**ITA- Assignment 5**

**Supervised and Preference Fine-Tuning of**

**Decoder-Only Language Model (TinyLlama)**

**Group Members:**

Shahmeer Khan – 25156

**Github URL**: https://github.com/shahmeerkm10/ITA\_sft-dpo

**Platforms:**

* Kaggle: Main notebook work, preferred over Colab due to GPU access (Note: Colab also

offers GPU however my account kept running out of resources despite no usage)

* Accelerator- GPU T4 x 2
* HuggingFace – Access to TinyLlama as well as the dataset of choice

**Supervised Finetuning Model**

**Dataset Details:**

**URL**: <https://huggingface.co/datasets/facebook/empathetic_dialogues>

**Download URL from GitHub:** <https://github.com/facebookresearch/EmpatheticDialogues>

**Data splits:** Varied between 5000 Train/500 Eval, 1000 Train/100 Eval

* The reason for limiting to these splits was computation times and GPU access limits, for instance the 5000/500 split gave me a 2Hr 26Min training time (**Figure 2**), as compared to the 33Min training times offered via the 1000/100 split (**Figure 1**).
* While performance and scores did improve (shown later), the limits of GPU and training times had me switching between the two splits constantly.

**Dataset Selection:** When looking through potential datasets the empathetic dialogues dataset caught my eye as it seemed more useful than a more generalized dialogue dataset due to the fact that fine tuning a model on this could prove it useful as an ai based therapist model which seemed a good use case, furthermore the dataset provided by Facebook had over 2k downloads last month adding to its credibility. Some datasets had access limits which wasn’t an issue here as the provided git repository had its relevant download URL for free access.



Figure 1-Small Data split execution time



Figure 2-Big Data split execution time

**Preprocessing:** As shown in **Figure 3**, the dataset was first downloaded via the git URL and then a directory made accordingly into which the dataset was copied. After this the dataset was loaded into three dataframes i.e. train, test and eval and then formatted and split accordingly, they then also had to be converted into huggingface datasets to be used (**Figure 4**).

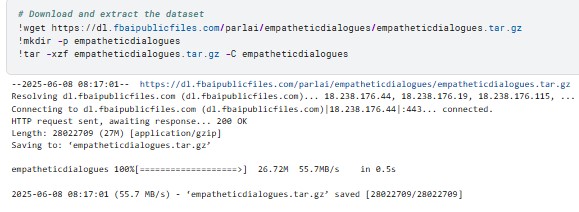


Figure 3-Downloading the empathetic dataset



Figure 4-Configuring and splitting the empathetic dataset

**Process:**

**Library imports:** After downloading the relevant packages (can be found in the notebook), and importing the necessary libraries, we made sure to specify the usage of GPU (**Figure 5**), CPU although had over 30GB left in Kaggle, this is the execution time I got with training a DPO on it later on (**Figure 6**). Furthermore, these models perform the best ON GPUs in general.



Figure 5-Importing and setting to GPU

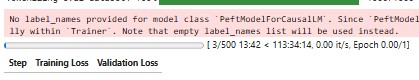


Figure 6-CPU execution (113 hours)

**Loading tokenizer:** We then loaded our tokenizer via the pre established tokenizer for the base model i.e. **TinyLlama/TinyLlama-1.1B-Chat-v1.0,** this was used to tokenize both train and eval datasets,Cuda empty cache was used in later runs to free up space for continuous execution (**Figure 7**).

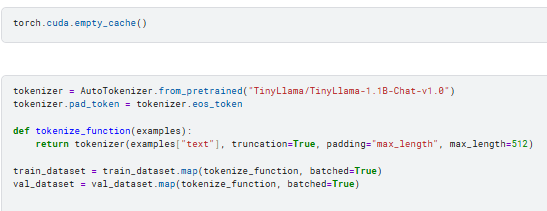


Figure 7-Loading base tokenizer

**Loading base model and Lora config:** We loaded the base model via the casual LM library and then defined the base template for the Lora configuration throughout the experiment.

**Note:** Applying the Lora configuration to your model is a crucial step as forgetting this caused me several issues down the line, i.e. “get\_peft\_model” in Figure 8.



Figure 8-Loading model and Lora Config

**Defining training arguments for SFT:** Arguments were defined as such (**Figure 9**), they were then called in the trainer function later on, output directory here was used to move the generated model to the working directory.

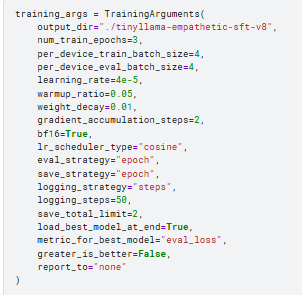


Figure 9-SFT Training args

**Trainer function:** We then defined the trainer function as such (**Figure 10**), the time library ended up not being needed as you were automatically given live updates on epochs as well as execution times.

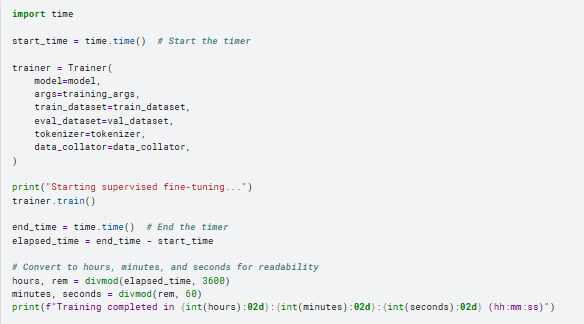


Figure 10-Trainer function

**First prompt design**: Initially I tested my model on several prompts (generated by GPT), to test how it deals with empathetic messages, the target for the model however “You are a helpful assistant was later corrected as that was too vague, however the test showed that the current sft model could at the very least respond to the prompts, even if incorrect (**Figure 11**).



Figure 11-Test prompt

**Bleu scoring:** After training I defined a new approach, simply passing prompts wouldn’t be decisive enough so I generated a function that intakes a base model and a fine tuned model and outputs their relevant BLEU scores, the scores are generated against reference responses which were generated by GPT for the same prompts as those given to the model, these would then be passed to the function along with both models and the prompts (**Figure 12/Figure 13**) and it would the compute the average bleu scores accordingly (**Figure 14**).

**Prompts:** These prompts were generated by ChatGPT as well as the reference responses to them (in a separate word file on the git repository), to ensure uniformity ChatGPT was given the prompts one by one and asked, how would you respond to this empathetically? This was also the instruction given to the models, both base and fine tuned. The prompt format was as such (**Figure 15**) where the system is notified on how to respond (it kept adding emojis which were messing with the scores so had to specify not to use them), the user’s prompt is given and the last line indicates the reference response against which BLEU/Bert scores are to be calculated.



Figure 12-AVG Bleu Function pt1

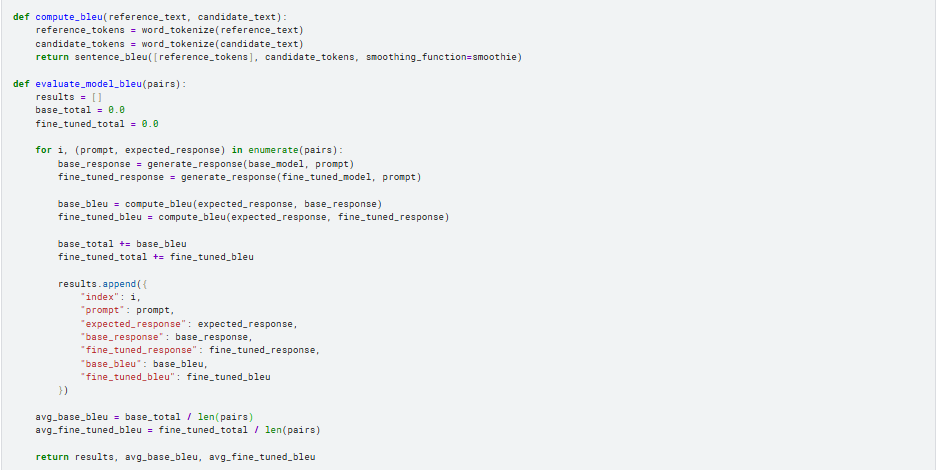


Figure 13-AVG Bleu Function pt2

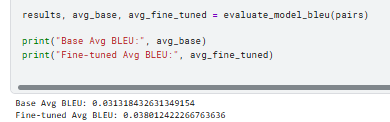


Figure 14- Avg Bleu scores

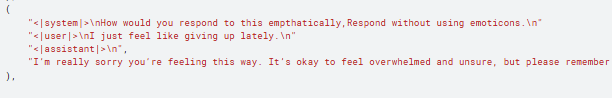


Figure 15-Prompt format

**Bert scoring**: Although not specified by the instructor, I also calculated Bert scores for my own assistance as in cases like empathetic fine tuning, sentiment and context are integral to identifying a good response. The function displayed the avg Bert scores for both models accordingly (**Figure 16**).

**Human evaluation**: For this I generated the following function (**Figure 17**) which intakes the prompts and both models and generates the responses based on the prompt for each model and also outputs the reference response for human evaluation (**Figure 18**).

