Homework 5

Problem 1: (RLHF) Direct Preference Optimization (DPO) for YouTube Title Generation (25 points)

1.1 Overview

In this assignment, you will implement Direct Preference Optimization (DPO) to fine-tune a Large Language Model for generating engaging YouTube titles. DPO is a method for training language models from preference data without the need for an explicit reward model or reinforcement learning. You will use the Qwen3-14B model and fine-tune it using preference pairs from a YouTube titles dataset, comparing chosen vs. rejected title examples. Notebook

1.2 Learning Objectives

By completing this assignment, you will:

- Understand the DPO framework and its advantages over RLHF
- Implement DPO fine-tuning using the TRL (Transformers Reinforcement Learning) library
- Work with preference datasets containing chosen/rejected pairs
- Use efficient fine-tuning techniques with LoRA (Low-Rank Adaptation)
- Evaluate model performance before and after DPO training
- Apply modern optimization libraries like Unsloth for accelerated training

1.3 Required Libraries and Setup

- Core dependencies: `transformers`, `datasets`, `trl`, `torch`
- Efficient training: `unsloth`, `peft`, `bitsandbytes`
- Utilities: `accelerate`, `pandas`
- GPU requirements: CUDA-compatible GPU with sufficient VRAM

1.4 Tasks and Scoring

1.4.1 Environment Setup and Model Loading (8 points)

1 Package Installation and Configuration (3 points)

- Install TRL library and its dependencies
- Set up Unsloth for accelerated training
- Configure proper GPU environment and memory optimization
- Install and configure bitsandbytes for quantization support

2 Dataset Loading and Exploration (3 points)

- Load the "EliasHossain/youtube-titles-dpo" dataset using Hugging Face datasets

- Examine the dataset structure including prompt, chosen, and rejected columns
- Understand the preference data format for DPO training
- Analyze sample data to understand the task and data quality

3 Model and Tokenizer Setup (2 points)

- Load the Qwen3-14B model using Unsloth's FastLanguageModel
- Configure 4-bit quantization for memory efficiency
- Set up the tokenizer with appropriate padding tokens
- Verify model loading and initial configuration

1.4.2 LoRA Configuration and Base Model Testing (7 points)

1 LoRA Adapter Configuration (4 points)

- Configure LoRA parameters using FastLanguageModel.get_peft_model
- Set appropriate rank (r=32), alpha, and target modules
- Configure LoRA for key attention and MLP layers (q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj)
- Verify the reduction in trainable parameters compared to full fine-tuning
- Document the memory and computational savings achieved

2 Base Model Evaluation (3 points)

- Create a proper chat prompt formatting function
- Test the base model's performance on YouTube title generation
- Generate sample outputs using the validation dataset prompts
- Document the baseline performance and identify areas for improvement

1.4.3 DPO Training Implementation (10 points)

1 DPO Training Configuration (4 points)

- Configure DPOConfig with appropriate training parameters
- Set learning rate, batch size, and number of epochs
- Configure evaluation strategy and model saving parameters
- Set up logging and monitoring for training progress

2 DPO Trainer Setup and Execution (6 points)

- Initialize DPOTrainer with model, tokenizer, and training arguments
- Configure the trainer with training and evaluation datasets
- Execute DPO training with proper error handling and monitoring
- Track training metrics including:
  - Training loss progression
  - Rewards for chosen vs rejected responses
  - Accuracy metrics and margins
  - Validation loss and other evaluation metrics
- Save the trained model checkpoint

1.5 Model Evaluation and Comparison

After completing the training, your notebook should demonstrate:

1. Qualitative comparison between base and DPO fine-tuned models
2. Analysis of generated YouTube titles for engagement and quality
3. Discussion of the DPO training process and observed improvements
4. Documentation of training metrics and their interpretation

1.6 Deliverables

Submit a complete Jupyter notebook containing:

1. All implementation code with comprehensive comments
2. Successful DPO training with documented progress
3. Before/after model comparison with sample generations
4. Analysis of training metrics and model performance
5. Discussion of DPO methodology and its effectiveness
6. Clean, reproducible code with proper error handling

Note: This assignment focuses on Direct Preference Optimization, which is more efficient than traditional RLHF approaches. DPO directly optimizes the policy using preference data without requiring a separate reward model or reinforcement learning loop. Pay attention to the preference data structure and how DPO uses the chosen/rejected pairs to improve model alignment. The use of Unsloth and LoRA makes this training feasible on consumer hardware while maintaining high quality results.

