ORPO: Odds Ratio Preference Optimization

Monolithic Preference Optimization without Reference Model

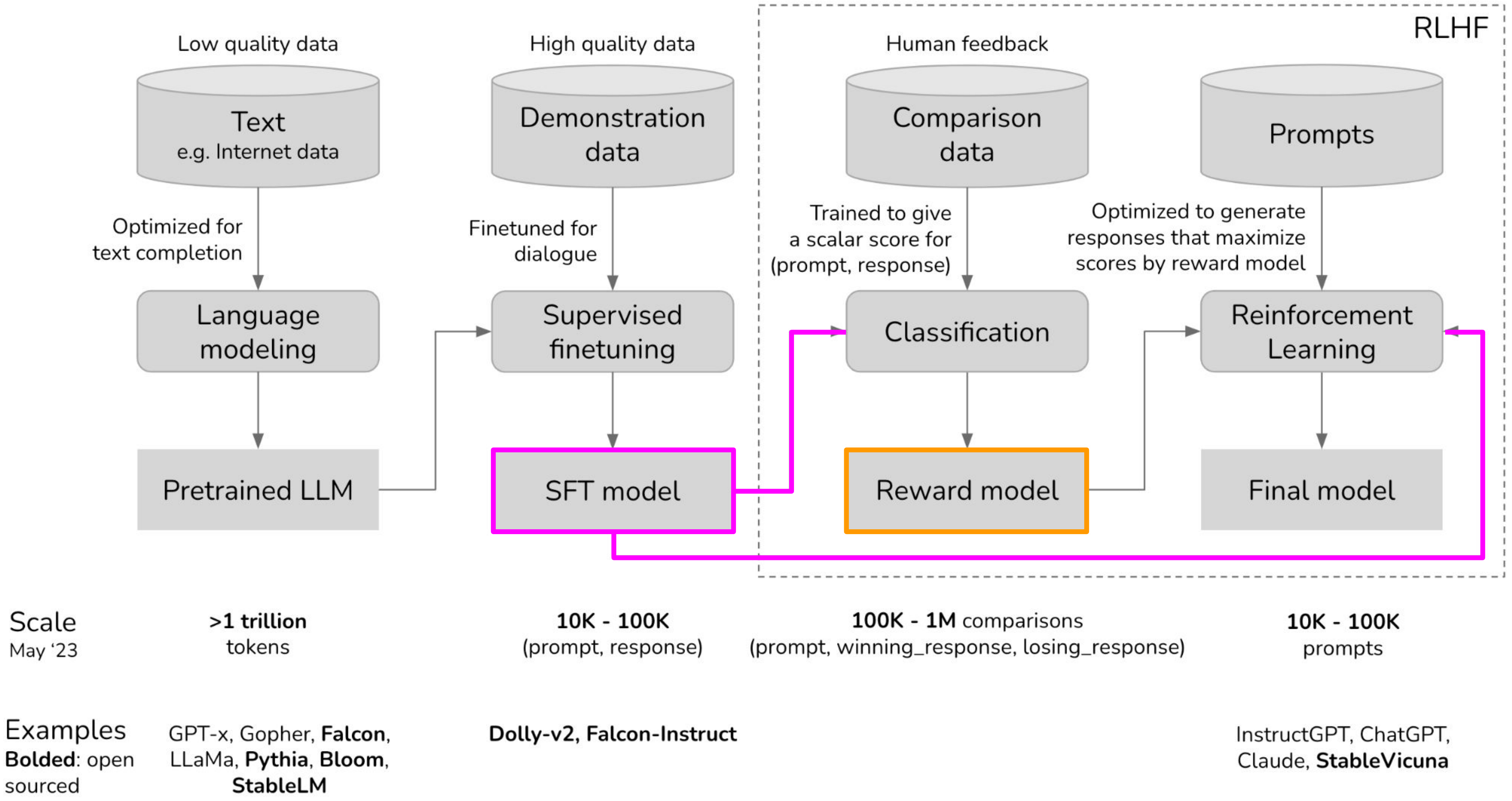
@vishal_learner

Outline

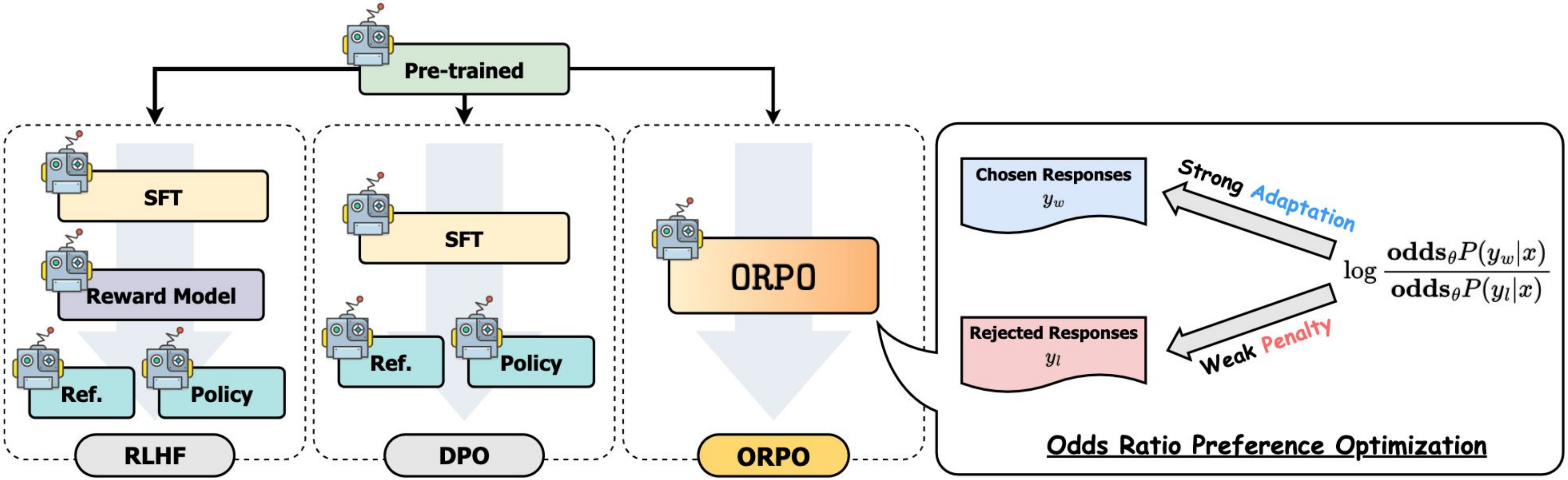
- Main takeaways
- SFT without ORPO
- ORPO Loss
- Gradient of ORPO Loss
- Training Details
- Results
- Limitations
- Related Work
- Future Work
- Extra Slides

Main Takeaways

- ORPO does not use a reference model like DPO (KL term) or a reward model and initial SFT model like RLHF.
- ORPO produced a preference-aligned SFT directly.
- The ORPO loss includes a penalty (added to the normal causal LM NLL loss) which maximizes the likelihood of generating a favored response.
- ORPO consistently preferred by a reward model against SFT and RLHF
- ORPO win rate vs. DPO increases as model size increases.


RLHF
Low quality data
Text
e.g. Internet data
Optimized for text completion
Language modeling
Pretrained LLM
High quality data
Demonstration data
Finetuned for dialogue
Supervised finetuning
SFT model
Human feedback
Comparison data
Trained to give a scalar score for (prompt, response)
Classification
Reward model
Prompts
Optimized to generate responses that maximize scores by reward model
Reinforcement Learning
Final model
Scale
May '23
>1 trillion tokens
10K - 100K (prompt, response)
100K - 1M comparisons (prompt, winning_response, losing_response)
10K - 100K prompts
Examples
Bolded: open sourced
GPT-x, Gopher, **Falcon**, LLaMa, **Pythia**, **Bloom**, **StableLM**
Dolly-v2, Falcon-Instruct
InstructGPT, ChatGPT, Claude, **StableVicuna**