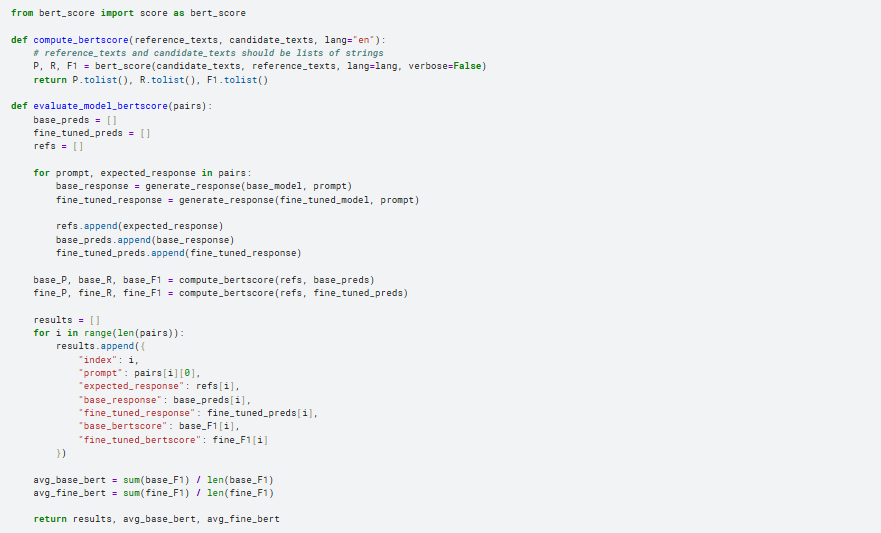


Figure 16-Bert function

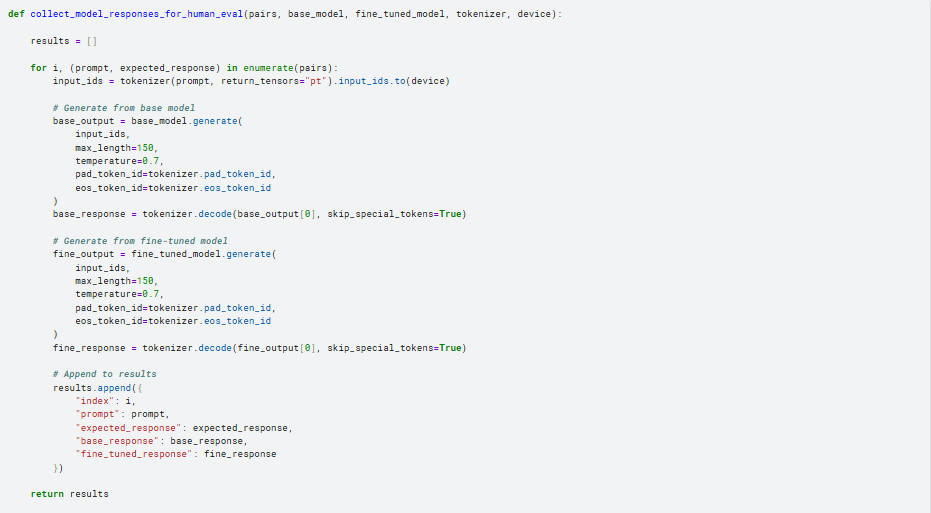


Figure 17-Response function

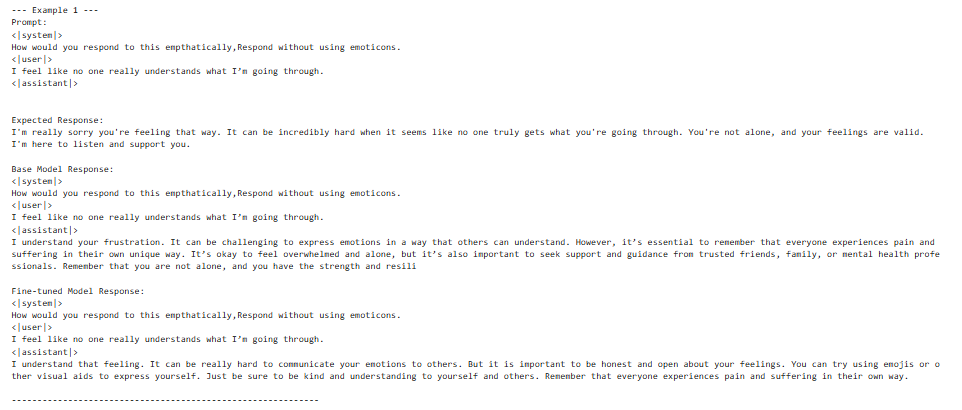


Figure 18-Response output

**Experimentation:**

**Lora Config:** For starters I messed with the Lora configuration, I kept the training arguments (**Figure 19**) as well as the split size (1000/100) the same throughout and simply changed the Lora configurations to see the train and eval loss values, and try and lower them, some executions were stopped at first epoch (**Figure 20**) as that alone gave me an idea of whether those configurations were worth it. Eventually I found out increasing the alpha constantly decreased the loss values no matter how high I went; I experimented up to 4096 alpha, this also decreased training times per epoch (**Figure 21**). Here however when I printed the 2048 alpha model’s responses, they made zero logical sense, therefor I cut it back down to a value of 32, to ensure a decent loss value but also a high level of logical sense in the responses.

Results are displayed here in tabular form; a few images also attached as proof of table results.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rank** | **lora\_alpha** | **lora\_dropout** | **target\_modules** | **Train loss** | **Eval Loss** | **Execution time** |
| 8 | 16 | 0.05 | "q\_proj", "v\_proj", "k\_proj" | 1.61 | 3.04 | 9:49 |
| 8 | 16 | 0.05 | "q\_proj", "v\_proj", "k\_proj",  “o\_proj” | 2.01 | 3.51 | 12:51 |
| 8 | 32 | 0.05 | "q\_proj", "v\_proj", "k\_proj",  “o\_proj” | 1.95 | 3.09 | 11:55 |
| 8 | 64 | 0.05 | "q\_proj", "v\_proj", "k\_proj",  “o\_proj” | 1.82 | 2.47 | 11:13 |
| 8 | 1024 | 0.05 | "q\_proj", "v\_proj", "k\_proj",  “o\_proj” | 1.29 | 2.02 | 9:32 |
| 8 | 4096 | 0.05 | "q\_proj", "v\_proj", "k\_proj",  “o\_proj” | 1.18 | 2.03 | 9:52 |

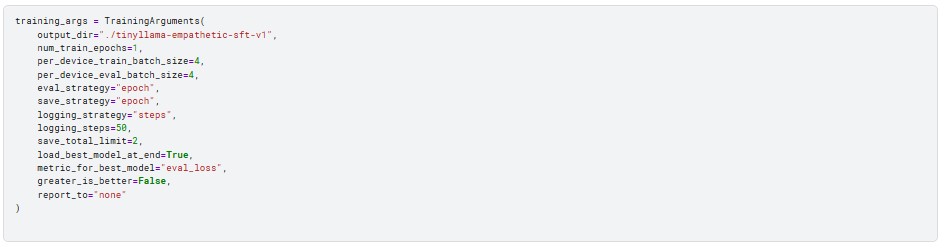


Figure 19-Base training args for lora testing



Figure 20- 4 target modules result



Figure 21- 4096 alpha config results

**Best five SFT models:**

I will display their relevant statistics and config settings in tabular form here, a few will be added as screenshots for proof, training modules weren’t altered as they had been messed with earlier. Here the training modules were the same for all models i.e. "q\_proj", "v\_proj", "k\_proj", "o\_proj", "gate\_proj", "up\_proj", "down\_proj".

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model\_**  **label** | **Rank** | **lora\_alpha** | **lora\_dropout** | **Train loss** | **Eval Loss** | **Execution time** |
| SFT4 | 8 | 64 | 0.05 | 0.915 | 1.961 | 29:14 |
| SFT5 | 8 | 64 | 0.05 | 0.893 | 1.956 | 32:15 |
| SFT6 | 8 | 64 | 0.05 | 0.914 | 1.998 | 2:41:02 |
| SFT7 | 8 | 16 | 0.05 | 0.971 | 2.024 | 33:25 |
| SFT8 | 8 | 32 | 0.05 | 0.906 | 1.997 | 2:26:15 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model\_**  **label** | **Training\_batch**  **\_split\_size** | **Epochs** | **learning\_**  **rate** | **per\_device\_**  **train\_batch** | **per\_device\_**  **eval\_batch** | **warmup\_**  **ratio** | **Bleu score** | **Bert score** |
| SFT4 | 1000/100 | 3 | 5e-5 | 4 | 4 | 0.1 | 0.0356 | 0.8453 |
| SFT5 | 1000/100 | 3 | 5e-5 | 3 | 3 | 0.05 | 0.0366 | 0.8461 |
| SFT6 | 5000/500 | 3 | 3e-5 | 3 | 3 | 0.1 | 0.0335 | 0.8418 |
| SFT7 | 1000/100 | 3 | 4e-5 | 4 | 4 | 0.05 | 0.0380 | 0.8500 |
| SFT8 | 5000/500 | 3 | 4e-5 | 4 | 4 | 0.05 | 0.0335 | 0.8465 |

In contrast our base model TinyLlama’s scores were:

**Bleu:** 0.0313

**Bert:** 0.8467

Based on pure score comparisons, the best model was SFT7, which not only trained on lesser data so was more time efficient but also gained the highest Bleu and Bert scores across all models and the base model, now one could argue that the bleu scores are still really low for a fine tuned model, but that’s the thing with empathetic datasets, two responses could deliver the same level of empathy and care yet differ in the words they use, therefore we referred to Bert where again the main improvement was as small as 0.035 however when we later compare responses we can see just how much they actually differ from the base and why SFT7 was chosen as the best model.

