Visual Question and Answering Preference Alignment with ORPO and DPO

Maanu Grover
Department of Computer Science
Stanford University
maanug@stanford.edu

Gerardus (Guy) de Bruijn
Department of Computer Science
Stanford University
gdebruyn@stanford.edu

Abstract

In Visual Question Answering (VQA), models like GPT-4V have gained popularity for generating answers to questions based on images. For VQA models to be effective and production-ready, they must go through multiple stages of training. After pretraining, models undergo a Supervised Fine Tuning (SFT) stage to improve the quality of the answers and adapt to specific domains. However, these models often produce systematically incorrect answers with illogical justifications. Traditionally, a third stage called Reinforcement Learning with Human Feedback (RLHF) is introduced to further fine-tune using a reference model and rewards for chosen and rejected answers. A more recent approach, Direct Preference Optimization (DPO), avoids using rewards altogether and only looks at the likelihood of the chosen and rejected outputs. However, both these approaches require an initial SFT stage and a reference model for alignment. These factors make post-training expensive. To simplify this process, we experiment with a novel approach called Odds Ratio Preference Optimization (ORPO) which was recently introduced for Large Language Models (LLMs) to simultaneously learn to answer questions and align with human preferences without a reference model. ORPO eliminates the need for a third fine-tuning stage altogether by adding a loss penalty on top of the negative log likelihood penalty for predicted answers that poorly align with human preferences. Using the pre-trained blip-vqa-base model, we incorporate ORPO into the fine-tuning process and compare that with vanilla SFT (our baseline) and DPO plus SFT. We not only evaluate the model based on performance metrics Exact Match, F1 score and AI judge (evaluation by external LLM), but also look at the efficiency to train the model. Our results show that there is not a real difference in the performance between using ORPO and DPO. However, training with ORPO has been much more efficient. Overall training time for ORPO is much faster because it incorporates both the SFT domain adaption and preference alignment goals at the same time.

Figure 1. Log probabilities for both chosen and rejected responses during fine-tuning of the OPT-350M model on the HH-RLHF dataset. Rejected responses show a comparable probability of generation even though only chosen responses were used for supervision. Hong et al. (2024)

1. Introduction

In Visual Question and Answering (VQA), a model that has been trained on image and text generates answers to questions about the image. Studies of production VQA models like GPT-4V have observed systematic errors in answer correctness justified with illogical explanations. (Tong et al., 2024). For example, struggling to count the number of items in the image, failing to determine the orientation of the image subject, or mislabeling the colors or lighting in a scene. This variety of patterns and the consistency of these mistakes are a major shortcoming of these models.

It turns out that SFT alone to adapt to specific domains is not sufficient. Diagram 1 shows how, during supervised training, the likelihood that an undesirable response is generated increases alongside the desirable response, despite not being input as a training example.

To solve this problem, an additional step is performed to align these models with human preferences. Traditionally, models are trained to minimize cross-entropy loss in order to maximize the likelihood of generating the correct answer. To take into account human preferences, human-aware loss

functions (HALO) are introduced to train models to better align with objectives that are not captured by cross-entropy loss.

We aim to compare the effectiveness and efficiency of preference methods when applied to VQA models. To do this, we will finetune a VQA model on question-answering data and preferred and dis-preferred responses. We experiment with a novel method called Odds Ratio Preference Optimization (ORPO), and compare it with the more common Direct Preference Optimization (DPO) approach. We find that using ORPO can more quickly produce an aligned model compared to DPO, as ORPO combines alignment and SFT stages.

2. Related Work

As mentioned above, model alignment is an important step in improving the quality of model outputs to align with human preferences. Li et al. (2024) compares a few different approaches for model alignment on the VQA task, namely, DPO, SteerLM, and Rejection Sampling, using LLaVA as the base model. They found that DPO is the most effective approach, improving the model's performance on several visual-instruction benchmarks while maintaining performance on text instruction tasks which had previously degraded with visual instruction tuning.

Those approaches, however, are expensive for many reasons. All three require SFT to be performed beforehand. DPO needs an additional, albeit frozen, reference model to train the target model. SteerLM even requires training separate annotation models. This is limiting for anyone interested in training effective VQA models with limited access to compute resources.

Hong et al. (2024) suggests a new alignment strategy, ORPO, that combines the typical SFT and alignment stages and requires only the target model to train. They accomplish this by adding a term to the SFT loss that penalizes the likelihood of incorrect predictions. They note that the normal Cross-Entropy Loss function doesn't directly penalize the probabilities of incorrect tokens, i.e. tokens that are not the next in the sequence. The result is that the log probabilities of rejected tokens can increase during SFT along with those of correct tokens, as shown in figure 1. This is the basis for the additional loss term they introduce.

One final consideration for the VQA task is evaluation of responses. Most questions in datasets for this task are open-ended, so multiple answers may be semantically correct while not matching the ground-truth output in the dataset. Luo et al. (2022) recommends using multiple alternative answers, and evaluating the model output's semantic similarity to these alternative answers, as opposed to only checking if the output matches the ground truth. They find that this leads to more reasonable and fair model evaluations.

