FLEPBot: Your personal instructor for studying STEM

Ahmed Elalamy | 324610 | ahmed.elalamy@epfl.ch
Baptiste Lecoeur | 316223 | baptiste.lecoeur@epfl.ch
Ghita Tagemouati | 330383 | ghita.tagemouati@epfl.ch
AGEPT Team

Abstract

The aim of this project is to create, based on a Large Language Model (LLM), an educational chatbot able to answer to multiple-choice questions in the STEM field, while giving concrete explanations to said answer. STEM questions, in opposition to other types of questions, appears to be quite a challenge for LLMs.

To create this chatbot, we finetuned an Apple OpenELM base model using data from a variety of sources. We've also trained a quantized version to try and optimized storage space. Metrics computed demonstrate the effectiveness of our strategy and also suggest quite promising results for the future development of such educational chatbots.

1 Introduction

LLMs have taken the world by storm. In education specifically, students are using LLMs more than ever in order to help them through tasks such as summarizing, rewriting texts, fact checking and even more. Nonetheless, the effectiveness of these models is not constant across all fields. Specifically, when thinking of STEM (Science, Technology, Engineering and Mathematics), the questions usually asked in this field require deeper analysis and reasoning about the subject than other questions. In fact, knowing that most of the LLMs we see and interact with today have been trained on datasets based on online articles and websites, and also knowing how causal language models work, it seems logical that "factual" questions such as "What is the capital of France?" are easier for the language model to answer compared to "Solve $x^2 + 5x + 4 = 9$ for x", as the former is easily found in the training data while the latter requires more reasoning.

Current methods for answering STEM multiple-choice questions have limitations. Traditional educational tools may provide answers but lack clear explanations, hindering a student's ability to learn from mistakes and solidify their understanding of the necessary reasoning.

In this project, our primary goal is to finetune an LLM for question-answering multiple-choice questions in the STEM field while also giving explanations. To acheive this, we chose **Apple's OpenELM** (Mehta et al., 2024) **with 270M parameters** as a base model which we trained using the Direct Preference Optimization algorithm (DPO). In simple words, this algorithm leverages the concept of human annotated preference pair data so that the model creates more substantive and "human-liked" answers. This was done using pre-existing datasets as well as constructing our data by prompting ChatGPT.

As a secondary goal, we wanted to optimize the storage needed for this model. Indeed, Large Language Models' sizes grow at an exponential pace, and this size increase hinders their accessibility. This is why we wanted to implement Quantization in our model. Briefly, quantization is a technique used to reduce the size of the model by converting internal high-precision floating point data (e.g. weights) into lower precision ones. This helps shrinking the storage needed for the model while arguably not damaging too much the capabilities and performance of the model.

2 Related Work

Iris: An AI-Driven Virtual Tutor For Computer Science Education: Recently, Bassner et al. (2024) from the Technical University of Munich developped an AI Instructor specifically for Computer Science Education. The general idea of providing students with a model to interact with is persistent, but they mainly rely on preexisting chatbot instances (namely, GPT-3.5 Turbo) with some tweaking while our goal is to train a chatbot by finetuning a pre-existing model.

Large Language Models are Zero-Shot Reasoners: As a first step we needed to create our own

preference data based on a set of questions from EPFL courses. Prompting an LLM can be quite intricate, and helping it find the nuances to create different answers is not at all trivial. Kojima et al. (2023) demonstrated that using as simple Chain-of-Thought reasoning as adding "Let's think step by step" in your prompt can significantly enhance the answer's quality, which in turn can help create qualitative preference data.

Direct Preference Optimization: Your Language Model is Secretly a Reward Model: Rafailov et al. (2023) suggest a method for training LLMs, by optimizing it directly for human preferences. This is significant because traditionally, training LLMs with reinforcement learning for specific tasks involves defining a reward function, which can be challenging and complex. Their approach simplifies the process by leveraging human feedback to guide the LLM's behavior without explicitly defining rewards. This work is precious as it can guide us to ensure the chatbot interacts with students in a way that is best for a human student. By directly incorporating human preferences into the training process, we can potentially train our chatbot to be more effective in an educational setting.