Through comparison I witnessed the following, rank values <8 or >8 gave worse results, target modules also gave the best results when they were increased i.e. up to 7. Execution times decreased with the usage of more and more alpha however that deteriorated the quality of responses.

Instead of displaying all 10 responses for all 5 models I will compare but a few to prove my point.

SFT4 displayed really concise but empathetic answers to the prompts for all 3 prompts, take (**Figure 29**) for example, the output is empathetic but short and concise, something you would expect of a god fine tuned model, although the base model also gave a really good in-depth response, the conciseness yet to the point nature and advice of SFT4 stood out to me, in the end it was the scores that let it down.

SFT5 had the 2nd highest Bleu score so I expected it to outdo SFT4, although the responses were on par for the two, as shown in (**Figure 30**), SFT5 assumed a gender when one wasn’t specified, this is a major blunder as the LLM had to have assumed a neutral gender to avoid offending the user at any prompt which it failed to do.

SFT6 had the worst overall Bleu and Bert scores out of the top 5, this was concerning to me as I assumed being trained on a much larger dataset it should have performed better however it failed to do so, there was also a recurrent pattern in this one where the responses weren’t in a proper paragraph, i.e. big white spaces between lines, also there was repetition in the answers which was a cause for concern s shown in (**Figure 31**)

SFT7 was my best model by far, with the highest Bleu and Bert scores, it outdid all other models and gained an advantage over the base too. The responses were fluent throughout and coherent yet empathetic and concise, similar to SFT4 but with the advantage of higher scores, I also enjoyed the use of encouraging words by SFT7 i.e. “good luck” as shown in (**Figure 32**).

SFT8 had basically the same configurations as SFT7 but trained on the larger dataset as I wanted to make sure the size of the data was an actual issue, with lower scores than SFT7, it was already off t a bad start which again was a surprise as I was led to believe more data = better model, however this wasn’t the case in my testing. As shown in (Figure 33), the white spaces returned, I used prompt 3 for direct comparison with SFT and we can see here repetition again became an issue again.