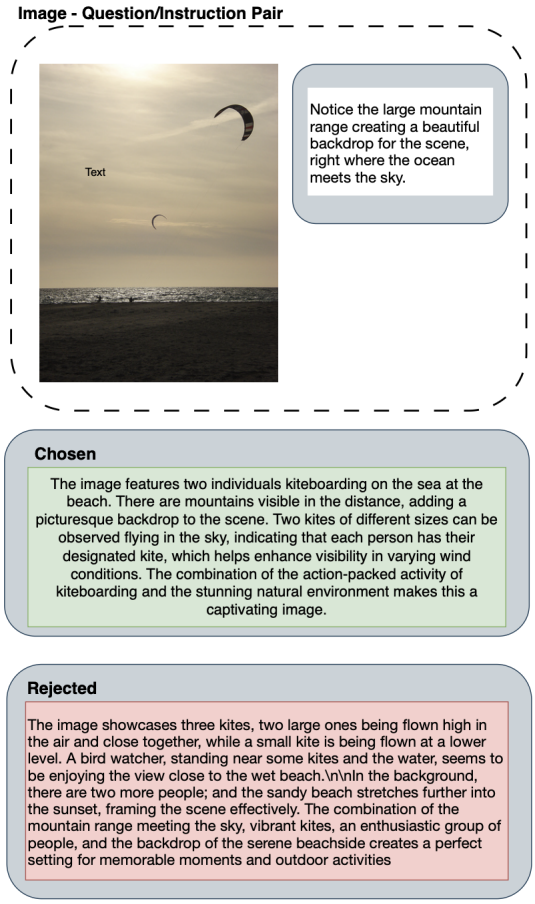

Figure 2. Example of a (image, question) pair with a chosen and rejected response.

3. Data

To train a VQA model, the dataset must include an image, question, and a target answer. For our preference optimization training objective, we require both a preferred answer and a rejected answer.

$$\mathcal{D} = \left\{ x^{(i)}, y_w^{(i)}, y_l^{(i)} \right\}_{i=1}^{N}$$

Here $x^{(i)}$ is a (image, question) pair, $y_w^{(i)}$ the preferred answer and $y_l^{(i)}$ the rejected answer.
We have found one HuggingFace dataset that uses this format and includes images. alexshengzhili/mllm-dpo, was used to train the LLaVA model with DPO. It contains almost five thousand examples. We have organized the referenced images, prompts, and answers from this dataset for our own training.

We have also incorporated a second dataset, MMInstruction/VLFeedback which has 80K examples. In this dataset, the answers have been labeled by various models on criteria such as helpfulness, visual faithfulness and ethical considerations. However, we have not been able to use this infor-

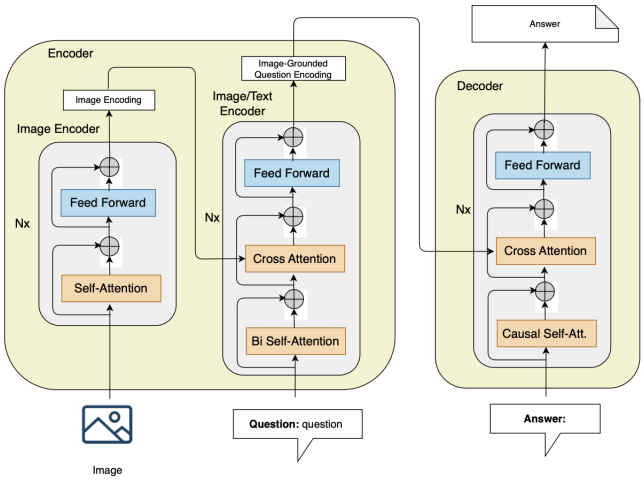

Figure 3. Multi modal Encoder-Decoder (MED): We use a Image-grounded text encoder to process the image and the question, and we use a image-grounded text decoder to generate the answer.

mation to come up with a chosen and rejected pair for each image and question pair. We use this dataset to compare the scores of vanilla SFT between the two data sets, and use it as an indication of the quality of the datasets.

4. Methods

We use a multi-modal encoder-decoder architecture as depicted in figure 3. An encoder transformer layer encodes the images. A second transformer layer encodes the question text and subsequently cross-attention is applied between the text encodings and image encodings to generate encodings of the text grounded in the image. The decoder then generates the output which is the predicted answer to the question about the image, again using cross-attention to ground the answer in the context of the image and question. We will only train the decoder. The encoder parameters remain frozen. For each method we use the following loss functions:

- Vanilla SFT: cross-entropy loss to maximize the likelihood generating the correct answer.
- ORPO: cross-entropy loss plus odds-ratio penalty between the positive (chosen) and negative (rejected) labels.
- DPO: loss based on the difference between how the currently trained model (policy model) and the reference model assign probabilities to the positive (chosen) and negative (rejected) labels.

4.1. DPO vs. RLHF

RLHF algorithms generally have two parts: a reward that is indicative of human preferences and the term for the Kullback–Leibler (KL) divergence constraint. The algorithm tries to maximize the reward, and the KL-divergence constraint is used to not trust the reward too much, i.e. to prevent reward hacking.

The RLHF objective can be formulated as

$$\max_{\pi_\theta} \mathbb{E}_{x\sim D, y\sim\pi_\theta(y|x)} \left[r_\phi(x,y)\right] - \beta \mathbb{D}_{KL}\left[\pi_\theta(y|x)||\pi_{\text{ref}}(y|x)\right]$$

Here β is a hyper-parameter controlling the deviation from the base reference policy π_{ref} which is the initially fine tuned SFT model. In practice, the MLLM policy π_θ is initialized with the SFT fine tuned π_{ref}.[1] Because the objective is not differentiable, it is typically fine tuned with reinforcement learning and optimized using PPO Schulman et al. (2017).

A big challenge is coming up with a scalar award that reflects how good a response is. Collecting pairwise preferences, though, is much easier to collect. [2] In the Bradley Terry model, the probability is estimated that a given pairwise preference is true. The probability distribution p^* is defined as follows:

$$p^*(y_w \succ y_l \mid x) = \frac{exp(r^*(x,y_w)}{exp(r^*(x,y_w)+exp(r^*(x,y_l)}$$

Here y_w is the chosen response and y_l the rejected response.

