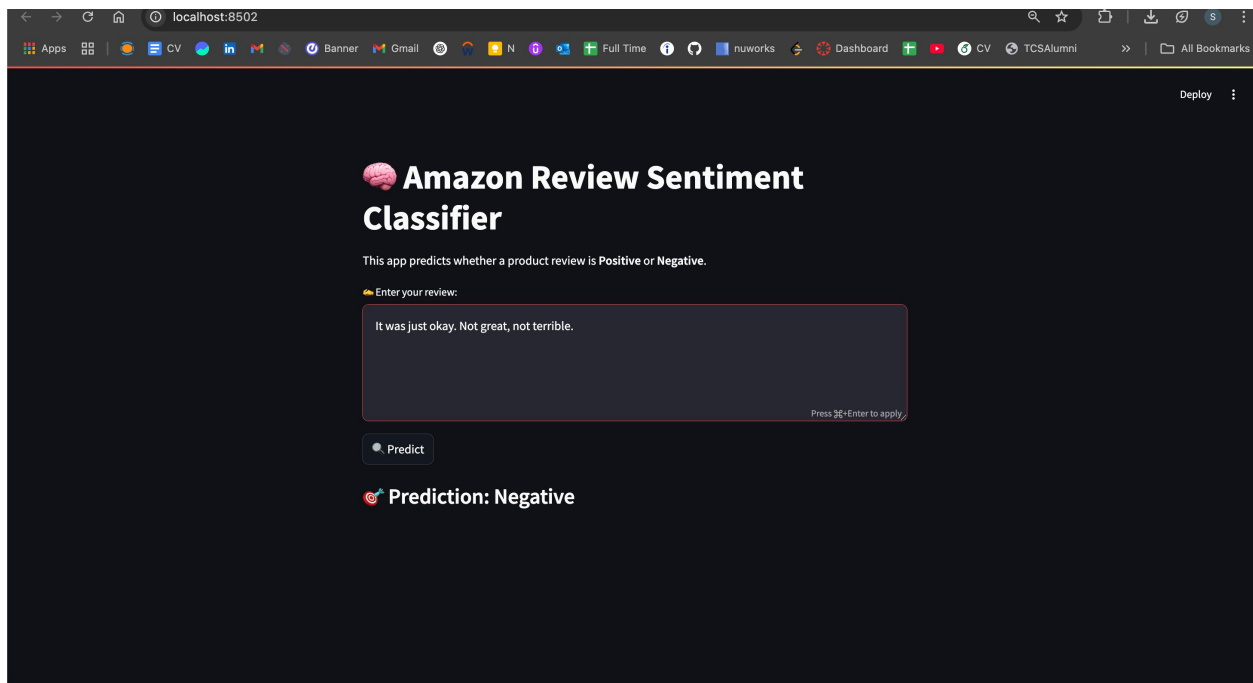
This allows for both real-time and batch predictions.

Streamlit Application



```
4 import numpy as np
5
6 # === Load Model and Tokenizer ===
7 @st.cache_resource
8 def load_model():
9     model = AutoModelForSequenceClassification.from_pretrained("./sentiment_model")
10     tokenizer = AutoTokenizer.from_pretrained("./sentiment_model")
11     return model, tokenizer
12
13 model, tokenizer = load_model()
14 id2label = {0: "Negative", 1: "Positive"}
15
16 # === Streamlit UI ===
17 st.title("🧠 Amazon Review Sentiment Classifier")
18 st.markdown("This app predicts whether a product review is **Positive** or **Negative**.")
19
20 user_input = st.text_area("📝 Enter your review:", height=150)
21
22 if st.button("🔍 Predict"):
23     if user_input.strip() == "":
24         st.warning("Please enter a review.")
25     else:
26         # Preprocess
27         inputs = tokenizer(user_input, return_tensors="pt", truncation=True, padding=True, max_length=128)
```





Step 9: Limitations and Future Improvements

Limitations

- Dataset size limited to 1k examples for performance
- LoRA only applied to q_lin , not v_lin or feedforward layers
- No exploration of neutral/multi-label sentiment

Future Work

- Use full Amazon Polarity dataset (3.6M samples)
- Integrate sarcasm detection and multi-label classification
- Fine-tune larger models like RoBERTa or DeBERTa
- Expand LoRA application to MLP layers
- Deploy using a simple web app (Streamlit or Gradio)

Step 10: Conclusion

In this project, we successfully fine-tuned a pre-trained DistilBERT model on the Amazon Polarity sentiment classification task using LoRA (Low-Rank Adaptation) through the Hugging Face PEFT library. By freezing the majority of model parameters and training only a lightweight set of adapter layers, we achieved **90.2% accuracy** on the test set — demonstrating that efficient fine-tuning methods can yield high performance, even with small labeled datasets and limited compute resources.

The workflow incorporated all essential steps of an NLP pipeline: dataset loading and preprocessing, tokenization, model adaptation, hyperparameter tuning, evaluation, error analysis, and inference design. LoRA significantly reduced the number of trainable parameters (to <1%) without sacrificing accuracy, while PEFT simplified the integration process. Through robust evaluation (confusion matrix, F1-score, classification report), we confirmed strong generalization across both positive and negative sentiment classes. Furthermore, the inference pipeline enables real-time prediction and batch processing for practical deployment.

This project highlights how modern transfer learning techniques like LoRA and tools like PEFT are reshaping NLP, enabling scalable, reusable, and computationally efficient model customization. Looking ahead, this approach can be extended to other domains, multi-class classification, and low-resource languages, while exploring techniques like prompt tuning, adapter fusion, or training on larger datasets to push performance even further.

Lessons Learned

Working on this project really opened my eyes to how powerful and efficient modern AI techniques can be when used the right way. I always thought fine-tuning large language models would require massive datasets and high-end hardware, but using LoRA and PEFT completely changed that perspective. I learned that it's not always about throwing more resources at a problem — sometimes, it's about being smart with the tools and methods you choose. I also realized the importance of understanding your data before jumping into training, as small things like token length or class balance can make a huge difference. Lastly, seeing how a model went from random guesses to actually understanding sentiment was a satisfying reminder of how effective even lightweight models like DistilBERT can be when fine-tuned properly. Overall, this project taught me how to approach AI tasks more efficiently, think critically about optimization, and appreciate the balance between performance and resource management.

Step 10: References

- Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). *DistilBERT: smaller, faster, cheaper and lighter*. [arXiv:1910.01108](https://arxiv.org/abs/1910.01108)
- Hu, E. J., et al. (2021). *LoRA: Low-Rank Adaptation of Large Language Models*. [arXiv:2106.09685](https://arxiv.org/abs/2106.09685)

- Hugging Face PEFT: <https://github.com/huggingface/peft>
- Hugging Face Transformers: <https://huggingface.co/transformers>
- Amazon Polarity Dataset: https://huggingface.co/datasets/amazon_polarity
- Vaswani, A., et al. (2017). *Attention is All You Need*. [arXiv:1706.03762](https://arxiv.org/abs/1706.03762)

License

Copyright 2025 Selvin Tuscano

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.