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


Pre-trained
SFT
Reward Model
Ref.
Policy
RLHF
SFT
Ref.
Policy
DPO
ORPO
ORPO
Chosen Responses
y_w
Rejected Responses
y_l
Strong Adaptation
Weak Penalty
$\log \frac{\mathbf{odds}_\theta P(y_w|x)}{\mathbf{odds}_\theta P(y_l|x)}$
Odds Ratio Preference Optimization

Using the ORPOTrainer

For a detailed example have a look at the `examples/scripts/orpo.py` script. At a high level we need to initialize the `ORPOTrainer` with a `model` we wish to train. **Note that ORPOTrainer eliminates the need to use the reference model, simplifying the optimization process.** The `beta` refers to the hyperparameter `lambda` in eq. (6) of the paper and refers to the weighting of the relative odd ratio loss in the standard cross-entropy loss used for SFT.

```
orpo_config = ORPOConfig(
    beta=0.1, # the lambda/alpha hyperparameter in the paper/code
)

orpo_trainer = ORPOTrainer(
    model,
    args=orpo_config,
    train_dataset=train_dataset,
    tokenizer=tokenizer,
)
```

After this one can then call:

```
orpo_trainer.train()
```

```
orpo_dataset_dict = {
    "prompt": [
        "hello",
        "how are you",
        "What is your name?",
        "What is your name?",
        "Which is the best programming language?",
        "Which is the best programming language?",
        "Which is the best programming language?",
    ],
    "chosen": [
        "hi nice to meet you",
        "I am fine",
        "My name is Mary",
        "My name is Mary",
        "Python",
        "Python",
        "Java",
    ],
    "rejected": [
        "leave me alone",
        "I am not fine",
        "Whats it to you?",
        "I dont have a name",
        "Javascript",
        "C++",
        "C++",
    ],
}
```

SFT without ORPO

$$\mathcal{L} = -\frac{1}{m}\sum_{k=1}^{m}\log P(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) \quad (1)$$

$$= -\frac{1}{m}\sum_{k=1}^{m}\sum_{i=1}^{|V|} y_i^{(k)} \cdot \log(p_i^{(k)}) \quad (2)$$

Effective for domain adaptation but doesn't have a mechanism to penalized rejected responses.

y_i is a boolean value that indicates if the ith token in the vocabulary set V is a label token

p_i refers to the probability of the ith token

m is the length of sequence

Figure 3: Log probabilities for chosen and rejected responses during OPT-350M model fine-tuning on HH-RLHF dataset. Despite only chosen responses being used for supervision, rejected responses show a comparable likelihood of generation.

ORPO Loss

$$\mathcal{L}_{ORPO} = \mathbb{E}_{(x,y_w,y_l)}\left[\underbrace{\mathcal{L}_{SFT}}_{\text{Adapt to the specified subset of the desired domain}} + \underbrace{\lambda \cdot \mathcal{L}_{OR}}_{\text{Disfavor generations in the rejected responses set}}\right] \tag{6}$$

$$\mathcal{L}_{OR} = -\log \sigma \left(\log \frac{\mathbf{odds}_\theta(y_w|x)}{\mathbf{odds}_\theta(y_l|x)} \right) \tag{7}$$

ORPO Loss - TRL

```python
def odds_ratio_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
) -> Tuple[torch.FloatTensor, torch.FloatTensor, torch.FloatTensor, torch.FloatTensor, torch.FloatTensor]:
    """Compute ORPO's odds ratio (OR) loss for a batch of policy and reference model log probabilities.

    Args:
        policy_chosen_logps: Log probabilities of the policy model for the chosen responses. Shape: (batch_size,)
        policy_rejected_logps: Log probabilities of the policy model for the rejected responses. Shape: (batch_size,)

    Returns:
        A tuple of three tensors: (losses, chosen_rewards, rejected_rewards).
        The losses tensor contains the ORPO loss for each example in the batch.
        The chosen_rewards and rejected_rewards tensors contain the rewards for the chosen and rejected responses, respectively.
        The log odds ratio of the chosen responses over the rejected responses ratio for logging purposes.
        The `log(sigmoid(log_odds_chosen))` for logging purposes.
    """

    # Derived from Eqs. (4) and (7) from https://arxiv.org/abs/2403.07691 by using log identities and exp(log(P(y|x)) = P(y|x)
    log_odds = (policy_chosen_logps - policy_rejected_logps) - (
        torch.log1p(-torch.exp(policy_chosen_logps)) - torch.log1p(-torch.exp(policy_rejected_logps))
    )
    sig_ratio = F.sigmoid(log_odds)
    ratio = torch.log(sig_ratio)
    losses = self.beta * ratio

    chosen_rewards = self.beta * (policy_chosen_logps.to(self.accelerator.device)).detach()
    rejected_rewards = self.beta * (policy_rejected_logps.to(self.accelerator.device)).detach()

    return losses, chosen_rewards, rejected_rewards, torch.mean(ratio).item(), torch.mean(log_odds).item()
```