Problem 2: Implementing ROUGE-L Score for LLM Summarization Evaluation (25 points)

Background

The ROUGE-L (Recall-Oriented Understudy for Gisting Evaluation with Longest Common Subsequence) score, introduced by Lin [2004], is a critical metric in the field of text summarization evaluation. Unlike traditional ROUGE metrics that consider n-gram overlaps, ROUGE-L measures the quality of machine-generated summaries by computing the longest common subsequence (LCS) between a generated summary and one or more reference summaries. This approach has the advantage of automatically identifying longest in-sequence matches and capturing word order, making it particularly effective for evaluating the fluency and coherence of generated text.

ROUGE-L-Sum, an extension of the base ROUGE-L metric, addresses the limitations of basic ROUGE-L when dealing with multi-sentence summaries. While ROUGE-L computes a single LCS over the entire text, ROUGE-L-Sum calculates separate LCS scores for each reference sentence and combines them, providing a more nuanced evaluation for longer texts Lin [2004]. This method has proven particularly effective for evaluating abstractive summarization models, such as those using pointer-generator networks See et al. [2017].

Homework 5

Provided Notebook

This assignment will guide you through the implementation and practical application of these important metrics. You will work with the CNN/DailyMail dataset, which has been extensively used in developing and evaluating state-of-the-art summarization models See et al. [2017].

Learning Objectives

Through this assignment, students will develop a comprehensive understanding of text summarization evaluation by:

- Implementing and analyzing the ROUGE-L scoring metric from first principles
- Working with real-world summarization data from the CNN/DailyMail dataset
- Developing robust text preprocessing techniques
- Evaluating and comparing machine-generated summaries using multiple metrics
- (Optional) Gaining hands-on experience with LLM APIs

Detailed Requirements

Part 1: Data Preparation and Preprocessing (7 points)

Students must implement a robust text preprocessing pipeline that includes:

Dataset Integration (3 points) Implement functionality to load and process the CNN/DailyMail dataset using the Hugging Face datasets library. The implementation should handle data extraction efficiently and include appropriate error handling for edge cases.

Text Preprocessing Pipeline (4 points) Develop a comprehensive text preprocessing system that includes:

- Basic text cleaning and special character handling (1 point)
- Robust handling of contractions and whitespace normalization (1 point)
- Implementation of NLTK tokenization with appropriate fallback mechanisms (1 point)
- Case normalization and word stemming using PorterStemmer (0.5 points)
- Comprehensive error handling for all preprocessing steps (0.5 points)

Part 2: ROUGE Score Implementation (18 points)

Basic ROUGE-L Implementation (7 points)

- Design and implement an efficient LCS table computation algorithm (3.5 points)
- Create a complete ROUGE-L score calculation system with precision, recall, and F1 measures (3.5 points)

Homework 5

ROUGE-LSum Implementation (6 points)

- Develop accurate sentence boundary detection and splitting functionality (2 points)
- Implement the complete ROUGE-LSum calculation algorithm (4 points)

Testing and Validation (5 points) Create a comprehensive testing framework that includes:

- Integration with the official rouge-score library for validation (2 points)
- Comparison of custom implementation results with official scores (1.5 points)
- Analysis of score differences with documentation (¡ 5% threshold) (1.5 points)

Bonus Part: Summary Generation with LLM APIs (+5 Extra Points)

Note: Due to API constraints, this section is optional. You may use the pre-generated summaries provided in the notebook repository, or implement your own generation pipeline for bonus credit.

Implement a system for generating summaries using an LLM provider of your choice (e.g., OpenAI, Hugging Face Inference API, Groq, Google Gemini, or local Ollama instances):

API Integration (3 Bonus Points)

- Secure setup of API authentication (keys must not be hardcoded)
- Implementation of robust API calling functions
- Demonstration of successful generation on a subset of the dataset

Response Processing (2 Bonus Points) Develop systematic handling of API responses, including error cases and parsing the returned JSON/text into a format suitable for your ROUGE evaluation pipeline.

