Practical 2025: Aligning AI-Generated Teacher Evaluations with Human Judgments

CS XXX: Advanced Machine Learning

Due 11:59 PM on Wednesday, May 31st, 2025

This assignment must be completed in groups of 2 to 3 students. You can seek partners via Ed, and course staff will release a team-finding spreadsheet. This document describes the fine-tuning project, including implementation details and submission logistics. In this same directory, you will find a file called `practical-template.tex`. Use it as a LaTeX template for your writeup, and be sure to read it for grading details.

Harvard College Honor Code: Members of the Harvard College community commit themselves to producing academic work of integrity – that is, work that adheres to the scholarly and intellectual standards of accurate attribution of sources, appropriate collection and use of data, and transparent acknowledgement of the contribution of others to their ideas, discoveries, interpretations, and conclusions. As such, you may not share your code, or have it viewed by any other student outside your group.

1 Implementation Guidelines

Important: Throughout this project, you will be working with several external libraries and APIs. Always search for the most current documentation and implementation examples before coding. Libraries like Hugging Face's Transformers, PEFT, TRL, and various LLM APIs (OpenAI, Anthropic, Google) receive frequent updates. The correct implementation methods may change over time, so consulting official documentation is essential.

When implementing any aspect of this project:

- Search for up-to-date tutorials, examples, and best practices for each library or API
- Review GitHub repositories with similar implementations to understand proper usage
- Pay special attention to data structures expected by these libraries
- Use JSON formatting for LLM outputs that need to be parsed
- Always constrain LLM generation using proper API syntax when expecting structured outputs
- Write well-documented, organized code with clear variable names and structure
- Include comments explaining complex operations and why certain decisions were made

2 Logistics

2.1 Implementation Details

Raw audio data and a starter framework have been included in the repo. You will be required to submit all of your implementation code to Gradescope. This assignment can be completed on either your own computer or Google Colab, if you are having issues locally.

2.2 Submission Details

The main deliverable of this practical is a 3-4 page typewritten document in PDF format to Gradescope. The document must follow the `practical-template.tex` file in this directory and follow all instructions outlined in Section 3. All relevant text—including your discussions of why you tried various things and why the results came out the way they did—must be included in the maximum of 4 pages. If you need more space than 4 pages for tables, plots, or figures, that is okay.

You should also submit your code as either a `.py` or `.ipynb` file on Gradescope. Make sure that the code is neatly organized to reflect which part is being answered. Before submitting, rerun the entire notebook and display all relevant results.

2.3 Grading

Our grading focus will be on your ability to clearly and concisely describe what you did, present the results, and most importantly discuss how and why different choices affected performance. Try to achieve at least 0.7 Cohen's Kappa on the validation set at the end, although you can still earn full credit without reaching this mark (provided that your methodology and writeup are sound).

The project will be graded holistically, with each major component evaluated on a check, check-minus, and minus basis. A check is provided for successfully and thoughtfully completing the section. A check-minus is provided for completing parts of the section and providing little interpretation. Lastly, a minus is provided for providing little to no work. Section 5 is an optional extra credit opportunity.

See `practical-template.tex` for our desired submission format and more tips for what a full-credit submission looks like. All team members will receive the same practical grade.

3 Problem Background

For this practical, you will fine-tune a large language model (LLM) to evaluate teacher performance based on classroom audio transcripts. Specifically, you will create a model that can analyze transcript text and produce evaluative feedback aligned with the World Bank's Teach framework.

The project is based on the SwiftScore-World Bank collaboration, which aims to develop a scalable teacher evaluation system that aligns closely with human expert judgments. The value of such a system lies in scalability and accessibility—AI can process far more classroom transcripts than human evaluators, providing timely feedback to inform teacher professional development.

Your model will focus on analyzing classroom audio transcripts (rather than videos) to emphasize accessibility for schools worldwide, recognizing that video recording infrastructure is often costly and impractical in many educational settings. Additionally, the audio-only approach helps preserve privacy by reducing concerns related to classroom recordings and sensitive interactions.

The primary goal is determining AI alignment with human judgment in education. Any AI-generated teacher evaluation must closely match what expert human evaluators would conclude based on the Teach framework. By aligning the model's evaluations with skilled human observers,

we ensure the technology augments human-led evaluation processes rather than conflicting with them.

4 Your Task and Deliverables

Your task is to implement a complete pipeline for aligning AI-generated teacher evaluations with human judgments. This includes data processing, model fine-tuning, and evaluation. Below, we outline the key components of this project.