ORPO Loss - TRL

```python
def cross_entropy_loss(logits, labels):
    if not self.is_encoder_decoder:
        # Shift so that tokens < n predict n
        logits = logits[..., :-1, :].contiguous()
        labels = labels[..., 1:].contiguous()
    # Flatten the tokens
    loss_fct = nn.CrossEntropyLoss()
    logits = logits.view(-1, logits.shape[-1])
    labels = labels.view(-1)
    # Enable model parallelism
    labels = labels.to(logits.device)
    loss = loss_fct(logits, labels)
    return loss

if self.is_encoder_decoder:
    labels = concatenated_batch["concatenated_labels"].clone()
else:
    labels = concatenated_batch["concatenated_input_ids"].clone()

chosen_nll_loss = cross_entropy_loss(all_logits[:len_chosen], labels[:len_chosen])

all_logps = self.get_batch_logps(
    all_logits,
    concatenated_batch["concatenated_labels"],
    average_log_prob=True,
    is_encoder_decoder=self.is_encoder_decoder,
    label_pad_token_id=self.label_pad_token_id,
)

chosen_logps = all_logps[:len_chosen]
rejected_logps = all_logps[len_chosen:]

chosen_logits = all_logits[:len_chosen]
rejected_logits = all_logits[len_chosen:]

return (chosen_logps, rejected_logps, chosen_logits, rejected_logits, chosen_nll_loss)
```

ORPO Loss - TRL

```python
    def get_batch_loss_metrics(
        self,
        model,
        batch: Dict[str, Union[List, torch.LongTensor]],
        train_eval: Literal["train", "eval"] = "train",
    ):
        """Compute the ORPO loss and other metrics for the given batch of inputs for train or test."""
        metrics = {}

        (
            policy_chosen_logps,
            policy_rejected_logps,
            policy_chosen_logits,
            policy_rejected_logits,
            policy_nll_loss,
        ) = self.concatenated_forward(model, batch)

        losses, chosen_rewards, rejected_rewards, log_odds_ratio, log_odds_chosen = self.odds_ratio_loss(
            policy_chosen_logps, policy_rejected_logps
        )
        # full ORPO loss
        loss = policy_nll_loss - losses.mean()

        reward_accuracies = (chosen_rewards > rejected_rewards).float()

        prefix = "eval_" if train_eval == "eval" else ""
        metrics[f"{prefix}rewards/chosen"] = chosen_rewards.mean().cpu()
        metrics[f"{prefix}rewards/rejected"] = rejected_rewards.mean().cpu()
        metrics[f"{prefix}rewards/accuracies"] = reward_accuracies.mean().cpu()
        metrics[f"{prefix}rewards/margins"] = (chosen_rewards - rejected_rewards).mean().cpu()
        metrics[f"{prefix}logps/rejected"] = policy_rejected_logps.detach().mean().cpu()
        metrics[f"{prefix}logps/chosen"] = policy_chosen_logps.detach().mean().cpu()
        metrics[f"{prefix}logits/rejected"] = policy_rejected_logits.detach().mean().cpu()
        metrics[f"{prefix}logits/chosen"] = policy_chosen_logits.detach().mean().cpu()
        metrics[f"{prefix}nll_loss"] = policy_nll_loss.detach().mean().cpu()
        metrics[f"{prefix}log_odds_ratio"] = log_odds_ratio
        metrics[f"{prefix}log_odds_chosen"] = log_odds_chosen
```

The odds of generating the output sequence y given an input sequence x:

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

> Intuitively $\mathbf{odds}_\theta(y|x) = k$ implies that it is k times more likely for the model θ to generate the output sequence y than not generating it.