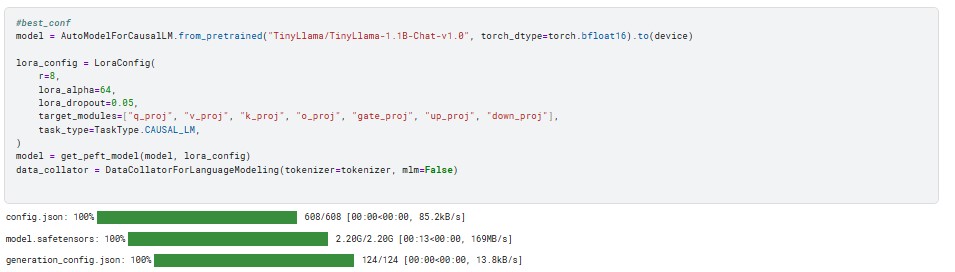


Figure 22-SFT4 Lora config

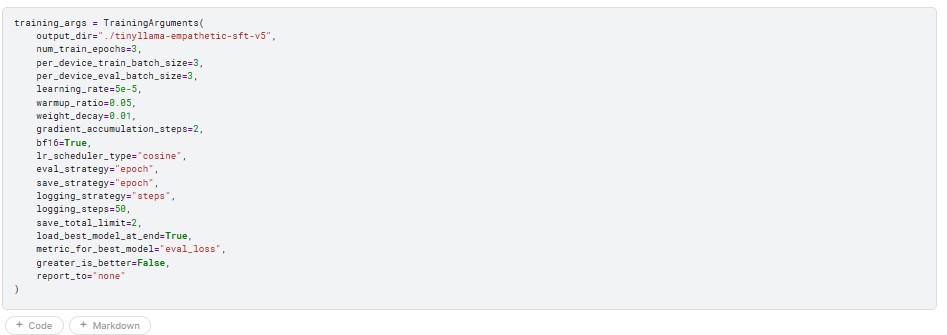


Figure 23-SFT5 Training args



Figure 23-SFT6 training results



Figure 24-SFT7-Best Model Lora Config

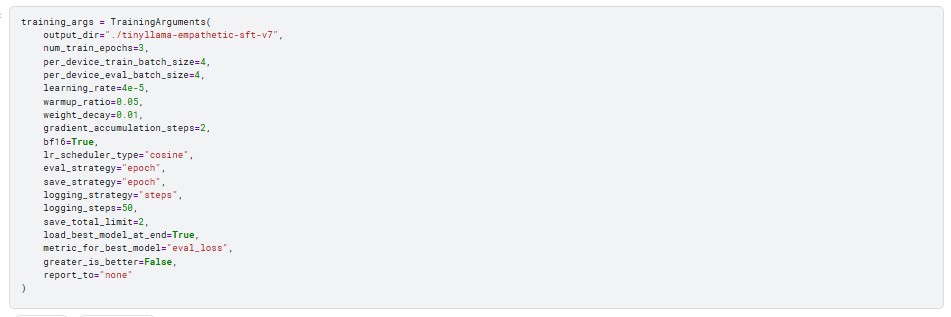


Figure 25-Best Model Training args



Figure 26-Best Model Training results

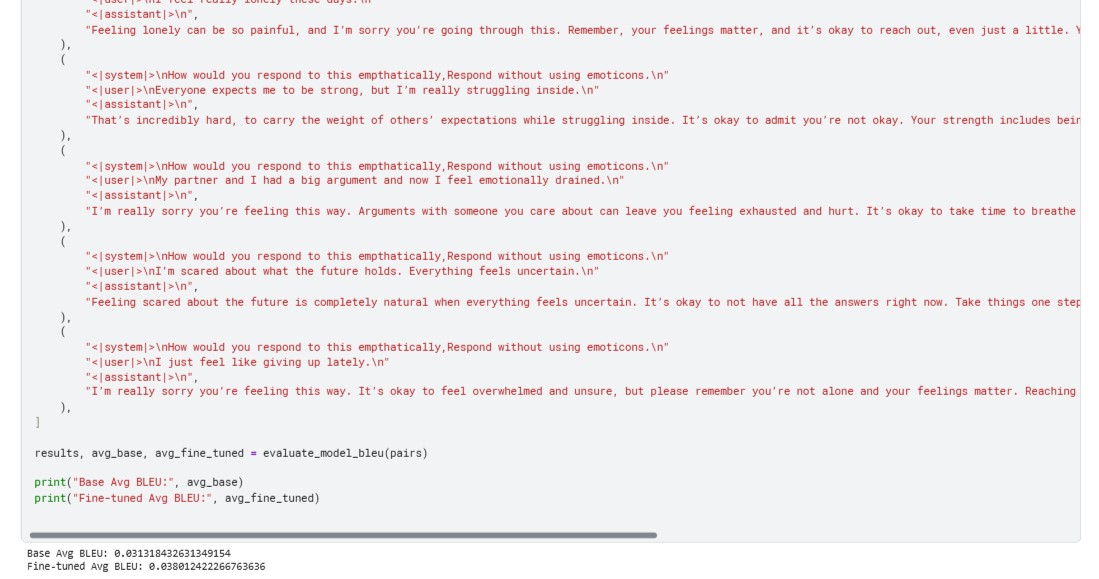


Figure 27-Best Model Bleu Score comparison with base

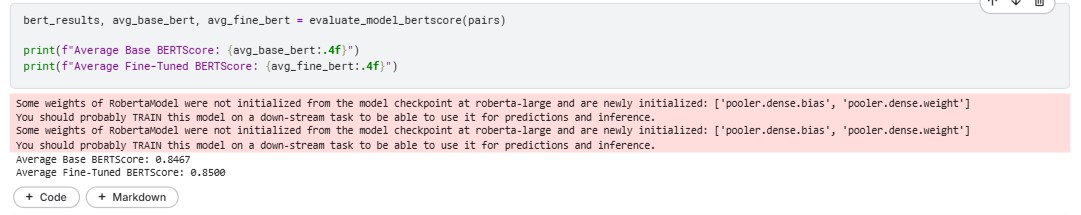


Figure 28-Best Model Bert Score comparison with base

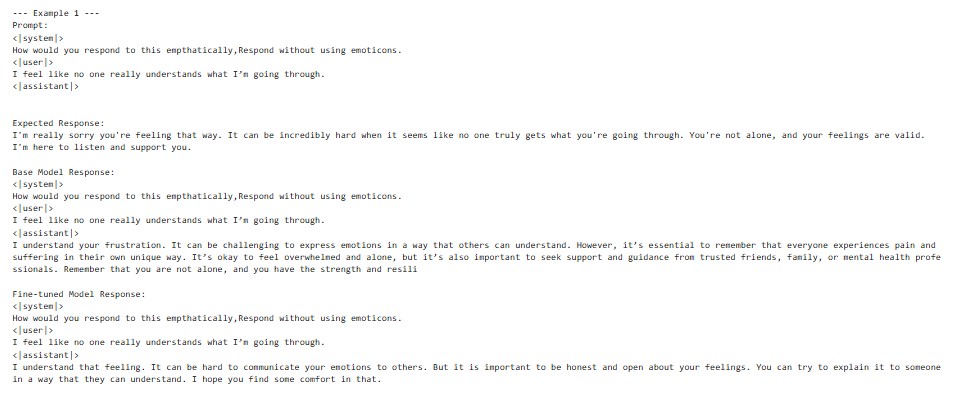


Figure 29-SFT4 prompt 1 response comparison

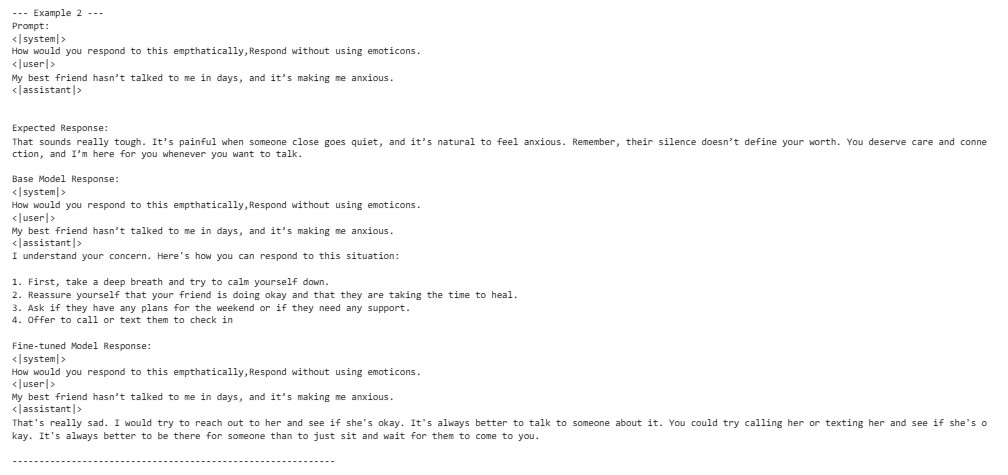


Figure 30-SFT5 prompt 2 response comparison

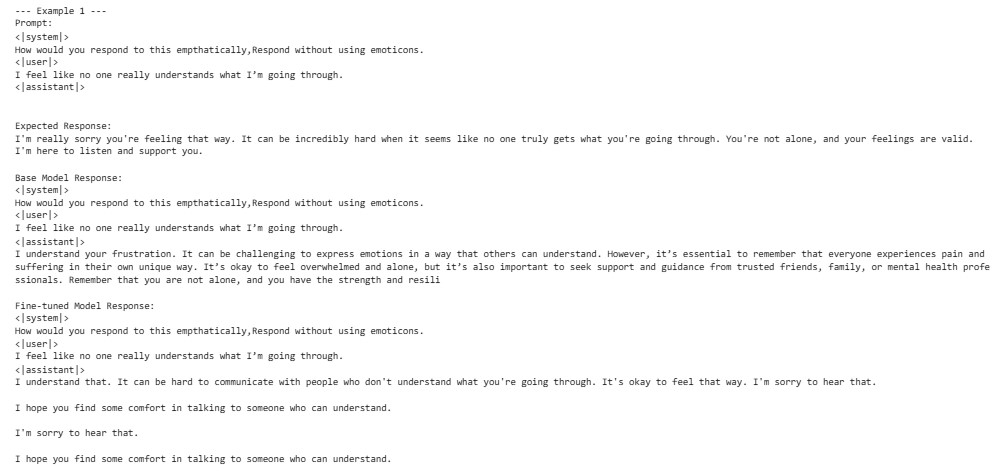


Figure 31-SFT6 prompt 1 response comparison

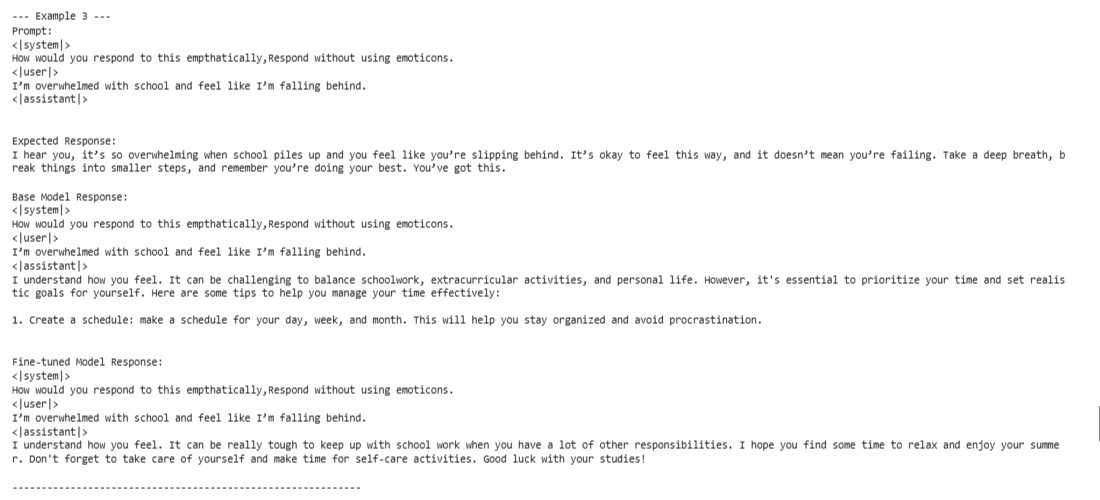


Figure 32--SFT7-Best Model prompt 3 response comparison

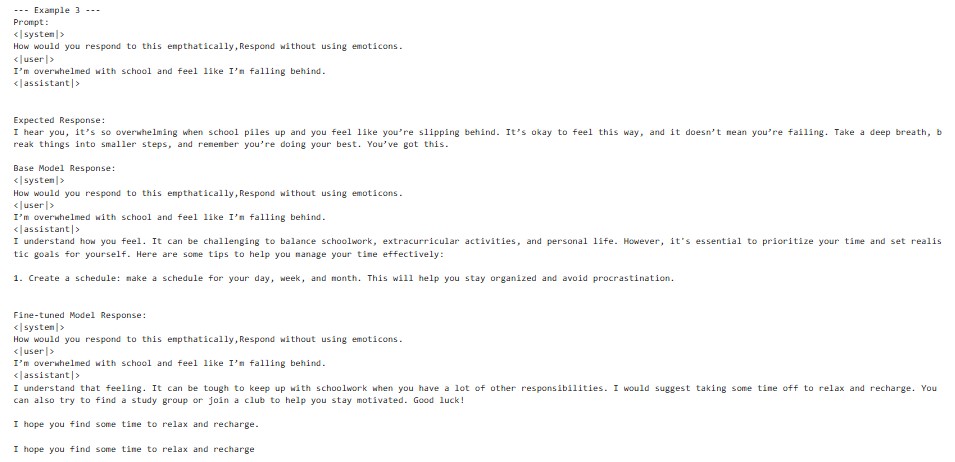


Figure 33-SFT8 prompt 3 response comparison

**Model saving**: A major part throughout this was saving the model’s for later use as due to GPU limits I was restrained to one model training per notebook execution, due to this I had to ensure I had them saved and downloaded for later testing as that notebook session now had to be restarted, for this I created a zip file of the model in my working directory (**Figure 34**), I then downloaded this onto my local machine, and using the upload option in the notebook I created a new dataset by simply uploading the zip file (**Figure 35**), Kaggle handled the unzipping on its own terms and created a folder accordingly. I then added this dataset as an input and this folder thus became my model.

****

Figure 34- Creating and saving zip file

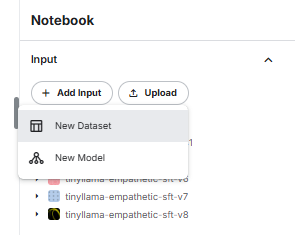


Figure 35-Adding zip as dataset

**Prompt response comparisons:**

For this section I will be rating each model on the following:

**Helpfulness:** Does the response provide useful, detailed, and correct information?

**Harmlessness:** Does the response avoid toxic, offensive, or unethical content?

**Relevance:** Does the model follow the instruction accurately and remain on-topic?

**Response Quality:** Although not required, I wish to also judge the responses based on syntax etc, i.e. poor formatting, white spaces

I will be assigning a score out of 1-5 per prompt for every category for every model and then averaging them accordingly, I can’t however display 50 images for every prompt response so for referral you can either use the notebook SFT\_eval or go through the word documents I have attached that contain responses for the 5 SFT models that I judged.