4.1 Evaluation Metrics

In this practical, you should focus on optimizing for inter-rater reliability between the AI model and human evaluators:

- **Cohen's Kappa:** Measures agreement between AI and human evaluators across Teach framework dimensions, accounting for agreement by chance.
- **Percentage of Exact Agreement:** The percentage of cases where AI and human scores match exactly.
- **Mean Absolute Error (MAE):** Average absolute difference between AI and human scores.
- **Intraclass Correlation Coefficient (ICC):** For measuring consistency between human and AI ratings (if applicable to your implementation).

For each model you train, calculate these metrics on both the train and validation datasets and include these results in your write-up.

Required Visualizations: Your evaluation section should include the following visualizations:

- Confusion matrices comparing human vs. AI scores for each Teach framework dimension
- Bar charts showing Cohen's Kappa values across different model variants (real data, synthetic data, mixed)
- Line graphs tracking evaluation metrics during training (to identify potential overfitting)
- Violin plots comparing score distributions between human evaluators and AI model outputs
- Heatmaps highlighting areas of highest disagreement between human and AI evaluations

4.2 Data Processing and Exploration

The first step in your project is to process the raw audio data and create structured training datasets:

1. **Data Transcription:** You need to transcribe the raw audio recordings to text. Use a speech-to-text model like Open Whisper v3 (via Groq or another inference method). Ensure you include timestamps in the transcripts.
2. **Exploratory Data Analysis (EDA):**
    - Understand how you will adapt the evaluation framework for creating the evaluation data

- Analyze the distribution of scores across different Teach framework dimensions
- Identify patterns in transcript characteristics that correlate with evaluation scores

3. **Data Cleaning and Preparation:**
   - Create appropriate train-validation-test splits, based on the number of evaluations
   - Split the data from whole evaluations into individual Teach components (this is likely an $\mathbb{R}^1 \to \mathbb{R}^4$ mapping)
   - Format the data into proper structure with columns such as:
     - Audio Transcript (with timestamps)
     - Domain
     - Score
     - Summary
   - Ensure all data is well-organized and easily accessible for later stages
   - Create clearly separated datasets for each training scenario (real, synthetic, mixed)
4. **Dataset Format:** Convert your processed data into a format suitable for fine-tuning. All data must be formatted in JSONL files or a Hugging Face dataset with the following structure:
   - Input: Classroom transcript text
   - Output: Formatted evaluation with scores and feedback in JSON format

Important: All model outputs (both in training data and evaluation) must be structured in JSON format as follows:

```
{
  "score": int,  // Integer score as per Teach framework
  "summary": "string"  // Text summary/feedback
}
```

This standardized JSON format ensures consistency in data processing and evaluation. When creating your training dataset, ensure that human-labeled evaluations are also converted to this format. This approach facilitates easier parsing during evaluation and matches best practices for structured LLM outputs in production environments.

4.3 Synthetic Data Generation

Following the data processing, you should implement synthetic data generation to augment your training set:

1. **Teacher Model Knowledge:** Before generating synthetic data, you must provide context to the teacher model about the Teach framework:
   - Feed the model relevant text/PDF documents that explain the Teach framework
   - Include detailed information about how to properly score classroom observations
   - Provide examples of high-quality evaluations following the framework
   - This context will help the model generate synthetic data that properly follows the scoring criteria, effectively acting like a human evaluator

2. **Generation Approach:** Use a high-capability LLM (e.g., Gemini 2.0 Flash) as your "teacher" model to generate synthetic transcript-evaluation pairs in JSON format. Create a variety of classroom transcript scenarios to ensure coverage of different teaching situations.

3. **Quality Control:** Implement checks to ensure the synthetic evaluations are realistic and aligned with expected human outputs:

   - Verify that all generated outputs follow the required JSON format
   - Check that scores fall within valid ranges for each dimension
   - Ensure summaries provide appropriate, constructive feedback
   - Manually review a sample to validate overall quality

4. **Dataset Creation:** Your implementation must create and maintain three distinct datasets:

   - **Real Dataset:** Contains only human-labeled evaluations
   - **Synthetic Dataset:** Contains only synthetic evaluations (start with 100 examples)
   - **Mixed Dataset:** Contains a balanced combination of real and synthetic data

5. **Flexible Implementation:** Design your code to be flexible and efficient:

   - Store generated synthetic data so it can be reused
   - Allow easy scaling of synthetic data generation (e.g., from 100 to 500 examples)
   - Include parameters to control the ratio of real to synthetic data in the mixed dataset
   - Implement caching to avoid regenerating data unnecessarily

Note: Start with a small synthetic dataset (around 100 examples) to verify your pipeline works correctly. Once you've successfully trained an initial model, you can scale up the synthetic data generation as needed to improve performance.