Writing that out:

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

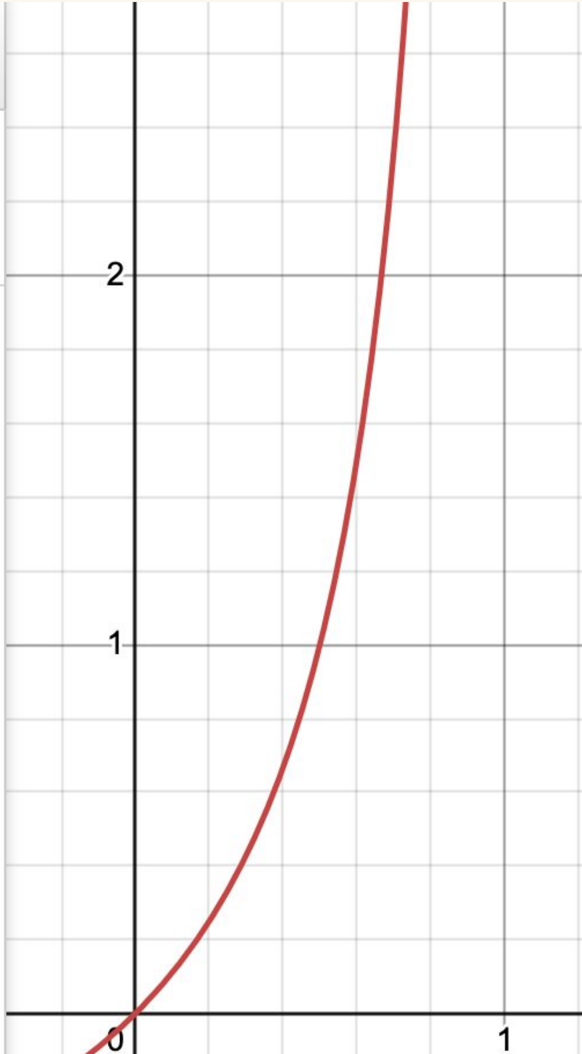
$$\frac{P_\theta(y|x)}{1 - P_\theta(y|x)} = k$$

$$P_\theta(y|x) = k[1 - P_\theta(y|x)]$$

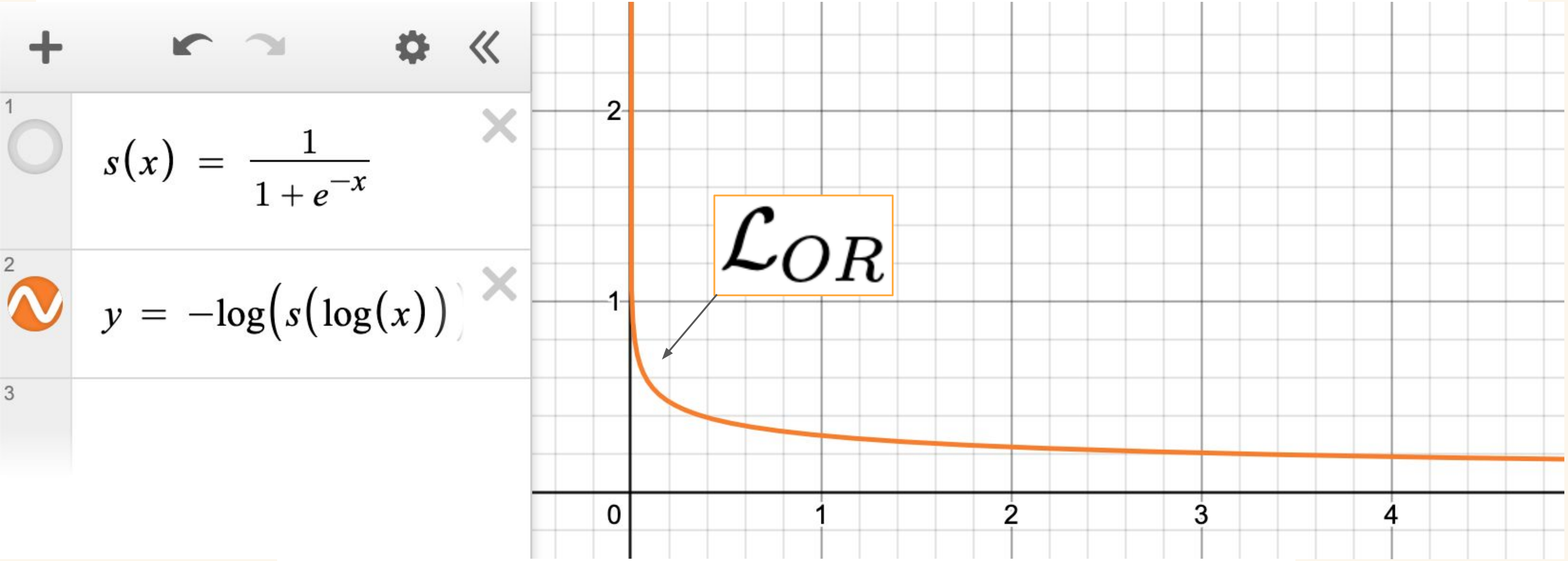
> it is k times more likely to generate y (i.e. P_θ)) than not generating it ($1 - P_\theta$).