Using a static dataset sampled from p^* the reward model $r_\phi(x,y)$ can be parameterized and we can estimate the parameters via maximum likelihood. Now we have a binary classification problem and we can use the negative log-likelihood loss:

$$\mathcal{L}_R(r_\phi, D) = \\ -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log\sigma(r_\phi(x,y_w) - r_\phi(x,y_l))\right]$$

.

What we see here is that this model only depends on the *difference* between the rewards. With several mathematical steps, paper Rafailov et al. (2023) shows that the DPO objective can be derived as follows, without using rewards:

.

$$\mathcal{L}_{DPO}(\pi_\theta;\pi_{\text{ref}}) = \\ -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right]$$

Here π_θ is the policy model, the model currently being trained, and π_{ref} is the reference model.

The DPO loss function aims to optimize the policy

[1] Therefore, in our DPO implementation for VQA we will start with the same model to be fine-tuned as the SFT model that we use as the reference model.

[2] The analogy is similar to having two sports teams play each other, rather than trying to come up with a number how good each sports team is.

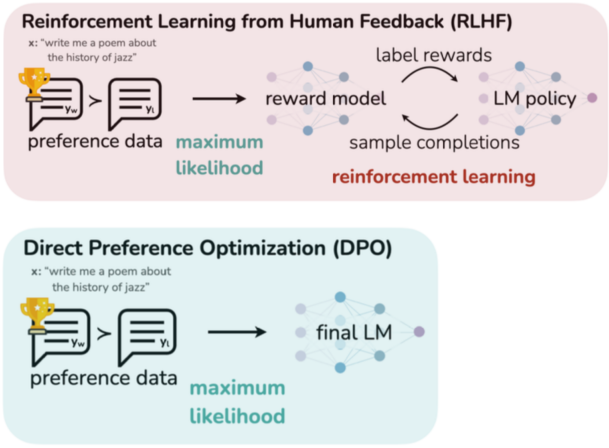

Figure 4. Per paper Rafailov et al. (2023): **DPO optimizes for human preferences while avoiding reinforcement learning.** Existing methods for fine-tuning language models with human feedback first fit a reward model to a dataset of prompts and human preferences over pairs of responses, and then use RL to find a policy that maximizes the learned reward. In contrast, DPO directly optimizes for the policy best satisfying the preferences with a simple classification objective, fitting an *implicit* reward model whose corresponding optimal policy can be extracted in closed form.

model by comparing the probabilities of the response chosen by this policy to those of the rejected response, in relation to a reference policy. β is a hyper parameter used to scale the comparison which aims to maximize the probability of the chosen response over the rejected response.

Because it directly optimizes for human preferences without the extra step of training an explicit reward model, DPO can more efficiently align models with the more intricate details of human preferences.

4.2. ORPO

While DPO eliminates the need for training a separate reward model, The Odds Ratio Preference Opimization (ORPO) training objective introduced by Hong et al. (2024) takes an even more efficient approach by combining finetuning and preference alignment stages. The ORPO objective function consists of the negative log-likelihood loss typically used in the SFT stage plus an additional term for penalizing undesirable outputs by maximizing the odds ratio of the likelihood of generating a desirable output versus an undesirable output. The term is defined as

$$\mathcal{L}_{OR} = -\lambda \log \sigma \left(\log \frac{\mathbf{odds}_\theta(y_w|x)}{\mathbf{odds}_\theta(y_l|x)} \right)$$

Where x is the input sequence, y_w is the desirable output sequence, y_l is the undesirable output sequence, and λ is a hyperparameter. The odds of a sequence y given a sequence

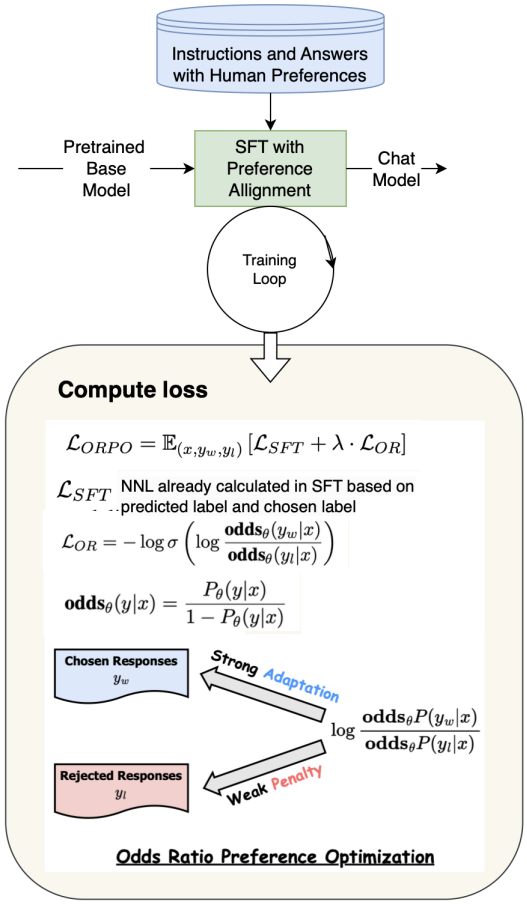

Figure 5. ORPO adds a penalty to the SFT loss during fine-tuning based on how much more likely a chosen response y_w is generated compared to a rejected response y_l. $\log P_\theta(y|x)$ is the log likelihood of generating output y given x. $\mathbf{odds}_\theta(y|x) = \frac{P_\theta(y|x)}{1-P_\theta(y|x)}$ indicates how much more likely it is that output y is generated compared to not generated.

x is defined as

$$\mathbf{odds}_\theta(y|x) = \frac{P_\theta(y|x)}{1 - P_\theta(y|x)}$$

As explained in the paper, when the odds ratio of desirable sequence to the undesirable sequence is high, the model is much more likely to generate the desirable sequence. Training with the loss term $\mathcal{L}_{OR}$ will maximize this odds ratio, which will increases the likelihood of favorable outputs and decreases the likelihood of unfavorable outputs, accomplishing the goal of preference alignment. Computing the odds of generating each sequence only relies on the target model being optimized. In other words, a reference model is not necessary as in DPO. As a result, SFT and ORPO preference alignment can be done in the same stage.