4.4 Model Fine-Tuning

For the main assignment, you will fine-tune Google's Gemma 3 27B model using QLoRA:

1. **Model Setup:**

   - Load the Gemma 3 27B model with appropriate quantization configuration
   - Implement QLoRA (Quantized Low-Rank Adaptation) to efficiently fine-tune the model
   - Configure the training pipeline using the TRL library's SFTTrainer

2. **Training Configuration:**

   - Implement supervised fine-tuning with an appropriate learning rate, batch size, and training duration
   - Use gradient checkpointing, mixed precision, and other efficiency techniques
   - Configure QLoRA settings (rank, alpha, target modules)

3. **Experimentation:**

   - Train multiple model variants:
     - Model variant trained on real data only
     - Model variant trained on synthetic data only
     - Model variant trained on mixed real+synthetic data

4.5 Model Evaluation

After training your models, you should implement comprehensive evaluation:

1. **Inter-rater Reliability:** Implement the reliability metrics described in section 3.1.

2. **LLM-as-Judge Evaluation:** Implement an LLM-as-a-Judge framework for preference evaluation, substituting for human preferences. This involves:

   - **Judge Selection:** Use a strong LLM (e.g., GPT-4, Claude, or Gemini) as your judge model. This model should be different from the one you're evaluating to avoid preference leakage and bias.
   - **Prompt Engineering:** Create well-structured prompts that:
     - Clearly define evaluation criteria based on the Teach framework
     - Include Chain-of-Thought (CoT) reasoning to improve judgment quality
     - Break down evaluation into specific aspects (e.g., completeness, relevance, accuracy)
     - Provide few-shot examples with reasoning for different quality levels
   - **Evaluation Formats:** Implement at least two of these evaluation types:
     - **Pairwise Comparison:** Have the judge choose between two model outputs (e.g., your model vs. reference human evaluation)
     - **Direct Scoring:** Have the judge assign scores (1-5) to model outputs based on specified criteria
     - **Batch Ranking:** Have the judge rank multiple outputs for the same transcript
   - **Validation:** Validate your LLM judge by calculating its agreement with human evaluations on a subset of data where both are available.

   Implementation Resources: You should research and utilize existing implementations of LLM-as-a-Judge. Useful resources include:

   - Hugging Face's evaluation libraries (especially their LLM judge templates)
   - GitHub repositories like JudgeLM and DeepEval
   - Academic papers such as "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena"

3. **Comparative Analysis:** Compare your fine-tuned model(s) with:

   - Base model (Gemma 3 27B without fine-tuning)
   - GPT-4o (through the OpenAI API)
   - Gemini 2.0 Flash
   - Any other models that you train

4. **Error Analysis:** Perform detailed error analysis focusing on:

   - Discrepancies between human and AI scores/summaries
   - Patterns in cases where AI-human disagreement is highest
   - Categorizing the types of classroom interactions or teaching behaviors where alignment is weakest
   - Feature extraction to identify what aspects of the transcripts lead to different evaluations

5. **Key Comparisons:** Analyze and visualize these important comparisons:
    - Whether the fine-tuned model (distilled from the teacher model) outperforms the teacher model on test data
    - Performance differences between models trained on real data only vs. synthetic data vs. mixed data
    - How different fine-tuning methods affect evaluation quality

5 Final Write-up and Reflections

To wrap up your exploratory research into applying parameter-efficient fine-tuning to teacher evaluation, document and summarize the steps you took in the ML research pipeline, and reflect on the choices that you made.

For each of the components below, include a paragraph (2-4 sentence) explanation and reflection of what you did for each step. Think through any trade-offs, domain-specific input, real-world considerations, and ethical decisions that you made along the way, in addition to any considerations made from a purely machine learning perspective.

Key Components:

1. **Data Processing**: How did you transcribe and format the audio-evaluation pairs? What challenges did you encounter in preparing the data for fine-tuning?
2. **Synthetic Data Generation**: How did you approach synthetic data generation? What quality control measures did you implement?
3. **Model Selection**: What considerations went into using Gemma 3 27B? What trade-offs did you make between model size, capability, and computational requirements?
4. **Fine-Tuning Method**: How did your QLoRA implementation perform? What were the key advantages and disadvantages of using parameter-efficient fine-tuning in this context?
5. **Hyperparameter Optimization**: What approach did you take to tuning hyperparameters? Which parameters had the most significant impact on performance?
6. **Evaluation**: How well did your AI evaluations align with human judgments across different dimensions of the Teach framework? Were there particular aspects of teaching that the model struggled to evaluate accurately?
7. **Domain-specific Insights**: Are there any insights about classroom evaluation that you gained through this project? How might educational experts view the use of AI for teacher evaluation?
8. **Deployment Considerations**: What would be required to deploy this model in real educational settings? What privacy, ethical, and practical considerations would need to be addressed?

