ITCS 6101/8101 Homework 6: Implementation (100 points)

March 30, 2026

1 Overview

In this assignment, you are provided with PyTorch code that shows how to employ post-training to improve the performance of a pretrained LLM on two toy tasks, as follows:

1. Finetune the pretrained Qwen3-0.6B-Base model on the task of Question Answering, using Supervised Fine Tuning (SFT):
   - Input: the base model is stored in the global read-only folder: `/projects/class/itcs6101_091/hw06/models/Qwen3-0.6B-Base`
   - Input: the finetuning QA data is stored in your local folder: `hw06/data/sft-qa-dataset.json`
   - Output: the finetuned model will be stored in your local folder: `hw06/models/Qwen3-0.6B-Base-SFT-qa`
2. Finetune the instruction tuned Qwen2.5-0.5B-Instruct model on the task of changing how the model refers to itself, using Direct Preference Optimization (DPO):
   - Input: the base model is stored in the global read-only folder: `/projects/class/itcs6101_091/hw06/models/Qwen2.5-0.5B-Instruct`
   - Input: the finetuning QA data is stored in your local folder: `hw06/data/dpo-dq-dataset.json`
   - Output: the finetuned model will be stored in your local folder: `hw06/models/Qwen2.5-0.5B-Instruct-DPO-dq`

Your task is to adapt the provided code for post-training in order to improve the performance of a a pretrained LLM on two tasks, as follows:

1. **Reading Comprehension using SFT**: Finetune the pretrained Qwen3-0.6B-Base model on this task, as detailed in Section 2:
   - Input: the base model is stored in the global read-only folder: `/projects/class/itcs6101_091/hw06/models/Qwen3-0.6B-Base`
   - Input: reading comprehension data for finetuning and evaluation is stored in your local folder:
     `hw06/data/sft-squad-train-dataset.json`
     `hw06/data/sft-squad-eval-dataset.json`
   - Output: the finetuned model will be stored in your local folder: `hw06/models/Qwen3-0.6B-Base-SFT-squad`
2. **Spanish answers to English questions, using DPO**: Finetune the instruction tuned Qwen2.5-0.5B-Instruct model on this task, as detailed in Section 3:

- Input: the base model is stored in the global read-only folder:
  `/projects/class/itcs6101_091/hw06/models/Qwen2.5-0.5B-Instruct`
- Input: the finetuning and evaluation data is stored in your local folder:
  `hw06/data/dpo-gk-enq2spa-train-dataset.json`
  `hw06/data/dpo-gk-enq2spa-eval-dataset.json`
- Output: the finetuned model will be stored in your local folder:
  `hw06/models/Qwen2.5-0.5B-Instruct-DPO-enq2spa`

The skeleton code and data are linked from the course webpage. Organize your code in folders as shown in Table 1 below.

```
/projects/class/itcs[68]101_091/
  hw06/models/
    Qwen3-0.6B-Base/
    Qwen2.5-0.5B-Instruct/

hw06/
  report.pdf
  requirements.txt
  data/
    sft-qa-dataset.json
    sft-squad-train-dataset.json
    sft-squad-eval-dataset.json
    dpo-dq-dataset.json
    dpo-gk-en-train-dataset.json
    dpo-gk-en-eval-dataset.json
    dpo-gk-enq2spa-train-dataset.json
    dpo-gk-enq2spa-eval-dataset.json
  models/
    Qwen3-0.6B-Base-SFT-qa
    Qwen3-0.6B-Base-SFT-squad
    Qwen2.5-0.5B-Instruct-DPO-dq
    Qwen2.5-0.5B-Instruct-DPO-enq2spa
  sft/
    sft_qa.py + slurm_qa_train.sh + slurm_qa_test.sh
    sft_squad.py + slurm_squad_train.sh + slurm_squad_test.sh
    traces.txt
  dpo/
    dpo_dq.py + slurm_dq_train.sh + slurm_dq_test.sh
    enq2ena_to_enq2spa.py
    dpo_enq2spa.py + slurm_enq2spa_train.sh + slurm_enq2spa_test.sh
    traces.txt
```

Table 1: Folder structure.