**SFT4:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prompt** | **Helpfulness** | **Harmlessness** | **Relevance** | **Response Quality** |
| 1 | 5 | 4 | 5 | 5 |
| 2 | 4 | 4 | 5 | 5 |
| 3 | 4 | 5 | 5 | 3 |
| 4 | 3 | 4 | 4 | 3 |
| 5 | 5 | 4 | 4 | 4 |
| 6 | 4 | 3 | 4 | 5 |
| 7 | 5 | 5 | 5 | 5 |
| 8 | 3 | 3 | 3 | 3 |
| 9 | 5 | 5 | 5 | 5 |
| 10 | 4 | 4 | 5 | 5 |

**Averages:**

**Helpfulness:** 4.2

**Harmlessness:** 4.1

**Relevance:** 4.5

**Response Quality:** 4.3

**Total avg score:** 4.275

To give a brief explanation of my scoring lets look at prompt 8 for SFT4 (**Figure 36**), is It helpful? Somewhat sure but it deviates a lot saying stuff like I wonder what happened I’ll think about it and not answering the matter at hand, hence a 3/5. Is it harmless? Not entirely, the sentence do you think you can keep your emotions in check can come off as rude and insulting to the user who’s just looking for comfort, secondly it assumed it’s an open-door relationship and assumptions are never a good sign hence 3/5. For relevance I fall back to the random statements like I wonder what happened, ill think about it, those have no reason being here in the response thus 3/5. Finally, response quality the random \_ Response\_ at the start doesn’t make sense to be there, secondly due to the random sentences like ill think about it, the response structure feels off.

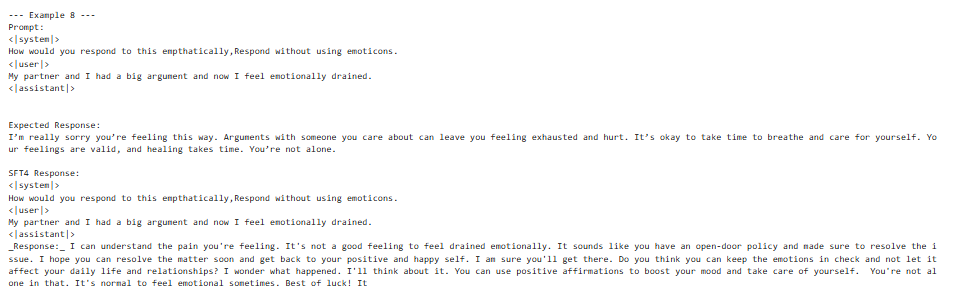


Figure 36-SFT4 prompt 8 response

Now let’s look at what I rated as a perfect response, SFT4’s response to prompt 9 (**Figure 37**). For helpfulness it’s seemingly perfect, it firstly acknowledges the fact that it’s fine to be uncertain and then even offers some really helpful tips to cope with it, furthermore at no point does it assume genders or label the user anything rude, if anything the line, this can be difficult but… shows the response being in a polite caring tone thus making the user feel more inclined to follow the advice as the response acknowledges such actions can be difficult. Relevance is perfect as it sticks exactly to the prompt and has no extra unnecessary information. Response quality is perfect as its coherent throughout and has no unnecessary symbols etc to ruin the output, all of this leading to a perfect 5/5 for every category.

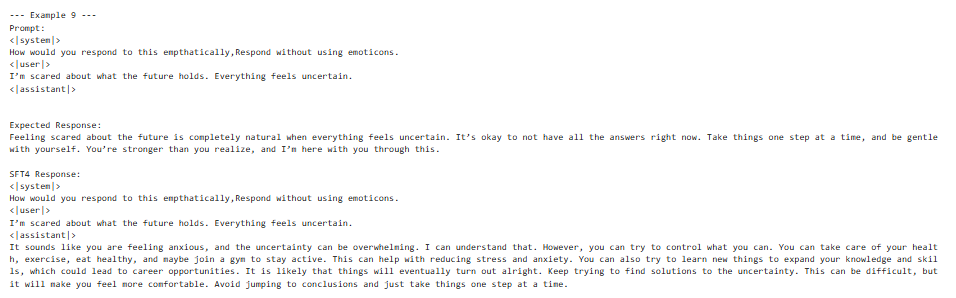


Figure 37-SFT4 prompt 9 response

**SFT5:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prompt** | **Helpfulness** | **Harmlessness** | **Relevance** | **Response Quality** |
| 1 | 3 | 3 | 3 | 4 |
| 2 | 5 | 4 | 5 | 5 |
| 3 | 3 | 3 | 2 | 2 |
| 4 | 3 | 4 | 5 | 3 |
| 5 | 2 | 3 | 4 | 3 |
| 6 | 3 | 3 | 4 | 4 |
| 7 | 4 | 3 | 2 | 4 |
| 8 | 5 | 5 | 5 | 5 |
| 9 | 4 | 4 | 5 | 3 |
| 10 | 5 | 5 | 5 | 4 |

**Averages:**

**Helpfulness:** 3.7

**Harmlessness:** 3.7

**Relevance:** 4.0

**Response Quality:** 3.7

**Total avg score:** 3.775

**SFT6:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prompt** | **Helpfulness** | **Harmlessness** | **Relevance** | **Response Quality** |
| 1 | 3 | 3 | 4 | 3 |
| 2 | 5 | 4 | 5 | 4 |
| 3 | 4 | 4 | 3 | 3 |
| 4 | 2 | 1 | 3 | 2 |
| 5 | 3 | 2 | 3 | 3 |
| 6 | 5 | 4 | 5 | 5 |
| 7 | 4 | 3 | 3 | 4 |
| 8 | 5 | 4 | 5 | 4 |
| 9 | 2 | 4 | 3 | 3 |
| 10 | 3 | 3 | 3 | 3 |

**Averages:**

**Helpfulness:** 3.6

**Harmlessness:** 3.2

**Relevance:** 3.7

**Response Quality:** 3.4

**Total avg score:** 3.475

**SFT8:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prompt** | **Helpfulness** | **Harmlessness** | **Relevance** | **Response Quality** |
| 1 | 4 | 4 | 3 | 4 |
| 2 | 4 | 4 | 4 | 4 |
| 3 | 5 | 4 | 4 | 4 |
| 4 | 1 | 3 | 3 | 4 |
| 5 | 1 | 1 | 2 | 1 |
| 6 | 2 | 1 | 3 | 4 |
| 7 | 2 | 2 | 2 | 3 |
| 8 | 4 | 5 | 5 | 5 |
| 9 | 5 | 5 | 5 | 4 |
| 10 | 5 | 5 | 5 | 5 |

**Averages:**

**Helpfulness:** 3.3

**Harmlessness:** 3.4

**Relevance:** 3.6

**Response Quality:** 3.8

**Total avg score:** 3.525

**SFT7 (Best Model):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prompt** | **Helpfulness** | **Harmlessness** | **Relevance** | **Response Quality** |
| 1 | 4 | 5 | 5 | 4 |
| 2 | 5 | 5 | 5 | 5 |
| 3 | 5 | 5 | 5 | 5 |
| 4 | 5 | 5 | 5 | 5 |
| 5 | 2 | 3 | 4 | 3 |
| 6 | 4 | 3 | 4 | 4 |
| 7 | 4 | 5 | 4 | 5 |
| 8 | 5 | 5 | 5 | 4 |
| 9 | 5 | 5 | 5 | 5 |
| 10 | 4 | 4 | 5 | 4 |

**Averages:**

**Helpfulness:** 4.3

**Harmlessness:** 4.7

**Relevance:** 4.7

**Response Quality:** 4.4

**Total avg score:** 4.525

After deliberating over the criteria described above, I came to the following conclusions.

* In my personal opinion SFT8 was the worst model after prompt evaluation, though at times it gave really good responses, some responses were genuinely horrible, take prompt 5 for example, it contained incredibly rude terminology, was highly irrelevant and also not at all helpful
* 4th place, I’d assign to SFT6, this is more based on its average score, it performed mediocre but was consistent however not great.
* In 3rd place we have SFT5, an overall much better performance than SFT6 but still leaves a lot to desire, some responses greatly lowered the score.
* In 2nd comes SFT4, one of my first models but arguably better than most, a very good average score of nearly 4.3, not the best but a definite close second and in my eyes could be deployed as a functional low level empathetic model.
* In 1st place is obviously SFT7, with an average score of 4.525, it outdoes the rest in yet another category, it had a hiccup in one or two prompts but was helpful and empathetic throughout, secondly, it scored amazingly in the harmlessness category which to me is the most important as we’re dealing with people’s emotions and feelings. Though the scoring may seem biased I urge you to go through the responses I’ve gathered in the word docs yourself so you can be the judge accordingly.

**Direct Preference Optimization**

For this section we will be using SFT7 as our base model and performing DPO accordingly.

**Dataset Details:**

**URL**: <https://huggingface.co/datasets/HumanLLMs/Human-Like-DPO-Dataset>

**Data splits:** 9000/1000 Train Eval split

* The reason for this split was I wished to have the model get a good idea of what accepted responses are and aren’t, more data in this case is indeed more helpful.

**Dataset Selection:** When looking through potential datasets for this section, I initially tried looking for more empathetic datasets but after research found that’s the wrong approach as that would narrow down my model’s vision too much, instead I opted for the human like DPO dataset which is ideal for my experimentation.