The inspiration for this additional term comes from in-

specting the cross-entropy loss.

$$\mathcal{L} = -\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{|V|} y_i^{(k)} \cdot \log(p_i^{(k)})$$

Normally, $y_i^{(k)}$ is binary with value 1 only for the correct next token of the sequence and 0 for the rest. Because of the softmax used to compute the probabilities from the logits, these log-probabilities have an indirect effect on each other in the backward pass. However, since $y_i^{(k)}$ is 0 for all incorrect tokens, the corresponding log-probabilities receive no direct penalty from the loss. This is the justification the authors of the ORPO method give for why the probabilities of rejected responses increase along with desired responses after Supervised Fine-Tuning.

The gradient of the additional term also suggests that ORPO will be effective for the purpose of preference alignment. From the paper:

$$\nabla_\theta \mathcal{L}_{OR} = \delta(d) \cdot h(d)$$

where,

$$\delta(d) = \left[1 + \frac{\textbf{odds}_\theta P(y_w|x)}{\textbf{odds}_\theta P(y_l|x)}\right]^{-1}$$

We can see that $\delta(d)$ will become smaller as the ratio of the odds of the preferred response to the dispreferred response increases. In other words, this gradient term will have a stronger effect when the model is just as likely to generate the dispreferred response. This is great property for the goal of preference alignment.

4.3. Implementation Details

For our experiments we use an existing encoder/decoder model that has been pre-trained for Visual Question And Answering: blip-vqa-base from Hugging Face. This open source BlipForQuestionAnswering model which was provided by Salesforce does not return logits that can be used to calculate the odds ratio or the log probabilities that we need to calculate the human-aware loss functions. We therefore have created our own custom model ORPOBlipForVQA which extends BlipForQuestionAnswering. A modified version of the forward() method was implemented where, as in the original forward method, the image embeddings and question embeddings are generated, as well output from the decoder for the predicted answer. We then also generate the output for the rejected (negative) answer. Loosely based on code from the original ORPO paper we use the logits of both predicted and rejected answer to calculate the ORPO ratio loss, and add that to the negative log likelihood loss multiplied by hyperparameter λ. Due to the way results from the policy model and reference model are combined to calculate the DPO loss we created a separate DPOBlipForVQA class that also extends BlipForQuestionAnswering [3]. For the DPO loss code, we took inspiration from a DPO implementation for a text-only LLM https://github.com/phymhan/llm-dpo.

See appendix A and B for details how we have implemented the ORPO and DPO loss for VQA. We found looking at an actual implementation helped our understanding of the math in the paper.

For the AI Judge we used an externally hosted model on together.ai, meta-llama/Llama-3-8b-chat-hf, which determines whether the predicted answer corresponds with the true answer for a given question. It returns a binary CORRECT/INCORRECT score for each example.

5. Evaluation and Experiments

5.1. Evaluation

For evaluation we use three different metrics:

- **Exact Match** Ratio of how often the predicted answer exactly matches the true answer.
- **F1** Average F1 score across evaluated examples. F1 is an indicator of the overlap of words between the true answer and the predicted answer with scores anywhere between 0 and 1.
- **AI Judge** Here we let an external LLM determine if the predicted answer is similar to the true answer for a given question. In our experiments, we use the QAEvalChain module from Langchain and the open source meta-llama/Llama-3-8b-chat-hf model to process a list of tuples (Question, True Answer, Predicted Answer). It returns a list of which ones are correct and which ones are incorrect. The LLM Evaluation score is the accuracy of our trained model, in other words the percent correct.

Training efficiency is an important goal of our experiments where we compare the different methods. Therefore, we use the exact same hyper parameters everywhere, like learning rate, batch size and the number of epochs. For method specific parameters we use the defaults as recommended by the original papers. We use vanilla SFT as a baseline, where we fine tune the BLIP decoder using the mllm-dpo dataset. The training and evaluation datasets are created with the following structure:

1. Tokenized image and text, generated from the BLIP processor encoder. As text the first completion response is used in each training example.

[3] A future version would have one class that implements all the various human aware loss functions, rather than having to deal with one class for each.

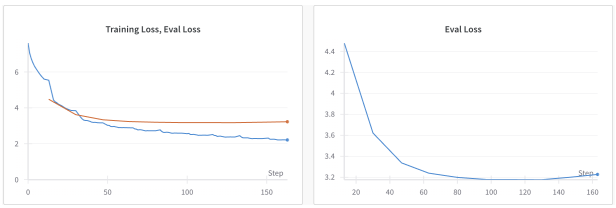

Figure 6. Vanilla SFT training loss and evaluation loss over 10 epochs on dataset MLLM-DPO.

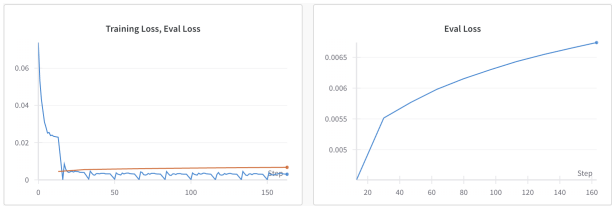

Figure 7. DPO training loss and evaluation loss with beta=0.5 over 10 epochs on dataset MLLM-DPO.