Disentangling Length from Quality in Direct Preference Optimization: This paper by Park et al. (2024) addresses the challenge of verbosity bias in Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO). Traditional RLHF often exploits human preferences for longer, more eloquent responses, even when they are not necessarily more helpful. This bias is less explored in DPO, which differs from RLHF by not using a separate reward model or direct reinforcement learning. We then used the DPO loss of this paper to address the difference of length between the single character answer and the rejected explanation.

3 Approach

We can decompose our approach in 3 main steps:

1. Aligning a reference model to the task of answering EPFL-courses questions, using hand-annotated data and Direct Preference Optimisation (DPO). This model is later on referred to as the M2 model.

2. Re-aligning the already aligned model to the even more specific task of answering Multiple Choice Questions (MCQ) without justifications, still using DPO. To do so we use a 2-step approach:

   (a) First, train our model on a smaller dataset in order to make it learn to provide a short answer (such as a single letter).

   (b) Second, train the model further on another dataset in order to make it learn to answer correctly, instead of providing a random single letter.

3. Finally, we want to use quantization to have a smaller model!

3.1 Direct Preference Optimization (DPO)

DPO uses pairs of data to teach the model the expected behavior. We compare the probability of obtaining a specific output y given a prompt input x between a *policy* model that we optimize $\pi_\theta(y|x)$ and a *reference* model $\pi_{ref}(y|x)$, which defines a reward as

$$r(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)} + \beta Z(x)$$

with Z an unknown repartition function.

We finally define our objective function as

$$\mathcal{L}_{DPO}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_c,y_r)\sim D}[\log \sigma(r(y_c|x) - r(y_r|x))] \tag{1}$$

with y_c and y_r the chosen and rejected output: for each pair of data, a human decides the best answer.

However, we wanted to adapt our precedent model so that it addresses one new challenge. While training on our MCQ datasets, we found ourselves training on different string length between the question, the single character chosen answer, the explanation or the rejected answer. Therefore, we took our inspiration from Park et al. (2024) and modified our objective function as follow. We added a new constant α that defines the importance of the length by multiplying the ratio of the chosen and the rejected length by this constant, which we set to 0.1. Our new and improved objective is then

$$\mathcal{L}_{DPO}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_c,y_r)\sim D}[\log \sigma(\beta \log \frac{\pi_\theta(y_c|x)}{\pi_{ref}(y_c|x)} - \beta \log \frac{\pi_\theta(y_r|x)}{\pi_{ref}(y_r|x)} - \alpha(|y_{c;\theta}| - |y_{c;\text{ref}}|))] \tag{2}$$

with $|y_{c;\theta}|$ and $|y_{c;\text{ref}}|$ denoting the lengths of the chosen response for the policy and reference models respectively.

3.2 Quantization

Once our model was trained, we looked more closely into the possibilities for quantization techniques. We quickly found the `bitsandbytes` HuggingFace extension. This is a very effective tool to load and save our models in a smaller format efficiently. After training, we used the extension to load the model in smaller size, precisely 4 bits for each weight. This way the quantizer is able to take the original model and use its algorithm to optimize memory footprint.

The algorithm used is QLoRA (Dettmers et al., 2023). It uses 4-bit quantization to compress the model by taking its parameters, freezing them and adding a relatively small number of trainable parameters. Then, QLoRA backpropagates gradients through the frozen 4-bit quantized pretrained language model into the added parameters.

4 Experiments

- **Data:** As we trained our model using DPO, our training data consists of preference data pairs. We first use the **CS-552 Preference Data**. This data was collected by prompting a GPT-wrapper with questions (open or MCQ) from different courses at EPFL, and obtaining preference pairs as answers. It contains the `prompt` used on the GPT-wrapper, and the `chosen` and `rejected` answers, whose field name explain their preference. We also used a condensed version of the **H4 Stack Exchange Preferences Dataset** (Lambert et al., 2023), which is made of preference pairs based on questions and answers asked on Stack Exchange, across multiple topics, which we restricted to only ones that are relevant to EPFL studies.

  Then, we also took the MCQ questions from the EPFL preference data, but we tweaked them. To be precise, note c the `chosen` answer in the original pair. In the altered pair, `chosen` is the letter corresponding to the answer that was given in long form in c, which we extracted using the GPT-wrapper, and `rejected` is c in its original form. This way, we want the model to understand it needs to answer with the letter first and foremost.