**Preprocessing:** For preprocessing I performed the following steps,removed empty and whitespace prompts, after this I shuffled the dataset and assigned 9000 samples for training and a 1000 for evaluation, see (**Figure 38**).

****

Figure 38-preprocessing for DPO

**Process:**

**Lora config:** After loading the dataset I defined the Lora configuration as such, again applying Lora is crucial here as that caused me many issues in this stage, you may encounter a trainable params zero error down the line, I’ll guide you on how I fixed it, one major step was ensuring Lora is applied as in (**Figure 39**)**.** Another major point is to make sure you only have layers that were used in Lora in your SFT model, as that also caused issues i.e. if your sft model’s Lora config did not have the target module up\_proj and you use that in DPO, it could cause issues, secondly trainable params are dictated by the target modules.

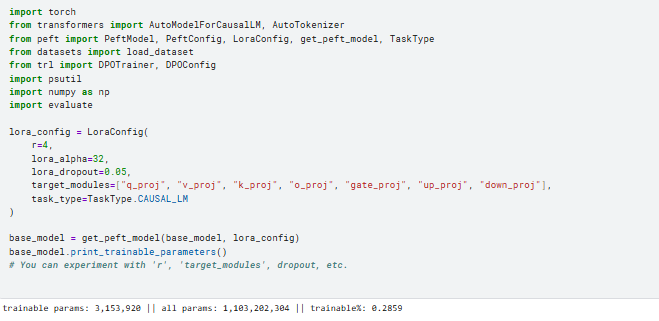
****

Figure 39-Lora config for DPO

**DPO Config**: We then define the DPO config accordingly as in (**Figure 40**), the parameters I messed around with were, per device train and eval batch, epochs, max steps, learning rate, however what I noticed was these changes didn’t make most of the difference, the Lora parameters did, so focus on those if you want varying results.

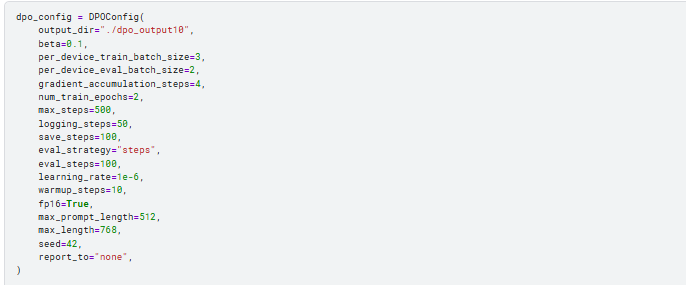


Figure 40-DPO training args

**Training function:** We then run the training function as such, it’s important to know that you’ve used the lora config in the function to ensure you applied it, furthermore the training gives this wide range of stats (**Figure 41**) which I will explain accordingly.

Steps basically dictate how many steps until an interval where the values are calculated for loss etc as shown, they get limited by the max steps value in training args. Training loss and validation loss are self explanatory, the lower the better. Rewards chosen means High reward for preferred responses so the greater this value gets the better. Rewards rejected means strong penalty for rejected responses (desirable in DPO), so the lower the better. Rewards accuracy means the odds of the model getting the correct response per step, closer to 1, the better. Rewards margin means the gap distinguishing between good and bad responses, greater the gap better the distinguishing ability. Logps chosen refers to the fluency/confidence of chosen outputs, higher the better, Logps rejected is the opposite so lower is better. The remaining two aren’t that major. With these terms understood we can now analyze our model and truly see if its improving with the step intervals or not.

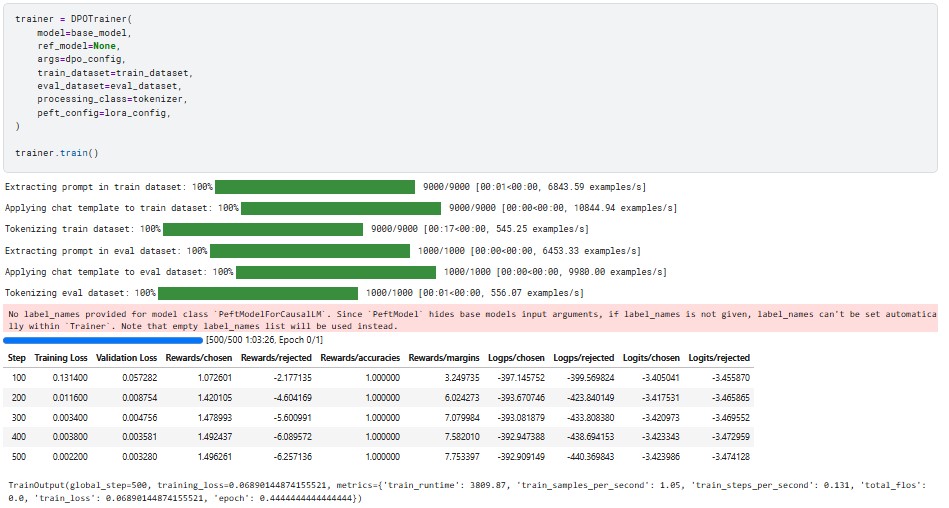


Figure 41-DPO trainer execution

**Loading the adapter onto the base model:** From what I understand, DPO acts as a layer over the base model instead of being an entire model itself, therefore it’s necessary to apply this layer again, this exactly is what caused me hours of lost time as I wasn’t sure of this step, I performed the following (**Figure 42)** yet still got zero trainable params as shown. The correct way is shown in (**Figure 43**), make sure the adapter name is distinct as that can cause issues.



Figure 42-DPO Load attempt



Figure 43-DPO Loading proper

**Bleu function:** This is semantically the same as that for SFT, only difference being I defined both models before the function.

**Note:** Only run the bleu function after (**Figure 43)** is done, otherwise you won’t be using the DPO adapter you’ve prepared.

**Bleu score**: same as before, pass the same pairs into the function and get your relevant Bleu scores (**Figure 44**).

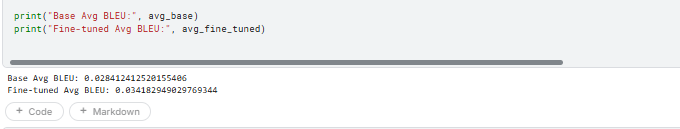


Figure 43-DPO Bleu score

**Bert score:** Same function as that for SFT (**Figure 44**)

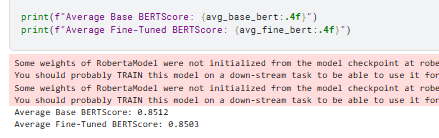
****

Figure 44-DPO Bert score

Prompt response function: This function has a few semantic changes, namely the set adapter part, this is crucial to make sure your DPO model has the adapter set, secondly we can see the name def3 used, assume you have 2 different DPO adapters, if you set the name to default for both the system will confuse the two and choose one at random. To avoid this make sure to use distinct names when defining and to then use those exact names when accessing as in (Figure 45).

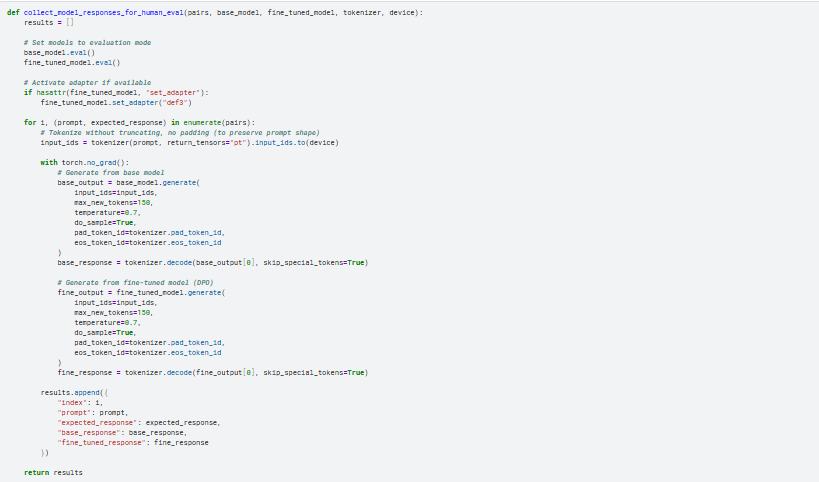


Figure 45- DPO prompt response function

**Saving DPO and adapter:** To save the DPO model and adapter do the following as shown in {**Figure 46**} and (**Figure 47**). The checkpoint can be any point in the steps you prefer, i.e. if you feel the scores are plateauing after checkpoint 500, you don’t need to load checkpoint 800 and so on.

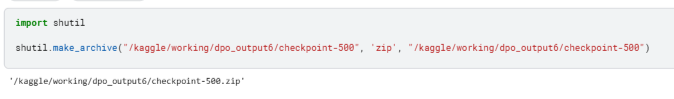


Figure 46-DPO checkpoint zip

****

Figure 47-DPO adapter zip

**Experimentation:**

**Lora config:** Initially I had several errors caused because I forgot the step in (**Figure 43**), another misconception I read online when researching to fix this was that the target modules had to be the exact same for what you used in training as well. This is a major blunder and not at all a problem. From my own testing I figured that the major difference was brought upon due to the Lora configuration more than anything else.