Submission Requirements

Submit a well-documented Jupyter notebook (.ipynb) that includes:

- Comprehensive implementation of all required functions with detailed documentation
- Example runs demonstrating functionality with sample data
- Thorough analysis of results, including comparison with official ROUGE scores
- Clear documentation of any external resources or references used

Technical Requirements

The implementation must use the following libraries:

```
datasets>=3.1.0
numpy>=1.17
nltk>=3.6.3
num2words
rouge-score
# Optional (for Bonus): openai, google-generativeai, or requests
```

Homework 5

Problem 3: Comparative Analysis of STT Vendors (25 points)

Vendor Assignment

Instruction: Calculate your **Student Card ID modulo 18** (i.e., the remainder when your ID is divided by 18). Match the result to the table below to find your assigned vendor.

(Example: If your ID ends in 100, $100 \div 18 = 5$ *with a remainder of **10**. You are assigned Google Cloud.)*

Remainder	Assigned Vendor	Remainder	Assigned Vendor
0	Amazon Transcribe	9	Gladia
1	AssemblyAI	10	Google Cloud STT
2	Azure AI Speech	11	Groq (Whisper implementation)
3	Azure OpenAI (Whisper)	12	Mistral AI (STT/Integration)
4	Baseten (Whisper host)	13	OpenAI (API)
5	Cartesia	14	Sarvam
6	Clova (Naver)	15	Soniox
7	Deepgram	16	Speechmatics
8	fal (Whisper host)	17	Spitch

Note for Inference Providers (Groq, Baseten, fal): If your vendor does not have a proprietary model but hosts open-source models (like Whisper), focus your research on their **inference engine optimizations**, latency claims, and hardware advantages (e.g., LPUs vs GPUs).

Research Questions

Instructions: Answer the questions below using **only 1–3 simple sentences**, followed immediately by the relevant reference (URL, paper title, or patent number).

Part 1: Vendor Profile (2 Points)

1a. Identify the Vendor & Product: State the specific vendor and the exact model/product name you are researching.

1b. Target Use Case & Pricing: What is the primary use case this vendor markets towards? Briefly explain the pricing unit (e.g., "$0.0043 per minute").

Part 2: WER Benchmarking Methodology (10 Base Points + 5 Bonus)

Instructions: Choose **ANY 2** of the following questions to answer for full base credit (10 Points). Answer the remaining 2 questions for **+5 Bonus Points**.

2a. Benchmarking Claim: How does the vendor claim to test their own accuracy? (e.g., "They compare their model against OpenAI Whisper on the Earnings-22 dataset").

2b. Datasets Used: List the specific datasets they cite in their technical blog or papers. (e.g., "LibriSpeech, Common Voice, and proprietary call logs").

2c. Reported WER & Conditions: What is the lowest WER they advertise, and what were the conditions? (e.g., "11.2% WER on noisy audio with background music").

2d. Normalization Rules: How do they handle text formatting during scoring? (e.g., "They lowercase all text and remove punctuation before calculating WER").

Part 3: The "Secret Sauce" (10 Base Points + 5 Bonus)

Instructions: Answer **3a and 3b** for full base credit (10 Points). Answer **3c** for **+5 Bonus Points**.

3a. Architecture/Technique (5 Points): What specific technology gives them an advantage? (e.g., "They use a 'End-to-End Deep Learning' architecture that skips the traditional phoneme alignment step" or "Groq uses LPU inference chips to speed up Whisper").

3b. Academic Publications (5 Points): Search for and cite **1–3 relevant academic papers** where this company's researchers are primary authors. (Format: *Title, Author, Year*). *If no direct papers exist, find the closest technical whitepaper released by their engineering team.*

3c. Patents (Bonus +5 Points): Search for and cite **1–3 relevant patents** granted to this company in the last 5 years. (Format: *Patent Title, Patent Number*). *Look specifically for patents related to latency reduction, acoustic modeling, or streaming architecture.*

Part 4: Appendix (3 Points)

4a. References: Paste a clean, scannable list of all URLs, paper DOIs, and Patent links used to answer the questions above.