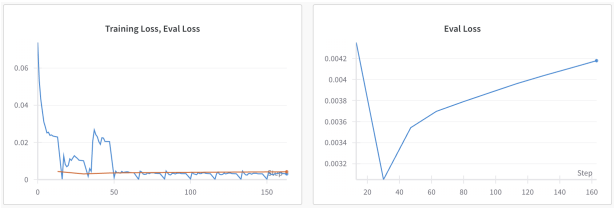

Figure 8. DPO training loss and evaluation loss with beta=0.1 over 10 epochs on dataset MLLM-DPO.

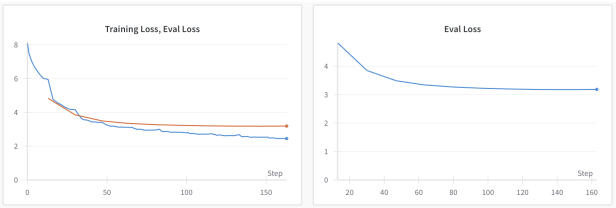

Figure 9. ORPO training loss and evaluation loss with gamma=0.5 over 10 epochs on dataset MLLM-DPO.

2. Attention mask generated by the BLIP processor.
3. Output encodings generated by the BLIP processor for the chosen (positive, true) answer and the rejected (negative) answer.

The final scores for each model will come from the checkpoint that had the lowest evaluation loss. Because of the fair comparison regarding efficiency, hyper parameter tuning is out of scope and hence a final validation on a test set has been deferred.

5.2. Results

In all our training experiments, our model uses the default AdamW optimizer from PyTorch. We have used the PyTorch scheduler to exponentially decrease the learning rate after each epoch, with gamma=0.9. The AI Judge score, where the external LLM decides if the predicted answer corresponds with the true answer, is only sampled over about 100 examples from the evaluation set. The reason for this reduced size is because we ran into maximum usage quota for the externally hosted service on together.ai.

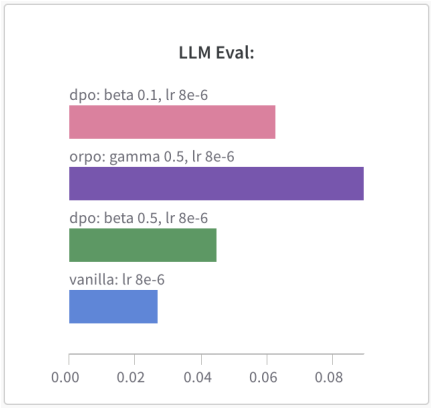

Figure 10. Evaluation from AI Judge.

5.2.1 Vanilla SFT, ORPO and DPO training on MLLM-DPO dataset

- All experiments were run on a single NVIDIA A6000 Ada Generation GPU.
- All three methods were trained over 10 epochs with a learning rate of 8e-6 and batch size 16.

Table 1 shows the scores and the training time per epoch for each method. For the SFT + DPO method, only the additional training time for DPO is shown, meaning, the SFT part is not included.

Some experimental details about each method:

1. Vanilla SFT: the lowest evaluation score was reached after eight epochs.
2. SFT + DPO: the recommended range for beta is between 0.1 and 0.5. Therefore, we ran two experiments: one for beta=0.1 and one for beta=0.5. DPO fine tuning uses the saved model from vanilla SFT as the start. After already 1 or 2 epochs, the lowest evaluation score was reached.
3. ORPO: For training we used lambda=0.5, which is the same value as used in the original ORPO paper Hong et al. (2024).

Figure 10 shows the LLM Evaluation score by the AI Judge. All methods with alignment clearly show a better score than vanilla SFT. But we have to note that only 100

Method	Exact Match	F1	AI Judge	Epoch Time (hr:min)
Vanilla SFT 7	0.0	0.147	0.026	0:02:06
SFT+DPO beta=0.5 7	0.0	0.159	0.044	0:03:09
SFT+DPO beta=0.1 8	0.0	0.16	0.0625	0:03:09
ORPO 9	0.0	0.151	0.089	0:02:21

Table 1. Experimental results for three fine tune methods, with links to corresponding training and evaluation graphs.

Epoch	Exact Match	F1	AI Judge	Epoch Time (hr:min)
1	0.003	0.216	0.31	3:30
2	0.003	0.225	0.23	3:29
3	0.004	.229	0.28	3:29
4	0.002	0.231	0.34	3:29

Table 2. Vanilla SFT Fine tuning the MMInstruction/VLFeedback dataset

samples have been sent over due to api credit limitations and therefore the result may not be representative of the dataset.

5.2.2 Vanilla SFT training on MMInstruction/VLFeedback

We also performed vanilla SFT fine tuning on the MMInstruction/VLFeedback dataset. This dataset is much larger (80K examples) but does not have chosen-rejected pairs that can be used to train on preference alignment.

- We trained this dataset on EC2 instance type g4dn.xlarge on AWS which has an NVIDIA Tesla T4 with 1 GPU and 16GB of memory.
- The training was done over 10 epochs with a learning rate of 8e-6 and batch size 16.

Table 2 shows the evaluation scores after each epoch of training. Exact Match and F1 are taken over the entire evaluation set.

Diagram 11 shows the results of the training loss and evaluation loss over the four epochs that MMInstruction/VLFeedback has been trained. An earlier experiment, with a slightly higher learning rate of 1e-5, and over five epochs, showed a steady increase of the evaluation loss after the third epoch.

5.3. Results Discussion

Fine tuning one epoch for DPO takes 35 percent longer than one epoch for ORPO. This comes on top of the 10 epochs SFT stage that ORPO does not need, because ORPO

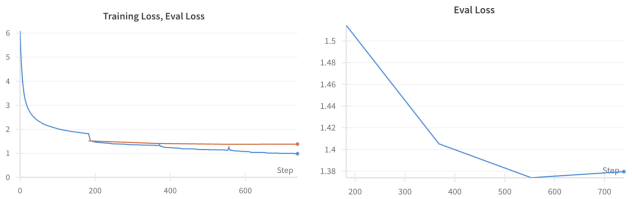

Figure 11. Training loss and evaluation loss over four epochs on dataset MMInstruction/VLFeedback

combines both. Even with incorporating the cross entropy loss calculation used in SFT, ORPO is only about 10 percent slower than the vanilla SFT baseline. This additional 10 percent is required for calculating the log probabilities for the negative response, which is not required for vanilla SFT.