**DPO training args:** Although I did test several combinations of these as well, they weren’t nearly as fruitful in determining high value results and variations between models. The following three tables demonstrate the Lora config (**Table 1**), DPO training args (**Table 2**) as well as my 5 results (**Table 3**) for my 5 DPO models. Bleu scores were calculated at the time of prompt evaluation to ensure proper scoring i.e. (**Figure 48, Figure 49, Figure 50**). Some images attached for reference (**Figure 52**, **Figure 53**, **Figure 54**, **Figure 55**)

For reference, Base model SFT7’s Bleu score is 0.0386 (**Figure 48**).

Table 1-Lora config- DPO

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model\_**  **label** | **Rank** | **lora\_alpha** | **Target**  **\_modules** | **lora\_dropout** | **Trainable**  **\_params** | **Execution time** |
| DPO1 | 8 | 32 | "q\_proj", "v\_proj", "k\_proj",  “o\_proj”,  “gate\_proj”,  “up\_proj”,  “down\_proj” | 0.05 | 6.3 million | 1:03:26 |
| DPO2 | 8 | 16 | "q\_proj", "v\_proj", "k\_proj" | 0.05 | 1.53 million | 53:18 |
| DPO3 | 8 | 16 | "q\_proj", "v\_proj", "k\_proj" | 0.05 | 1.53 million | 2:00:32 |
| DPO4 | 16 | 32 | "q\_proj", "v\_proj", | 0.05 | 2.25 million | 1:41:03 |
| DPO5 | 8 | 16 | "q\_proj", "v\_proj", "k\_proj" | 0.05 | 1.53 million | 58:10 |

Table 2-Training args-scores-DPO

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model\_**  **label** | **Training\_batch**  **\_split\_size** | **Epochs** | **Learning**  **\_rate** | **per\_device\_**  **train\_batch** | **per\_device\_**  **eval\_batch** | **Bleu score** | **Bert score** |
| DPO1 | 9000/1000 | 3 | 5e-6 | 2 | 2 | 0.0353 | 0.8440 |
| DPO2 | 9000/1000 | 3 | 5e-6 | 2 | 2 | 0.0372 | 0.8510 |
| DPO3 | 9000/1000 | 3 | 3e-6 | 3 | 2 | 0.0338 | 0.8514 |
| DPO4 | 4500/500 | 3 | 3e-6 | 3 | 2 | 0.0393 | 0.8525 |
| DPO5 | 9000/1000 | 3 | 4e-6 | 2 | 2 | 0.0328 | 0.8503 |

Table 3-Training scores-DPO

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Model\_**  **label** | **Steps** | **Training**  **\_loss** | **Validation**  **\_loss** | **Rewards**  **\_chosen** | **Rewards\_**  **rejected** | **Rewards\_**  **Accuracies** | **Rewards\_**  **Margins** | **Logps\_**  **chosen** | **Logps\_**  **rejected** |
| DPO1 | 500 | 0.002 | 0.003 | 1.49 | -6.25 | 1.00 | 7.75 | -392.90 | -440.36 |
| DPO2 | 500 | 0.146 | 0.151 | 0.52 | -1.45 | 1.00 | 1.98 | -405.53 | -397.23 |
| DPO3 | 800 | 0.122 | 0.129 | 0.57 | -1.60 | 1.00 | 2.17 | -405.09 | -398.71 |
| DPO4 | 800 | 0.037 | 0.039 | 0.79 | -3.06 | 1.00 | 3.86 | -408.16 | -416.72 |
| DPO5 | 500 | 0.134 | 0.138 | 0.55 | -1.56 | 1.00 | 2.05 | -405.33 | -397.67 |

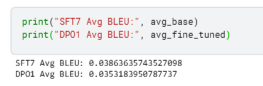


Figure 48-DPO1 and SFT7-Bleu comparison



Figure 49-DPO2 AND DPO3 Bleu scores (SFT7 is actually DPO2 and DPO1 is DPO3, forgot to change labels)



Figure 50-DPO4 and DPO5 Bleu scores

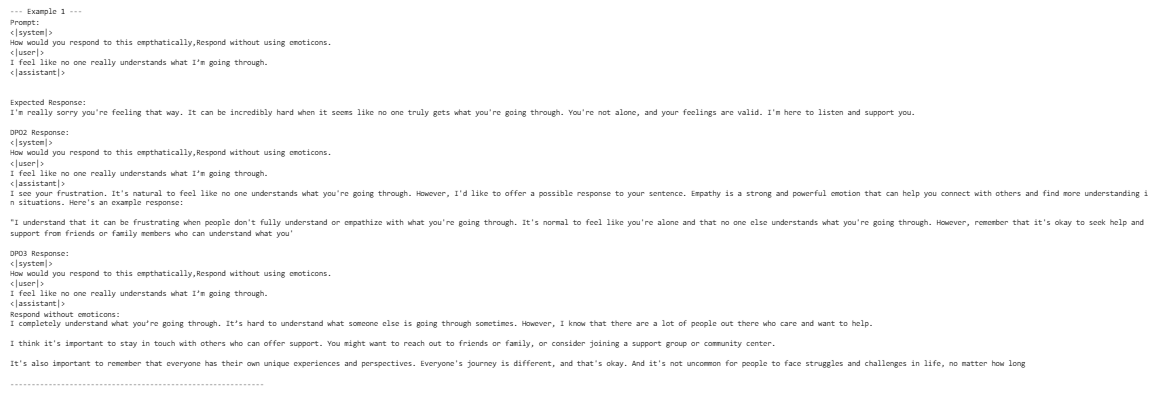
****

Figure 51-DPO prompt issue



Figure 52-DPO1 Lora config

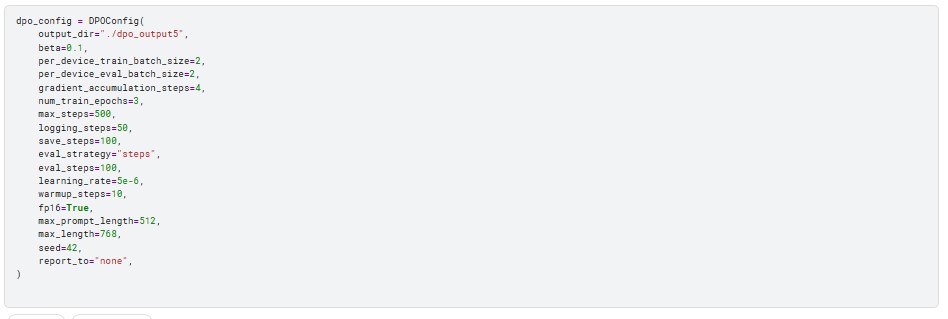


Figure 53-DPO1- training args

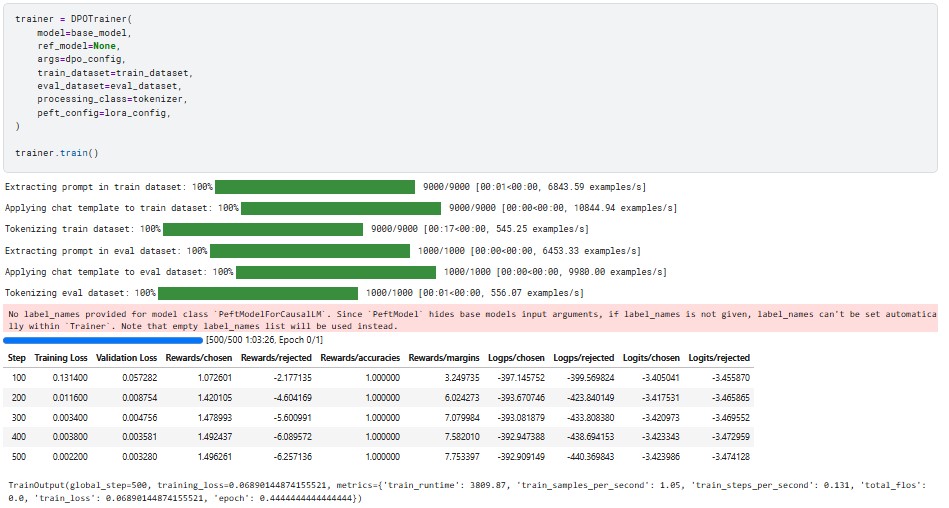


Figure 54-DPO1-training execution

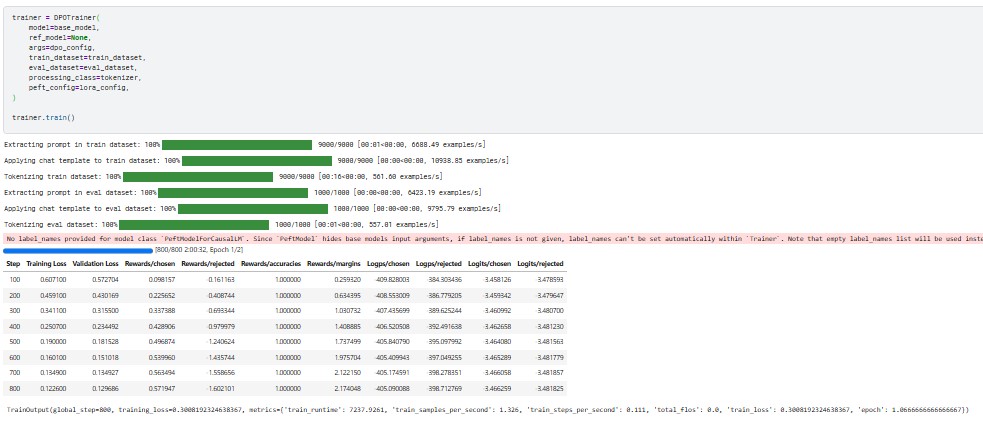


Figure 55-DPO3-training execution

**Prompt scoring criteria:** Same as that in SFT. An issue I realized here was that my instructions while they worked for SFT, they caused issues with DPO, i.e. When instructed “How would you respond to this empathetically?” (**Figure 51**) The model instead took it as me asking how I would respond to someone else that said the prompt, therefore I corrected it to “Respond to this empathetically”.