  Also, using the GPT wrapper, we retrieve the answer letter from the rejected answer and get the explanation of this answer. Using that we created another dataset that take a single letter as chosen and rejected answer. We make sure to remove the data that does not have a single letter and we choose to generate a random answer for the rejected one, when the chosen and the rejected where the same (it only concerned a small part of the data).

  Here is the prompt we made to the GPT Wrapper.

```
message = chat.ask("You will be given a prompt, that
      consists of a MCQ question, with a certain
      number of possible answers, and you will also
      have the correct answer along with an
      explaination. You will also get a rejected
      answer along with justification. And finally
      you will get a subject. You will have to
      seperate the question and possible answers from
       the correct answer and explaination, the
      rejected asnswer and justification and the
      subject. The output has to follow the following
       format in a json manner: { \n \"question\": <
      question>, \n \"possible_answers\": <possible
      answers>, \n \"answer\": <correct answer>, \n
      \"explaination\": <explaination of the correct
      answer>, \n \"rejected\": <rejected answer>, \n
       \"justification\": <justification of the
      rejected answer>,\n \"subject\": <subject>  \n
      } The correct answer has to be a single letter.
       For example: { \n \"question\": \"What is the
      capital of France?\", \n \"possible_answers\":
      [\"A. Paris\", \"B. London\", \"<C. Berlin\"],
      \n \"answer\": \"A\", \n \"explaination\": \"
      Paris is the capital of France.\", \n \"
      rejected\": \"B\",  \n \"explaination\": \"
      London is the capital of England.\", \n \"
      subject\": \"Geography\" \n}")
message2 = chat.ask("Let's start with the following
      test: the prompt is:" + data["prompt"] + "and
      the answer is:" + data["chosen"] + "and the
      rejected answer is:" + data["rejected"]+ "and
      the subject is:" + data["subject"])
```

  We also used data from the **Massive multitask language understanding (MMLU)** (Hendrycks et al., 2021), which is a dataset containing MCQ questions coming from a variety of topics. To stay in the STEM fields and to have the data consistent with the preference data get in milestone 1, we choose the following subjects: *"abstract_algebra, college_biology, college_chemistry, college_computer_science, college_mathematics, college_medicine, college_physics, machine_learning, econometrics"* As `chosen` we used the correct answer (its letter) and as `rejected` another letter chosen at random.

  Finally, to evaluate the efficacy of our model, we used an evaluation dataset comprised of 1K samples from the same EPFL preference data, but retaining only the MCQ questions.

- **Evaluation method:** To first evaluate how competent our model was at identifying human preferences, we used reward accuracy during our training. Then, the same metric was used twice, once in each of our 2-step approach: after the short DPO training, and then after the longer one. Moreover, we used accuracy on the letters between our predictions and the gold truth labels, to have an idea of how well our model performs on MCQs.

- **Baselines:** To make sure our model is at least doing best than a student randomly choosing an answer, we go with the baseline of $\frac{1}{N}$ with N being the number of possible answers, as the minimal accuracy we would like to obtain.

  As for the improvements we suggest, the base model's performances is a baseline: if we are trying to improve, then it should perform better than the base one.

  Regarding optimizations, namely quantization, we take the first trained model's performance as the baseline. We expect optimizations to have some impact on performance, but we'd like to make it as minimal as possible.

- **Experimental details:** We trained using the built-in `DPOTrainer` from the `trl` package (von Werra et al., 2020), which is a package built on HuggingFace with tools for reinforcement learning in the context of Transformers. We paired it with an AdamW optimizer and a learning rate scheduler with polynomial decay between $1e^{-4}$ and $1e^{-7}$. Overall, training for M2 took 6 hours, resulting in a validation reward accuracy of 0.67. The difference between the trained model's accuracy on the training data and the validation one can be explained by a small over-fit, which seems normal as our model is small, and we have 3 epochs.

- **Results:** The evolution of the training accuracy on the first training is reported in Figure 1. The evolution of the training accuracy after each of the 2 steps described above are reported in Figure 2.