6 Optional: Extra Credit and Extensions

6.1 Competition: Extra Credit

We will be awarding 5 percentage points of extra credit on this assignment to the 5 teams with the best models. This will be measured by performance on a test set of classroom transcripts, which can be found in the file `test.csv`. Note that you only have the raw audio for these clips—the evaluation scores have not been provided.

For this extra credit opportunity, you should:

1. Train an alternative LLM model (e.g., Llama 3.3, Llama 4, or another state-of-the-art model)
2. Use the validation set as additional training data (this is only allowed for the extra credit component)
3. Make predictions on the test set and submit those predictions to our Kaggle competition (click here for the link)

An example of the submission format can be found in the file `sample_submission.csv`.

Please separate your code for this component of the practical from your code for the main assignment, and ensure that we can replicate your predictions through running your code.

There are a few things you should keep in mind, should you choose to enter the competition. First of all, please do not cheat, e.g., by using external models or datasets. We will be verifying the top 5 teams by running their code files and confirming that the trained models do generate the submitted predictions. Second, you will get to see your model's performance on a subset of the test data through the public leaderboard. However, your ultimate ranking will be determined by your model's performance on the remainder of the test data, and this will be reflected in the private leaderboard, which will be released at 12:00am ET on June 3rd. As such, be careful not to "overfit" to the public leaderboard.

6.2 Direct Preference Optimization (Optional)

For students interested in exploring more advanced fine-tuning approaches, consider implementing Direct Preference Optimization (DPO):

1. **Dataset Creation for DPO:** DPO requires a preference dataset consisting of (prompt, chosen, rejected) triplets:
   - **Prompt:** The classroom transcript
   - **Chosen (Winner):** The human evaluation or high-quality synthetic evaluation
   - **Rejected (Loser):** A flawed evaluation that is clearly inappropriate
2. **Creating Flawed Evaluations:** Generate deliberately imperfect evaluations that:
   - Miss important aspects of the teaching performance
   - Provide overly vague or generic feedback
   - Give scores inconsistent with the transcript content
   - Contain factual errors about what happened in the classroom
   - Use inappropriate or unhelpful language

3. **Preference Dataset Structure:** Each item in your preference dataset should contain:
    - The original transcript (input prompt)
    - The high-quality evaluation (winner)
    - The flawed evaluation (loser)
4. **Implementation:** Use Hugging Face's TRL library which provides a `DPOTrainer`:
    - Start with a model that has already undergone supervised fine-tuning (SFT)
    - Configure DPO parameters like beta (typically 0.1-0.5) to control regularization
    - Balance the training to avoid overfitting to the preference dataset
    - Consider implementing conservative DPO with label smoothing for better stability
5. **Evaluation:** Compare DPO-tuned models against your SFT models to assess whether:
    - DPO improves alignment with human evaluations
    - The model better avoids common evaluation pitfalls
    - The feedback quality improves in subjective assessments

6.3 Contrastive Fine-Tuning (Optional)

Another advanced approach is Contrastive Fine-Tuning, which explicitly teaches the model what not to do:

1. **Approach:** Create contrasting pairs of good vs. bad evaluations for the same transcripts:
    - Generate multiple types of "bad" evaluations with different flaws
    - Pair each with the corresponding "good" evaluation
    - Fine-tune the model to increase probability of good outputs and decrease probability of bad ones
2. **Implementation Methods:**
    - Use a pairwise contrastive loss similar to DPO implementation
    - Create a "negative persona" model to generate flawed evaluations
    - Train with pairs showing both good and bad examples for the same prompt
3. **Analysis:** Study how contrastive tuning affects:
    - The model's ability to avoid common pitfalls in evaluations
    - Overall alignment with human judgments
    - The specificity and helpfulness of feedback

7 Other FAQs

What language should I code in? As you will be submitting your code, you should code in Python.

Can I use {transformers — trl — peft — other ML library}? You can use these tools, but not blindly. You are expected to show a deep understanding of the methods we study in the course, and your writeup will be where you demonstrate this.

What do I submit? You will submit both your write-up on Gradescope and all of your practical code to a supplemental assignment on Gradescope.

Can I have an extension? Yes, your writeup can be turned in late according to the standard homework late day policy. You can use at most two late days on the practical.

Where can I find resources on fine-tuning? Start by consulting Hugging Face's documentation for PEFT, TRL, and BitsAndBytes libraries. Look for existing examples of QLoRA implementation, especially those that involve training on structured outputs like evaluations or ratings.

How should I handle data for fine-tuning? For best results with LLM fine-tuning, your data should be formatted as JSON or JSONL files. Each example should contain both the input (transcript) and the expected output (evaluation in JSON format). This standardized approach ensures that your model learns to produce consistently structured outputs that are easy to parse and evaluate.