None of the methods were able to come up with a prediction that had an exact match. The F1 score remained in the 0.15-0.16 range for all methods. However, the alignment methods showed better result on the AI Judge metric compared to vanilla SFT. Especially ORPO showed a better performance on this metric.

Low evaluation scores, in SFT as well as the following alignment fine tuning, could indicate deficiencies in the model. However, we see that the VLFeedback dataset performed significantly better on all metrics after vanilla SFT fine tuning. Because of the lack of a rejected response this dataset could not be fine tuned on the other methods.

6. Conclusion

The MLLM-DPO dataset is probably too complex and nuanced for our BLIP model. The questions and instructions in the dataset are open ended, with small nuances between the chosen and rejected answer. Figure 2 mentioned in the Data section is one of the more simpler examples we could find in the dataset. See appendix C for a more sophisticated example of a physics graph and corresponding question and answers. Properly fine tuning on this sophisticated dataset will likely require a model with billions of parameters, while our model has less than 400 million parameters. F1 and AI Judge scores remain low at about 0.15 and below 0.1 respectively. The MMInstruction/VLFeedback dataset that was only used for SFT fine tuning because of lack of preference data shows much higher scores for both metrics, around 0.3 for each.

Training SFT + DPO is more complex to manage because it involves handling two stages instead of just one stage in case of ORPO. Overall training time is also more efficient for ORPO than DPO because of the separate SFT stage for DPO. And besides the forward pass for the model being trained (policy model), the DPO alignment stage requires an extra forward pass for the reference model. Lastly, the DPO model has a larger memory footprint because

Figure 12. With ORPO, SFT fine-tuning and preference alignment are accomplished all at the same time

it needs to keep the parameters of both the model being trained and the reference model in memory. At least there is no backward pass for the reference model so results of the forward pass do not need to be kept in memory.[4] In summary, with ORPO we accomplish in one stage both goals: SFT fine tuning and preference alignment. This makes the combined domain adaptation and preference alignment much more efficient for ORPO than for DPO. 12

7. Limitations and Future Work

Using ORPO alone requires a sufficiently large preference dataset, at least as large as the SFT dataset you might use instead. For VQA this is very difficult to find. On the other hand, there are many datasets with chosen - rejected pairs for just LLM fine tuning.

We need a dataset with simpler questions and chosen - rejected answer pairs to fine tune a model that only has a few hundred million parameters. One option is to take an existing dataset that has triplets (image, question, answer) and use an external LLM to generate a different answer that is less preferred, and then add it to the original dataset.

So far we have only compared ORPO to DPO which is at the moment the most popular method for preference alignment. A challenge for methods such as ORPO and DPO is that they all require datasets that include rejected answers. These datasets are difficult to generate. Another method, Kahneman-Tversky Optimization (KTO), described in Ethayarajh et al. (2024), eliminates the need for chosen - rejected pairs altogether. This method only quires a binary signal of whether an output is desirable or undesirable for a given input. This information can be incorporated in the human-aware loss function.

Due to api limitations, we could only use a limited (one hundred) examples to validate against and external LLM model to determine if the predicted response is similar to the true answer, given the question. A carefully created script, that at times pauses to not go over the rate limit, could be used on a larger final test set.

Hyper parameter tuning for each of the methods, and then use it on the final test set, can give a more accurate picture of which method has the potential for the best performance. The model was too complex and compute resources too limited to do this for all three methods. In the future we hope to do this for a more suitable dataset which should give a better indication of the performance of the method.

Another evaluation metric we can use is win rate, which also takes into account preferences. For text-only LLM evaluation, various models have already been designed such as Alpaca, Dubois et al. (2024).

[4] When using LoRA or other Parameter Efficient Fine Tuning method, for DPO the parameters of the policy and the reference model can be shared. Only the policy model then uses the LoRA parameters.

References

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. 2024. Length-controlled alpacaeval: A simple way to debias automatic evaluators.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. Kto: Model alignment as prospect theoretic optimization.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. Orpo: Monolithic preference optimization without reference model.

Shengzhi Li, Rongyu Lin, and Shichao Pei. 2024. Multi-modal preference alignment remedies regression of visual instruction tuning on language model.

Man Luo, Shailaja Keyur Sampat, Riley Tallman, Yankai Zeng, Manuha Vancha, Akarshan Sajja, and Chitta Baral. 2022. 'just because you are right, doesn't mean i am wrong': Overcoming a bottleneck in the development and evaluation of open-ended visual question answering (vqa) tasks.

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.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms.

Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. 2024. Eyes wide shut? exploring the visual shortcomings of multimodal llms.

