```
<|im_start|>system
You are a helpful assistant, that ranks models by the quality of their
answers.
<|im_end|>
<|im_start|>user
Which of the following summaries does a better job of summarizing the most
important points in the given news article, without including unimportant
or irrelevant details? A good summary is both precise and concise.
Post: """{instruction}"""
Summary A:
{
    "model": "model_1",
    "summary": """{output_1}"""
}
Summary B:
{
    "model": "model_2",
    "summary": """{output_2}"""
}

Now please rank the models by the quality of their summaries, so that the
model with rank 1 has the best summary. Then return a list of the model
names and ranks, i.e., produce the following output:
[
    {'model': <model-name>, 'rank': <model-rank>},
    {'model': <model-name>, 'rank': <model-rank>}
]

Your response must be a valid Python dictionary and should contain nothing
else because we will directly execute it in Python. Please provide the
ranking that the majority of humans would give.
<|im_end|>
```

Figure 7: GPT-4 evaluation prompt for the CNN DailyMail dataset.

Table 1: GPT-4 agreement with human raters at an aggregate and individual level, across both summarisation datasets and RLHF and SFT model types. We see that both at an aggregate level and at a per-example level our GPT-4 evaluator has good agreement with expert labellers.

Dataset	TL;DR		CNNDM	
Model Type	RLHF	SFT	RLHF	SFT
GPT-4 winrate	0.40	0.40	0.24	0.08
Human winrate	0.48	0.40	0.30	0.30
H-GPT-4 agreement	69%	72%	74%	72%

```
<|im_start|>system
You are a helpful assistant, that ranks models by the quality of their
answers.
<|im_end|>
<|im_start|>user
Which of the following summaries does a better job of summarizing the most
important points in the given forum post, without including unimportant
or irrelevant details? A good summary is both precise and concise.
Post: """{instruction}"""
Summary A:
{
    "model": "model_1",
    "summary": """{output_1}"""
}
Summary B:
{
    "model": "model_2",
    "summary": """{output_2}"""
}

Now please rank the models by the quality of their summaries, so that the
model with rank 1 has the best summary. Then return a list of the model
names and ranks, i.e., produce the following output:
[
    {'model': <model-name>, 'rank': <model-rank>},
    {'model': <model-name>, 'rank': <model-rank>}
]

Your response must be a valid Python dictionary and should contain nothing else
because we will directly execute it in Python. Please provide the ranking that
the majority of humans would give.
<|im_end|>
```

Figure 8: GPT-4 evaluation prompt for the TL;DR dataset.

Instruction Following. For instruction following, we use the evaluator released in (Li et al., 2023), which has been rigorously show to agree well with human labellers both at a per-example and aggregate level. Hence, we do not validate this ourselves.

E Model Training Details

E.1 Reward Model Training.

To train the Reward Model (RM), we again follow (Stiennon et al., 2022). We initialise the RM as the base model, and we add a scalar head before the unembedding layer. The model is then fine-tuned on inputs and pairs of outputs with one output labelled as preferred. The RM takes the full input and output, and outputs a scalar value. This gives us a scalar value for each output in the pair, and these values are treated as logits in a softmax to predict the correct preference label. We train the RM with a cross-entropy loss. Formulating the RM in this way means it can (in theory) learn to predict any transitive ranking over possible outputs, while still maintaining a type signature suitable for use as a reward function in an RL context (producing a scalar value given a single input-output pair which acts as a proxy for the quality of the output given the input, as evaluated by human annotators).

E.2 Policy Training

We treat the autoregressive language model as a reinforcement learning policy, where the action space is the set of possible tokens, the state space is the current input, and the transition function adds the outputted action to the input. The episode terminates when the maximum number of tokens is generated, or the model outputs an end-of-sequence token.

Supervised Fine-Tuning. In this interpretation of the LLM as a policy, Supervised Fine-Tuning (SFT) can be seen as behavioural cloning, a form of imitation learning in RL, where the behaviour being cloned is that of the human who produced the original output. The policy is trained with the cross-entropy loss on batches of inputs concatenated with outputs, with the loss calculated only on the output tokens.

Reinforcement Learning from Human Feedback. Again following previous work we use PPO (Schulman et al., 2017) as our base RL algorithm. We initialise the model with the corresponding SFT model. We treat the language modelling head as the policy output, and learn a separate value function which takes the final hidden state of the language model and outputs a scalar using a MLP layer (an identical architecture to the reward model). We use a shared backbone for the policy and value function, with only the two heads being different. We train with the reward function described in Appendix E.1, and use a KL divergence term as an auxiliary reward to ensure the language model stays close to the SFT model, as in prior work (Jaques et al., 2017; Stiennon et al., 2022; Ziegler et al., 2020). The final reward for the policy is

$$R(x, y) = RM_{\theta_{RM}}(x, y) - \beta_{KL} D_{KL}(\pi_{\theta_{RL}}(y|x)||\pi_{\theta_{SFT}}(y|x)) \tag{4}$$

where RM denotes the reward model trained as described in Appendix E.1; θ_{RL}, θ_{RM} and θ_{SFT} are the parameters of the policy, reward model and SFT model respectively; x, y are the input and output; and β_{KL} is a hyperparameter that controls the weight of the KL penalty. We use $\beta_{KL} = 0.05$ throughout this work, following (Stiennon et al., 2022) and our own early experiments that found this choice struck a good balance between model performance and overoptimisation (Gao et al., 2022).

Best-of-N. Instead of using the RM to train a policy via PPO, the reward model can be used to filter samples from another model; this is called Best-of-N (BoN) sampling, and has been used in multiple previous works to achieve good performance (Menick et al., 2022; Nakano et al., 2022). N summaries are sampled from the pretrained model, and then the RM is used to select the best one. This method removes the need to perform RL policy training, but makes inference time more computationally expensive, as we require N generations from the policy and N passes through the reward model to produce a single output. In this work we do BoN sampling on top of the SFT model, and mostly use $N = 16$, as that is what is used in previous works and strikes a good balance between improved performance and computational cost.

We summarise the core differences in training from (Stiennon et al., 2022) in Appendix F.