Problem 4: Model Context Protocol (MCP) Server Development (25 points)

4.1 Overview

In this assignment, you will build a Model Context Protocol (MCP) server that enables AI assistants like Claude to interact with local data files. You will implement both server and client components, integrate with Claude Desktop, and demonstrate the system's capabilities through comprehensive testing. This problem introduces you to modern AI tooling architectures and the emerging MCP standard for AI-tool integration. File

4.2 Learning Objectives

By completing this assignment, you will:

- Understand the Model Context Protocol and its role in AI system integration
- Implement server-side tools that AI assistants can invoke
- Build client applications for testing MCP servers
- Configure and integrate with Claude Desktop for natural language interaction
- Analyze the performance and capabilities of AI-tool integrations

4.3 Required Libraries and Setup

- Core dependencies: `mcp[cli]`, `pandas`, `pyarrow`
- Testing dependencies: `asyncio`, `pathlib`
- Claude Desktop application (for integration testing)
- Python 3.8+ with virtual environment support

4.4 Tasks and Scoring

4.4.1 MCP Server Implementation (10 points)

1 Project Setup and Environment Configuration

- Create a proper project structure with virtual environment
- Install required dependencies using `requirements.txt`
- Implement environment activation scripts for cross-platform compatibility
- Document the setup process and verify installation

2 Core MCP Server Development (5 points)

- Implement the main MCP server using `FastMCP` framework
- Create utility functions for reading CSV and Parquet files
- Implement error handling for file operations and data validation
- Design the server to automatically create sample data on first run

3 MCP Tools Implementation (5 points)

- Implement `list_data_files` tool with proper documentation
- Create `summarize_csv_file` and `summarize_parquet_file` tools
- Develop `analyze_csv_data` tool with multiple analysis operations (describe, head, info, columns)
- Build `create_sample_data` tool for generating synthetic datasets
- Ensure all tools follow MCP specification with proper type hints and docstrings

4.4.2 MCP Client Development and Testing (10 points)

1 MCP Client Implementation (5 points)

- Develop an asynchronous MCP client using `ClientSession` and `stdio_client`
- Implement connection management with proper error handling and cleanup
- Create methods for listing tools, calling tools, and retrieving resources
- Build both demo mode and interactive mode functionality

2 Comprehensive Testing Suite (5 points)

- Design and execute automated tests for all MCP tools
- Implement interactive testing mode with command parsing
- Validate tool responses and error handling scenarios
- Create test cases for edge cases (missing files, invalid parameters, etc.)
- Document test results and demonstrate client-server communication

4.4.3 Claude Desktop Integration and Analysis (5 points)

1 Claude Desktop Configuration (1 point)

- Create launcher scripts for cross-platform Claude Desktop integration
- Configure `claude_desktop_config.json` with proper absolute paths
- Implement proper virtual environment activation in launcher scripts
- Verify successful integration with Claude Desktop (tool icon appears)

2 Natural Language Interaction Testing (2 points)

- Design and execute comprehensive test prompts for Claude Desktop
- Test basic functionality: file listing, summarization, data analysis
- Evaluate advanced scenarios: multi-step analysis, data creation, error handling
- Document the quality and accuracy of Claude's responses to data analysis requests

3 Performance Analysis and Comparison (2 points)

- Compare response times between direct client calls and Claude Desktop integration
- Analyze the effectiveness of natural language vs. programmatic tool invocation
- Evaluate the user experience and practical applications of the MCP integration
- Discuss limitations and potential improvements for the system

4.5 Deliverables

Submit the following files in your Jupyter notebook:

1. **main.py**: Complete MCP server implementation with all required tools
2. **client.py**: Full-featured MCP client with demo and interactive modes
3. **requirements.txt**: All necessary Python dependencies
4. **run_mcp_server.sh**: Cross-platform launcher script for Claude Desktop
5. **claude_desktop_config.json**: Example configuration file for Claude Desktop
6. **Testing documentation**: Comprehensive test results showing:
   - Client-server communication logs
   - Tool functionality verification
   - Claude Desktop integration screenshots/outputs

Note: This assignment requires careful attention to environment setup and configuration. Start early and test thoroughly on your target platform. The MCP ecosystem is rapidly evolving, so refer to the latest documentation for any updates to the protocol or SDK.

References

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.