1.1 The `transformer` and `trl` libraries

The packages necessary for running the provided code are enumerated in the `requirements.txt` file. While the actual version is not essential (you can install the most recent version available), it is important that you install them in an environment that is activated before running the code. Besides the usual packages, e.g., PyTorch and NumPy, you will work with classes from the following packages:

- `transformers`: This package enables loading and using state-of-the-art pretrained Transformer-based models for inference and training. Documentation is available at https://github.com/huggingface/transformers.
  - The code uses the `AutoConfig`, `AutoTokenizer`, and `AutoModelForCausalLM` classes.
- `trl`: This is the Transformers Reinforcement Learning package, which provides a set of tools to train transformer language models with methods like Supervised Fine-Tuning (SFT), Group Relative Policy Optimization (GRPO), Direct Preference Optimization (DPO), Reward Modeling, and more. Documentation is available at https://huggingface.co/docs/trl/index.
  - For SFT, the code uses the `trl.SFTTrainer` and `trl.SFTConfig` classes.
  - For DPO, the code used the `trl.DPOTrainer` and `trl.DPOConfig` classes.

2 Supervised Fine Tuning (SFT)

The toy SFT dataset `data/sft-qa-dataset.json` was created from conversation logs with a frontier LLM. The code in `sft/sft_qa.py` can be run in one of two modes:

- Training, with the command line `-train`. In this mode, the pretrained LLM is finetuned using the class `trl.SFTTrainer` to teach a pretrained LLM how to respond to questions, i.e., the fact that an LLM needs to provide an answer with the information requested in the question, as opposed to simply producing the most likely tokens that can follow the question text.
- Testing, with the command line `-test`. In this mode, the finetuned model is used in inference mode to answer a set of predefined questions that are hard-coded, namely:
  1. *Give me an 1-sentence introduction of LLMs.*
  2. *Calculate 1+1-1.*
  3. *What's the difference between a thread and a process?*

  Feel free to try the finetuned model on other questions as well.

Due to the high computational demands, it is important that you run the code on the Educational Cluster, or a similar infrastructure that offer GPU resources. Correspondingly, we provide the Slurm script templates for training in `slurm_qa_train.sh` and testing in `slurm_qa_test.sh`.

Your task is to adapt the provided code and Slurm scripts so that you finetune the same pretrained model on the task of Reading Comprehension. For this, you are provided with a training dataset and evaluation dataset, both extracted from the Stanford Question Answering Dataset (SQuAD) dataset. Each example in SQuAD consist of a Wikipedia passage, a question, and the answer to the question. The answer is a segment of text, or span, from the Wikipedia passage that answers the question (answerable), or empty if the answer is not present in the Wikipedia passage (unanswerable). The provided datasets are:

- `sft-squad-train-dataset.json` contains 1,500 answerable and 1,500 unanswerable examples.
- `sft-squad-eval-dataset.json` contains 500 answerable and 500 unanswerable examples.

Your code and scripts should be provided in the green files under the `sft/` folder above. The finetuned model should be saved in the corresponding file indicated in violet under the `models/` folder above. You code should also implement a function for computing the accuracy of the finetuned model on the evaluation dataset.

3 Direct Preference Optimization (DPO)

The toy DPO dataset `data/dpo-dq-dataset.json` was created from short conversations with a Qwen model, where the aim is to finetune an instruction-tuned Qwen model to change how it refers to itself, from "Qwen" to "Deep Qwen". The code in `dpo/dpo_dq.py` can be run in one of two modes:

- Training, with the command line `-train`. In this mode, the pretrained LLM is finetuned using the class `trl.DPOTrainer` to teach an instruction-tuned LLM how to refer to itself.
- Testing, with the command line `-test`. In this mode, the finetuned model is used in inference mode to answer a set of predefined questions that are hard-coded, namely:
  1. *What is your name?*
  2. *Are you ChatGPT?*
  3. *Tell me your name and about your organization.*

  Feel free to try the DPO-tuned model on other questions as well.

Due to the high computational demands, it is important that you run the code on the Educational Cluster, or a similar infrastructure that offer GPU resources. Correspondingly, we provide the Slurm script templates for training in `slurm_dq_train.sh` and testing in `slurm_dq_test.sh`.