$$y = \frac{x}{1-x}$$

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

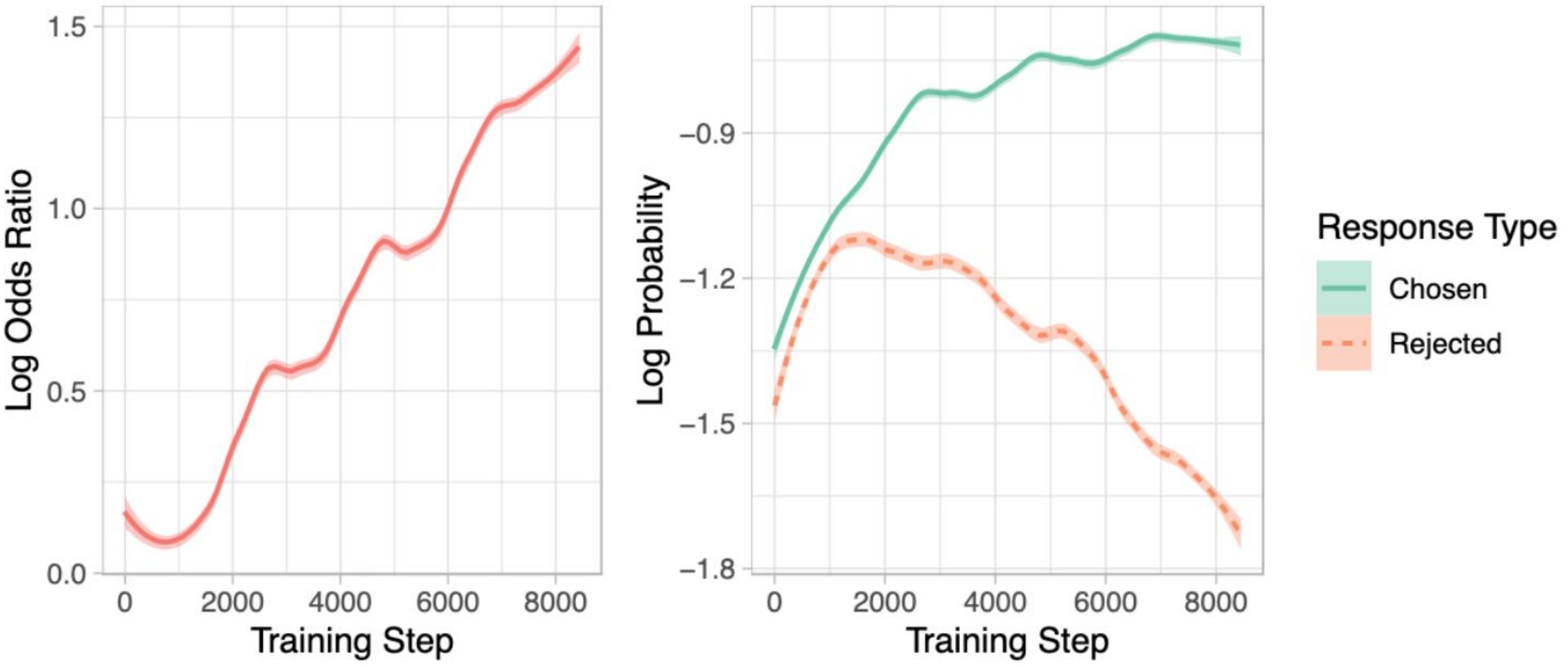


$$\mathcal{L}_{OR} = -\log \sigma \left(\log \frac{\mathbf{odds}_\theta(y_w|x)}{\mathbf{odds}_\theta(y_l|x)} \right) \quad (7)$$

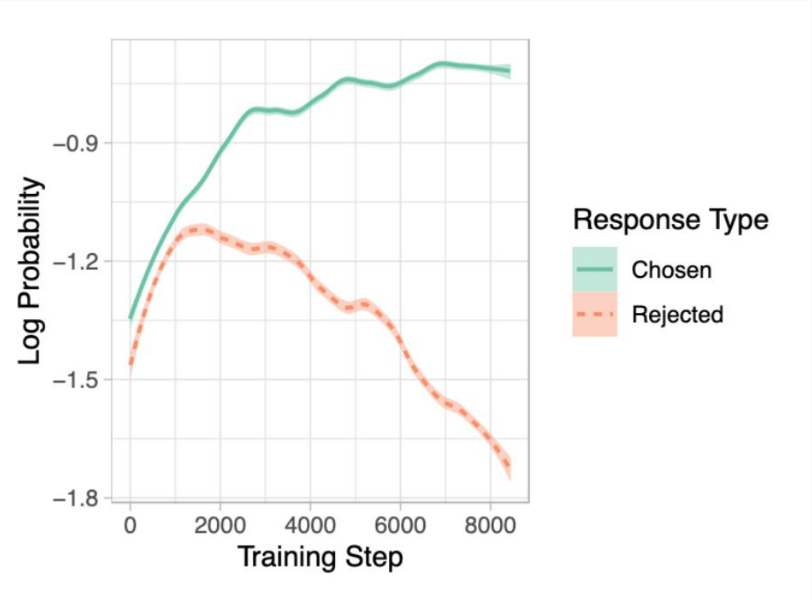


Minimizing $\mathcal{L}_{OR}$ means maximizing $\frac{\mathbf{odds}_\theta(y_w|x)}{\mathbf{odds}_\theta(y_l|x)}$

$$\mathcal{L}_{OR} = -\log \sigma \left(\log \frac{\mathbf{odds}_\theta(y_w|x)}{\mathbf{odds}_\theta(y_l|x)} \right) \qquad (7)$$



$$\mathcal{L}_{ORPO} = \mathbb{E}_{(x,y_w,y_l)}\left[\mathcal{L}_{SFT} + \lambda \cdot \mathcal{L}_{OR}\right] \quad (6)$$