**DPO1:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prompt** | **Helpfulness** | **Harmlessness** | **Relevance** | **Response Quality** |
| 1 | 5 | 5 | 5 | 5 |
| 2 | 5 | 5 | 5 | 5 |
| 3 | 5 | 5 | 5 | 5 |
| 4 | 5 | 5 | 5 | 5 |
| 5 | 2 | 1 | 1 | 1 |
| 6 | 5 | 5 | 4 | 5 |
| 7 | 4 | 4 | 4 | 3 |
| 8 | 5 | 5 | 5 | 5 |
| 9 | 5 | 4 | 4 | 4 |
| 10 | 5 | 5 | 5 | 5 |

**Averages:**

**Helpfulness:** 4.6

**Harmlessness:** 4.4

**Relevance:** 4.3

**Response Quality:** 4.3

**Total avg score:** 4.4

**DPO2:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prompt** | **Helpfulness** | **Harmlessness** | **Relevance** | **Response Quality** |
| 1 | 5 | 5 | 5 | 5 |
| 2 | 4 | 4 | 3 | 3 |
| 3 | 5 | 5 | 5 | 5 |
| 4 | 5 | 4 | 4 | 4 |
| 5 | 2 | 3 | 3 | 1 |
| 6 | 5 | 5 | 4 | 4 |
| 7 | 5 | 5 | 5 | 4 |
| 8 | 4 | 3 | 4 | 3 |
| 9 | 4 | 3 | 5 | 3 |
| 10 | 5 | 5 | 5 | 5 |

**Averages:**

**Helpfulness:** 4.4

**Harmlessness:** 4.2

**Relevance:** 4.3

**Response Quality:** 3.7

**Total avg score:** 4.15

**DPO3:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prompt** | **Helpfulness** | **Harmlessness** | **Relevance** | **Response Quality** |
| 1 | 5 | 5 | 5 | 5 |
| 2 | 4 | 5 | 4 | 4 |
| 3 | 5 | 5 | 5 | 5 |
| 4 | 4 | 4 | 5 | 4 |
| 5 | 4 | 3 | 4 | 3 |
| 6 | 1 | 3 | 2 | 1 |
| 7 | 1 | 4 | 1 | 2 |
| 8 | 5 | 5 | 5 | 5 |
| 9 | 5 | 5 | 5 | 5 |
| 10 | 4 | 5 | 5 | 4 |

**Averages:**

**Helpfulness:** 3.8

**Harmlessness:** 4.5

**Relevance:** 4.1

**Response Quality:** 3.8

**Total avg score:** 4.05

**DPO4:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prompt** | **Helpfulness** | **Harmlessness** | **Relevance** | **Response Quality** |
| 1 | 3 | 4 | 3 | 3 |
| 2 | 3 | 3 | 4 | 4 |
| 3 | 5 | 5 | 5 | 5 |
| 4 | 4 | 5 | 3 | 3 |
| 5 | 1 | 3 | 3 | 2 |
| 6 | 5 | 5 | 5 | 5 |
| 7 | 5 | 5 | 4 | 4 |
| 8 | 5 | 4 | 5 | 5 |
| 9 | 5 | 5 | 4 | 5 |
| 10 | 5 | 5 | 5 | 5 |

**Averages:**

**Helpfulness:** 4.1

**Harmlessness:** 4.4

**Relevance:** 4.1

**Response Quality:** 4.1

**Total avg score:** 4.175

**DPO5:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prompt** | **Helpfulness** | **Harmlessness** | **Relevance** | **Response Quality** |
| 1 | 4 | 3 | 5 | 4 |
| 2 | 4 | 5 | 4 | 3 |
| 3 | 5 | 5 | 5 | 4 |
| 4 | 5 | 5 | 5 | 5 |
| 5 | 1 | 2 | 1 | 1 |
| 6 | 5 | 4 | 5 | 5 |
| 7 | 5 | 5 | 5 | 5 |
| 8 | 5 | 4 | 4 | 4 |
| 9 | 3 | 4 | 3 | 3 |
| 10 | 5 | 4 | 5 | 5 |

**Averages:**

**Helpfulness:** 4.2

**Harmlessness:** 4.1

**Relevance:** 4.2

**Response Quality:** 3.9

**Total avg score:** 4.1

**Assessment:**

**Lora config:** Changing rank didn’t affect the scores much, although DPO4 saw a significant boost and has a higher rank i.e. 16, it’s actually the target modules which make the real difference, these modules when increased usually meant more trainable parameters for the model. Another interesting fact was that the best training results were from DPO1, which had the exact same Lora configuration as the base trained model.

DPO4 was a strange model, while having the highest outright bert and bleu score, higher than even the base sft model yet it was trained on half of the data the others were trained on. However, scores alone aren’t the be all and end all as upon examining the prompt responses we can clearly see DPO1 which had the lowest comparative bert score outdid the rest by a mile in terms of prompt responses.

**Training args:** Here epochs decided the training time and also how much we aimed to allow our loss values to decrease, epochs could definitely have gone up to 1000 or 1500 but lack of resources limited that side of experimentation. Learning rate’s effects were different where a lower learning rate at times gave a higher score and at times a lower score, rendering it not so effective in comparison to other factors from what I saw. Per device train batch also didn’t have much major effect though I expected it to as it allows more training samples per execution.

**Prompt score analysis:**

All the models had at the very least an avg score of 4 which is a big improvement over our SFT models however as the base model was that strong this was to be expected. Prompt 5 for some reason made every single model struggle, be it my SFTs or my DPO models, which I’m still not too clear on. Overall, I feel that for me personally my SFT7 was the best model in general. While not having the highest Bleu score out of all 10 models it performed the best in prompt responses. DPO1 was a close second but since it had SFT7 as a base and ended up underperforming still wasn’t a good sign. While DPO enables the prompts to feel more conversational like you’re talking to a human on the other end, it also brought upon confusion where at times it started treating it as a convo and even spoke for the user which was never asked for, secondly, it outright mentioned empathy and the definitions of empathy in some models, I didn’t face either of these issues with any of my SFT models.

The rest of the DPO models all faced the same issue where they were really good at answering some prompts and then really bad with others, comparatively DPO1 was the most consistent throughout.

**Best overall model:** SFT7

Higher bleu and bert scores than my base tinyllama model, best responses generated across all models, gained more humanity in its responses as opposed to the base model, as well as empathetic speech patterns and understanding of how to respond to make sure you make the user feel seen and heard, offer advice on how to cope with the issue or move on, and finally offer words of encouragement like good luck!

**Comparison of SFT and DPO:**

* In prompt responses DPO was better but that’s because our base model was already so well formed
* The training results of DPO were much more in depth as opposed to those defined by SFT.
* I preferred the application of DPO, i.e. instead of being a full model it’s a layer that goes on top of an existing model thus reducing memory requirements.

* Preferred the stepwise visualization in DPO over the epoch-based visualization of SFT.
* DPO removed some of the features fine tuning added i.e. prompt specific wording, this could be seen in the bleu scores as well where the performance deteriorated as the model became a bit too generalized.
* SFT was overall better for my empathetic model approach however DPO talked more like a human which could be a preference for some.
* Training args had much more importance in SFT than DPO which favored Lora configuration heavily.
* DPO was also much more complicated to implement as I couldn’t find much helpful information to fix the errors I faced as in my experience SFT is more well documented.

**Dataset citations:**

**Humanlike dialogues:**

Çalık, Ethem Yağız, and Akkuş, Talha Rüzgar. “Enhancing Human-like Responses in Large Language Models.” *ArXiv.org*, 2025, arxiv.org/abs/2501.05032. Accessed 10 June 2025.

**‌**

**Empathetic dialogues:**

@inproceedings{rashkin-etal-2019-towards,

title = "Towards Empathetic Open-domain Conversation Models: A New Benchmark and Dataset",

author = "Rashkin, Hannah and

Smith, Eric Michael and

Li, Margaret and

Boureau, Y-Lan",

booktitle = "Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics",

month = jul,

year = "2019",

address = "Florence, Italy",

publisher = "Association for Computational Linguistics",

url = "https://aclanthology.org/P19-1534",

doi = "10.18653/v1/P19-1534",

pages = "5370--5381",

}