A. Source code BlipForQuestionAnswering for ORPO

Below code is based on open source https://github.com/huggingface/transformers/tree/main/src/transformers/models/blip . Class BlipForQuestionAnswering has been extended to include the ORPO loss, inspired by the ORPO implementation for text-only LLMs in the original ORPO paper: https://github.com/xfactlab/orpo/tree/main.

```
from transformers import BlipForQuestionAnswering
from dataclasses import dataclass
from typing import Any, Optional, Tuple, Union
import torch
import torch.nn.functional as F

class ORPOConfig:
    def __init__(self, orpo_lambda):
        self.orpo_lambda = orpo_lambda

@dataclass
class ORPOBlipTextVisionModelOutput():
    loss: Optional[torch.FloatTensor] = None
    logits: Optional[torch.FloatTensor] = None
    image_embeds: Optional[torch.FloatTensor] = None
    last_hidden_state: torch.FloatTensor = None
    hidden_states: Optional[Tuple[torch.FloatTensor, ...]] = None
    attentions: Optional[Tuple[torch.FloatTensor, ...]] = None


class ORPOBlipForVQA(BlipForQuestionAnswering):
    def __init__(self, config, orpo_config=None):
        super().__init__(config)
        self.orpo_config = orpo_config

    def _calculate_logps(self, logits: torch.FloatTensor, labels: torch.LongTensor,
                attention_mask: torch.LongTensor=None):
        assert logits.shape[:-1] == labels.shape

        labels = labels[:, 1:].clone()
        logits = logits[:, :-1, :]
        loss_mask = (labels != -100)

        labels[labels == -100] = 0

        per_token_logps = torch.gather(logits.log_softmax(-1), dim=2,
                            index=labels.unsqueeze(2)).squeeze(2)

        return (per_token_logps * loss_mask).sum(-1)

    def calculate_orpo_loss(self, outputs_pos, outputs_neg, positive_labels, negative_labels,
            attention_mask):
        #check self.tokenizer.pad_token_id

        orpo_lambda=self.orpo_config.orpo_lambda

        pos_loss = outputs_pos.loss

        pos_prob = self._calculate_logps(outputs_pos.logits, positive_labels, attention_mask)

        neg_prob = self._calculate_logps(outputs_neg.logits, negative_labels, attention_mask)

        # Calculate log odds
```

```
    log_odds = (pos_prob - neg_prob) - (torch.log(1 - torch.exp(pos_prob))
                                        - torch.log(1 - torch.exp(neg_prob)))

    sig_ratio = torch.nn.functional.sigmoid(log_odds)
    ratio = torch.log(sig_ratio)

    # Calculate the Final Loss
    loss = torch.mean(pos_loss - orpo_lambda * ratio)
    return loss

def forward(
        self,
        input_ids: torch.LongTensor,
        pixel_values: torch.FloatTensor,
        decoder_input_ids: Optional[torch.LongTensor] = None,
        decoder_attention_mask: Optional[torch.LongTensor] = None,
        attention_mask: Optional[torch.LongTensor] = None,
        output_attentions: Optional[bool] = None,
        output_hidden_states: Optional[bool] = None,
        labels: Optional[torch.LongTensor] = None,
        neg_labels: Optional[torch.LongTensor] = None,
        return_dict: Optional[bool] = None
    ):

    vision_outputs = self.vision_model(
        pixel_values=pixel_values,
        output_attentions=output_attentions,
        output_hidden_states=output_hidden_states,
        return_dict=return_dict
    )

    image_embeds = vision_outputs[0]
    image_attention_mask = torch.ones(image_embeds.size()[:-1], dtype=torch.long)

    question_embeds = self.text_encoder(
        input_ids=input_ids,
        attention_mask=attention_mask,
        encoder_hidden_states=image_embeds,
        encoder_attention_mask=image_attention_mask,
        return_dict=return_dict,
    )

    if labels is not None and decoder_input_ids is None:
        # labels are already shifted right, see: https://github.com/huggingface/transformers/pull/231
        decoder_input_ids = labels

    question_embeds = question_embeds[0] if not return_dict
                                         else question_embeds.last_hidden_state

    answer_output = self.text_decoder(
        input_ids=decoder_input_ids,
        attention_mask=decoder_attention_mask,
        encoder_hidden_states=question_embeds,
        encoder_attention_mask=attention_mask,
        labels=labels, #positive labels
        return_dict=return_dict,
        reduction="mean",
    )
```

```
        if labels is not None:
            if self.orpo_config == None:
                decoder_loss = answer_output.loss.mean() if return_dict
                                                 else answer_output[0].mean()
            else:
                neg_answer_output = self.text_decoder(
                                        input_ids=decoder_input_ids,
                                        attention_mask=decoder_attention_mask,
                                        encoder_hidden_states=question_embeds,
                                        encoder_attention_mask=attention_mask,
                                        labels=neg_labels,
                                        return_dict=return_dict,
                                        reduction="mean"
                                    )
                decoder_loss = self.calculate_orpo_loss(outputs_pos=answer_output,
                                                        outputs_neg=neg_answer_output,
                                                        positive_labels = labels,
                                                        negative_labels = neg_labels,
                                                        attention_mask=attention_mask
                                                        )
        else:
            decoder_loss = None

        return ORPOBlipTextVisionModelOutput(
            loss=decoder_loss,
            image_embeds=image_embeds,
            last_hidden_state=vision_outputs.last_hidden_state,
            hidden_states=vision_outputs.hidden_states,
            attentions=vision_outputs.attentions,
        )
```