Your task is to adapt the provided code and Slurm scripts so that you DPO-tune the same instruction-tuned model on the task of answering in Spanish to any question that is asked in English. For this, you are provided with a training dataset and evaluation dataset, both extracted from the General Knowledge (GK) dataset. Example in GK are question-answer pairs themed on general facts and reasoning. The provided datasets are:

- `dpo-gk-en-train-dataset.json` contains 1,000 QA pairs.
- `dpo-gk-en-eval-dataset.json` contains 3 QA pairs.

However, the GK datasets above only contain the English answer, whereas DPO also needs to have the prefered Spanish answer for the same question. Therefore, you will need to implement the script `dpo/enq2ena_to_enq2spa.py` that takes as input the datasets above in orde rto generate a dataset that, for each English question, specifies not only the English answer (*rejected*, or not preferred), but also its Spanish version (*chosen*, or preferred). To translate the answers from English to Spanish, you can use a local, self-contained Neural Machine Translation model. For example, you can use the MarianMT framework to load and run the `Helsinki-NLP/opus-mt-en-es` model for translating from English to Spanish: https://huggingface.co/docs/transformers/en/model_doc/marian
Alternatively, you can use an LLM through a chat completion API.

Your code then should generate the DPO training and evaluation data in the files:

- `dpo-gk-enq2spa-train-dataset.json` containing 1,000 QA training triples.
- `dpo-gk-enq2spa-eval-dataset.json` containing 3 QA evaluation triples.

Once you are done with creating these DPO datasets, write your code and scripts in the corresponding green files under the `dpo/` folder above. The finetuned model should be saved in the corresponding file indicated in violet under the `models/` folder above. Make sure that in your Slurm script you add a command for activating the environment before running the code.

4 Implementation Details

Both `trl.SFTTrainer` and trl.DPOTrainer are configured through a configuration object. The provided default values are as follows:

- `per_device_train_batch_size = 1` means 1 sample is loaded at a time on GPU.
- `gradient_accumulation_steps = 8` means gradients are accumulated 8 batches before one optimizer step (gradient update).
- `logging_steps = 2` means training metrics are printed every 2 optimizer steps.

Furthermore, each trainer class constructor takes a `formatting_func` KW argument that specifies a function that is to be run on each example in the dataset before it is used for training. You can write your own function, or you can preprocess the dataset directly so that it fits the expected format for the trainer class, as explained in the documentation.

5 Homework Report

When you evaluate the outputs, try to explain any unexpected results or mistakes, by referring to properties of the algorithm or the data. Whenever you formulate a hypothesis, it is important that it is supported empirically. As such, feel free to train and evaluate models that go beyond the setting described above. All the required results and analyses should be

included in an appropriately edited homework report, using proper indentation, section titles, and formatting. If you include formulas, make sure that you use appropriate formatting of equations. The PDF of the report `report.pdf` should be submitted on Canvas, together with the code in the folder above. The assignment will be graded based on **both** the code and the quality of the report.

6 Bonus

Any non-trivial and insightful extra work can be worth bonus points. For example, you can use the toy code provided for GRPO-tuning that uses the `trl.GRPOTrainer` class to fine-tune the base model `Qwen2.5-0.5B-Instruct` for a task of your choosing. You would need to adapt the code provided in the `grpo/` folder as well as provide the necessary datasets for training and evaluation. The task should be interesting and useful enough to warrant using GRPO.

7 Submission

Electronically submit on Canvas a `hw06.zip` file that contains the hw06 folder in which your code, datasets, and scripts, are in the required files, as well as the `report.pdf` and `traces.txt` files. **Do not submit the finetuned models on Canvas**. Remove the `models/` subfolder before zipping the hw06 submission.

On a Linux system, creating the archive can be done using the command:
`> zip -r hw06.zip hw06`.

Please observe the following when handing in homework:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. Verify your code runs correctly when used in the directory structure shown above. We will not debug your code.
4. For your report, it is recommended to use an editor such as Overleaf for Latex, Word, or Jupyter-Notebook that allows editing and proper formatting of equations, plots, and tables with results.
5. Verify that your Canvas submission contains the correct files by downloading and unzipping it after posting on Canvas.