  Accuracy on evaluation data with different data on our model or on the base one are reported in Table 1. The results labelled with M2 represent training done on the M2 model and the ones labelled OELM represent training done directly on the base Apple OpenELM model. EPFL means that only the EPFL MCQ dataset was used, MMLU that only the MMLU MCQ dataset was used, and EPFL + MMLU means that both datasets were used.

  The quantization to 4-bits per weight was done on the M2 EPFL model as it yielded the best results. This lead to a decrease in model memory footprint from *1'090'305'024B* to *201'466'368B*. Reward accuracy is also reported in Table 1.

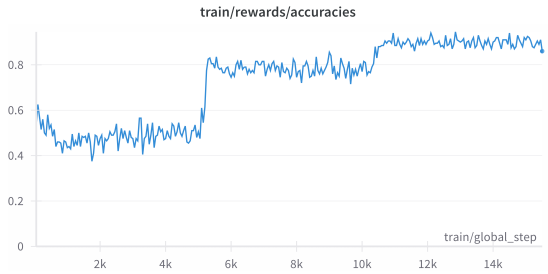

Figure 1: WANDB generated reward accuracy for M2

Model	Accuracy
M2 EPFL	0.34
OELM EPFL	0.32
M2 MMLU	0.27
OELM MMLU	0.26
M2 EPFL + MMLU	0.32
OELM EPFL + MMLU	0.30
Quantized M2 EPFL	0.27

Table 1: Reward accuracies at the end of training

5 Analysis

5.1 Model's performance

When it comes to the model's performance, our results show that the M2 model, which was trained on preference data an extra round compared to the basis OELM model, consistently performs better across all datasets, which indicates that our DPO training actually enhanced and fine-tuned our model.

While we are definitely above the random guess baseline, the scores suggest room for improvement. The highest accuracy achieved is 0.34, so around one third of correct answers. This can be rationally explained by the fact that our model is overall relatively small, so its ability to perform well on

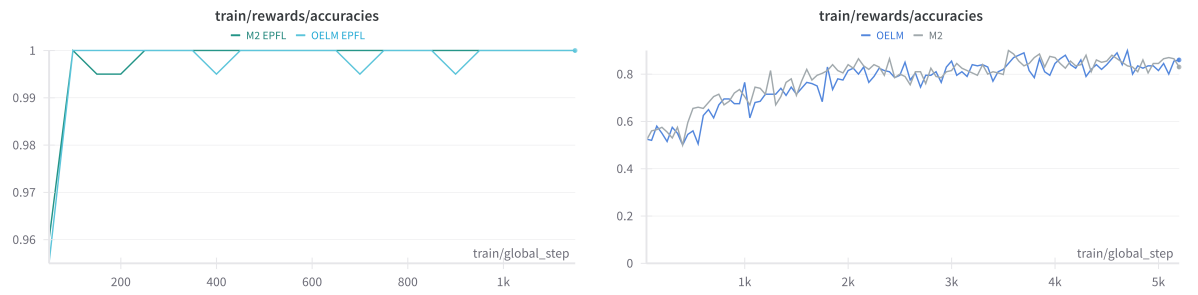

Figure 2: WANDB generated reward accuracies evolution for the first "letter only" training (Left) and the second "correct answer" training (Right).

questions requiring deeper reasoning, even with fine-tuning, might still be limited. The quantity of data to train on might not be enough either to tackle the inherent difficulty of STEM questions.

5.2 Ablation studies: Impact of Training Data

The performance on the EPFL dataset is generally higher compared to the MMLU dataset. In general, this suggests that training on institution-specific data can be beneficial. That being said, the difference in performance can also easily be explained by the size difference of the two sets, the EPFL one being around 7 times bigger.

An interesting result appears when combining both datasets. For both M2 and OELM models, the combination, while bringing forth more data, leads to a 2% decrease in accuracy compared to the EPFL only result. This implies that, while more general training data can be helpful, the specific EPFL focus might have been more useful in this case.

5.3 Quantization

Regarding the quantized version of our model, we achieved a decrease of more than 80% of the model's original size. With that great save in memory space comes the price of decrease in model accuracy, losing 7%. Nevertheless, this is still a better improvement compared to our baseline of random guess.