B. Source code BlipForQuestionAnswering for DPO

Below code is based on open source https://github.com/huggingface/transformers/tree/main/src/transformers/models/blip. Class BlipForQuestionAnswering has been extended to include the DPO loss, inspired by one of the DPO implementations for text-only LLMs: https://github.com/phymhan/llm-dpo.

```
class DPOConfig:
    def __init__(self, dpo_beta):
        self.beta = dpo_beta

@dataclass
class DPOBlipTextVisionModelOutput():
    loss: Optional[torch.FloatTensor] = None
    positive_logits: Optional[torch.FloatTensor] = None
    negative_logits: Optional[torch.FloatTensor] = None


class DPOBlipForVQA(BlipForQuestionAnswering):
    def __init__(self, config, dpo_config=None):
        super().__init__(config)
        self.dpo_config = dpo_config

    def _calculate_logps(self, logits: torch.FloatTensor, labels: torch.LongTensor,
                         attention_mask: torch.LongTensor=None):
```

```
    assert logits.shape[:-1] == labels.shape

    labels = labels[:, 1:].clone()
    logits = logits[:, :-1, :]
    loss_mask = (labels != -100)

    labels[labels == -100] = 0

    per_token_logps = torch.gather(logits.log_softmax(-1), dim=2,
                           index=labels.unsqueeze(2)).squeeze(2)

    return (per_token_logps * loss_mask).sum(-1)

def forward(
        self,
        input_ids: torch.LongTensor,
        pixel_values: torch.FloatTensor,
        decoder_input_ids: Optional[torch.LongTensor] = None,
        decoder_attention_mask: Optional[torch.LongTensor] = None,
        attention_mask: Optional[torch.LongTensor] = None,
        output_attentions: Optional[bool] = None,
        output_hidden_states: Optional[bool] = None,
        labels: Optional[torch.LongTensor] = None,
        neg_labels: Optional[torch.LongTensor] = None,
        return_dict: Optional[bool] = None,
        dpo_config: Optional[DPOConfig] = None
    ):

    vision_outputs = self.vision_model(
        pixel_values=pixel_values,
        output_attentions=output_attentions,
        output_hidden_states=output_hidden_states,
        return_dict=return_dict
    )

    image_embeds = vision_outputs[0]
    image_attention_mask = torch.ones(image_embeds.size()[:-1], dtype=torch.long)

    question_embeds = self.text_encoder(
        input_ids=input_ids,
        attention_mask=attention_mask,
        encoder_hidden_states=image_embeds,
        encoder_attention_mask=image_attention_mask,
        return_dict=return_dict,
    )

    if labels is not None and decoder_input_ids is None:
        decoder_input_ids = labels

    question_embeds = question_embeds[0] if not return_dict
                                    else question_embeds.last_hidden_state

    answer_output = self.text_decoder(
```

```
            input_ids=decoder_input_ids,
            attention_mask=decoder_attention_mask,
            encoder_hidden_states=question_embeds,
            encoder_attention_mask=attention_mask,
            labels=labels, #positive labels
            return_dict=return_dict,
            reduction="mean",
        )

        if labels is not None:
            decoder_loss = answer_output.loss.mean() if return_dict
                                    else answer_output[0].mean()

            neg_answer_output = self.text_decoder(
                                        input_ids=decoder_input_ids,
                                        attention_mask=decoder_attention_mask,
                                        encoder_hidden_states=question_embeds,
                                        encoder_attention_mask=attention_mask,
                                        labels=neg_labels,
                                        return_dict=return_dict,
                                        reduction="mean"
                            )
        else:
            decoder_loss = None

        #Return log probabilities for chosen and rejected
        return DPOBlipTextVisionModelOutput(
            loss=decoder_loss,  #loss is from positive anwer
            positive_logits=answer_output.logits,
            negative_logits=neg_answer_output.logits,
        )

    def calculate_dpo_loss(self, policy_outputs, reference_outputs, positive_labels,
                           negative_labels, attention_mask):

        policy_chosen_logps = self._calculate_logps(policy_outputs.positive_logits,
                                positive_labels, attention_mask)

        policy_rejected_logps = self._calculate_logps(policy_outputs.negative_logits,
                                negative_labels, attention_mask)

        reference_chosen_logps = self._calculate_logps(reference_outputs.positive_logits,
                                positive_labels, attention_mask)

        reference_rejected_logps = self._calculate_logps(reference_outputs.negative_logits,
                                negative_labels, attention_mask)

        pi_logratios = policy_chosen_logps - policy_rejected_logps
        ref_logratios = reference_chosen_logps - reference_rejected_logps

        logits = pi_logratios - ref_logratios

        losses = -F.logsigmoid(self.dpo_config.beta * logits)
```

```
loss = losses.mean()

return loss
```

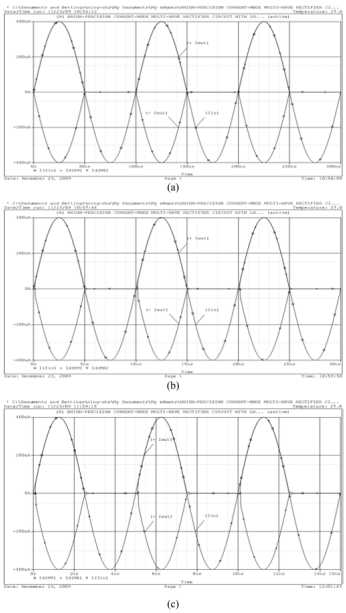

Figure 13. Example image from Sci Graph dataset

C. Example of MLLM-DPO dataset

- Image 13 is from the MLLM-DPO data set, generated from the original sci graph collection.
- Question: What is the significance of the half-wave output signal in Figure 4?
- Chosen answer: The half-wave output signal in figure 4 is significant as it demonstrates the rectification process and verifies the efficiency of the multiwave rectifier circuit in operation. In the diagram, the half-wave current signal passes through the comparators (Q9 and Q10) and then connects to a CMOS inverter (Q8), all located within the current mirror circuit. The inverter is designed to convert this analog half-wave signal into another half-wave signal of the same form, but alternating polarity, with no additional noise or information loss. This confirmation process is critical to ensuring the functional integrity of the multiwave rectifier, ultimately contributing to the design and optimization of low-voltage current mode rectification system.
- Rejected answer: The half-wave output signal shown in Figure 4 is an important part of the proposed rectifier circuit's functioning. It indicates the circuit's performance and capability to generate the output signal in accordance with the input signal. Additionally, this output signal provides insights into the efficiency and precision of the multiwave rectifier in processing the input signal under varying low voltage conditions. The test results help in evaluating the circuit's effectiveness in handling different input and load situations, as the half-wave output signal will have specific characteristics and shape unique to the multiwave rectifier configuration.