6 Ethical considerations

It's crucial to acknowledge that our tool has the potential to be misused. Most obviously, exam cheating becomes a possibility. Students could exploit our tool's ability to answer questions quicker, gaining an unfair advantage on exams. In its current form, our model limits this problem by being itself limited in its performance, as if a 4 question-single answer MCQA follows EPFL's grading scheme of $+3$ for a correct answer, -1 for a wrong answer, the student would have an expected $0.32/3$ points per question: it's better than 0, but the student still can't pass a class without understanding it! Nonetheless, let's imagine a model with a way better performance, and imagine a student feeding a complex physics problem into the system and getting instant solutions, bypassing the need for actual understanding. This would undermine the entire purpose of exams, learning and studies, which is to gain genuine knowledge and critical thinking skills which are then assessed. Additionally, The tool's ability to answer scientific inquiries rapidly could be used for unethical purposes. A malicious individual with limited programming knowledge could potentially exploit our system as a "confused deputy". For instance, they might input a vulnerable piece of C code and receive guidance on how to exploit it, creating security risks. Similarly, a student might attempt to obtain dangerous chemical knowledge, such as how to synthesize illegal substances using the tool's understanding of chemistry protocols. Let it be noted that some security measures could be put in place, but this goes beyond the scope of this project.

Also, our model can reflect bias, even in seemingly objective fields like math, computer science, and physics. This depends on how it's applied. Consider a question that uses statistics to compare salaries based on gender. As our tool becomes more and more fine-tuned with vast amounts of data, it could unintentionally mimic societal biases, leading to inaccurate answers. Furthermore, some scientific topics, such as the debate on vaccines efficiency, are ongoing discussion within the scientific community. Our tool could potentially contribute to confusion by providing answers that align with a specific viewpoint within this debate, neglecting

the lack of total consensus. For example, someone could ask a question about a patient's health based solely on their Body Mass Index (Callahan, 2021) which might lead to a biased answer. These are ongoing issues that have yet seen solutions, but the data selection process must be reviewed with utmost dedication as to limit as much as possible the pitfalls of these problems.

7 Conclusion

We investigated the creation of a chatbot capable of answering multiple-choice questions in STEM fields. Our findings demonstrate that fine-tuning with DPO is an effective strategy. Additionally, training the model on domain-specific data from EPFL courses seems to lead to better performance. While these results are promising, the overall performance of our model suggests limitations in its ability to tackle the problem at hand. The analysis could be done with more metrics

Quantized results show interesting prospects to gain considerable memory space while maintaining performance. More in depth studies could be made to try and identify certain specific weights on which we could focus the quantization process, in order to minimize the loss in accuracy while keeping the lowest memory usage possible.

Overall, this project demonstrates the potential of DPO for creating educational chatbots, especially in STEM fields. By building upon this, by for example using the same techniques but on bigger amounts of data or bigger models, might show even more promising results and help bring AI to students in a meaningful way.

8 Contribution statement

- **Design of model and training strategies**: Whole team
- **Implementation of training infrastructure**: Baptiste Lecoeur
- **Dataset development**: Ghita Tagemouati
- **Results and analyses**: Ahmed Elalamy

References

Patrick Bassner, Eduard Frankford, and Stephan Krusche. 2024. Iris: An ai-driven virtual tutor for computer science education.

Alice Callahan. 2021. Is b.m.i. a scam? In *NY Times*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners.

Nathan Lambert, Lewis Tunstall, Nazneen Rajani, and Tristan Thrush. 2023. Huggingface h4 stack exchange preference dataset.

Sachin Mehta, Mohammad Hossein Sekhavat, Qingqing Cao, Maxwell Horton, Yanzi Jin, Chenfan Sun, Iman Mirzadeh, Mahyar Najibi, Dmitry Belenko, Peter Zatloukal, and Mohammad Rastegari. 2024. Openelm: An efficient language model family with open training and inference framework.

Ryan Park, Rafael Rafailov, Stefano Ermon, and Chelsea Finn. 2024. Disentangling length from quality in direct preference optimization.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. 2020. Trl: Transformer reinforcement learning. `https://github.com/huggingface/trl`.