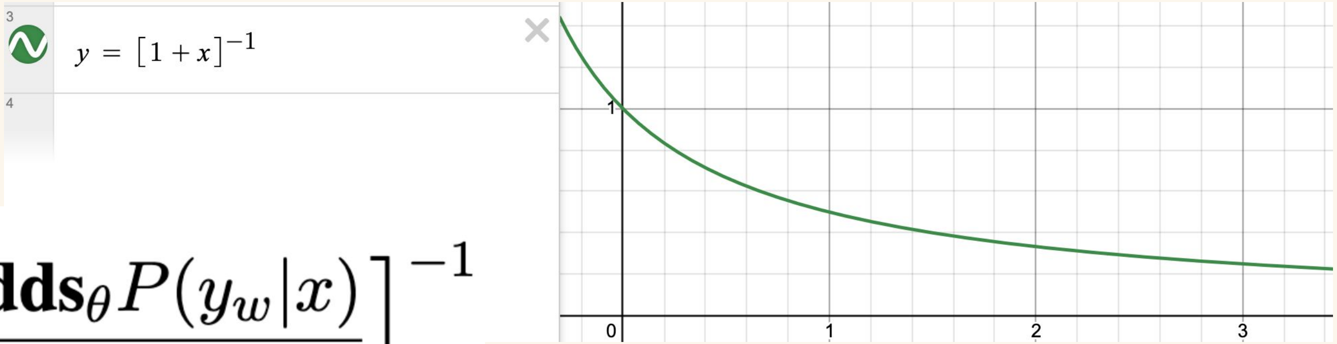
Gradient of ORPO Loss

$$\nabla_\theta \mathcal{L}_{OR} = \delta(d) \cdot h(d) \tag{8}$$

$$\delta(d) = \left[1 + \frac{\mathbf{odds}_\theta P(y_w|x)}{\mathbf{odds}_\theta P(y_l|x)}\right]^{-1} \tag{9}$$

$$h(d) = \frac{\nabla_\theta \log P_\theta(y_w|x)}{1 - P_\theta(y_w|x)} - \frac{\nabla_\theta \log P_\theta(y_l|x)}{1 - P_\theta(y_l|x)} \tag{10}$$

Gradient of ORPO Loss: δ(d)



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

When the odds of the favored response are relatively higher than the disfavored responses, δ(d) will converge to 0.

Gradient of ORPO Loss: δ(d)

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

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

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

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

δ(d) will play the role of a penalty term, accelerating the parameter updates if the model is more likely to generate the rejected responses.

Gradient of ORPO Loss: h(d)

$$h(d) = \frac{\nabla_\theta \log P_\theta(y_w|x)}{1 - P_\theta(y_w|x)} - \frac{\nabla_\theta \log P_\theta(y_l|x)}{1 - P_\theta(y_l|x)} \tag{10}$$

For the chosen responses, this accelerates the model's adaptation toward the distribution of chosen responses as the likelihood increases.

Gradient of ORPO Loss

$$\nabla_\theta \mathcal{L}_{OR} = \delta(d) \cdot h(d) \tag{8}$$

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

accelerates the parameter updates if the model is more likely to generate the rejected responses

$$h(d) = \frac{\nabla_\theta \log P_\theta(y_w|x)}{1 - P_\theta(y_w|x)} - \frac{\nabla_\theta \log P_\theta(y_l|x)}{1 - P_\theta(y_l|x)} \tag{10}$$

accelerates the model's adaptation toward the distribution of chosen responses as the likelihood increases

Gradient of ORPO Loss

$$\nabla_\theta \mathcal{L}_{OR} = \delta(d) \cdot h(d) \tag{8}$$

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

larger when rejected odds are larger than favored odds

$$h(d) = \frac{\nabla_\theta \log P_\theta(y_w|x)}{1 - P_\theta(y_w|x)} - \frac{\nabla_\theta \log P_\theta(y_l|x)}{1 - P_\theta(y_l|x)} \tag{10}$$

larger when favored odds are larger then rejected odds

Training - Models

- OPT (125M to 1.3B) for SFT, PPO, DPO and ORPO
- Phi-2 (2.7B)
- Llama 2 (7B)
- Mistral (7B)

Training - Hyperparameters

- Flash Attention 2
- OPT series and Phi-2: Deep Speed Zero 2
- Llama-2 (7B) and Mistral (7B): FSDP
- Optimizer: AdamW and paged Adam
- Linear Warmup with Cosine Decay
- Input Length: 1024 (HH), 2048 (UltraFeedback)

Training - Hyperparameters

- SFT: Max LR = 1e-5, 1 epoch
- DPO: β = 0.1, LR=5e-6, 3 epochs
- ORPO: LR=8e-6, 10 epochs

Hyperparameter	Setting
ppo_epoch	4
init_kl_coef	0.1
horizon	2,000
batch_size	64
mini_batch_size	8
gradient_accumulation_steps	1
output_min_length	128
output_max_length	512
optimizer	AdamW
learning_rate	1e-05
gamma	0.99

Table 5: Hyperparameter settings for RLHF.

Training - Datasets

- Anthropic’s HH-RLHF
- Binarized UltraFeedback

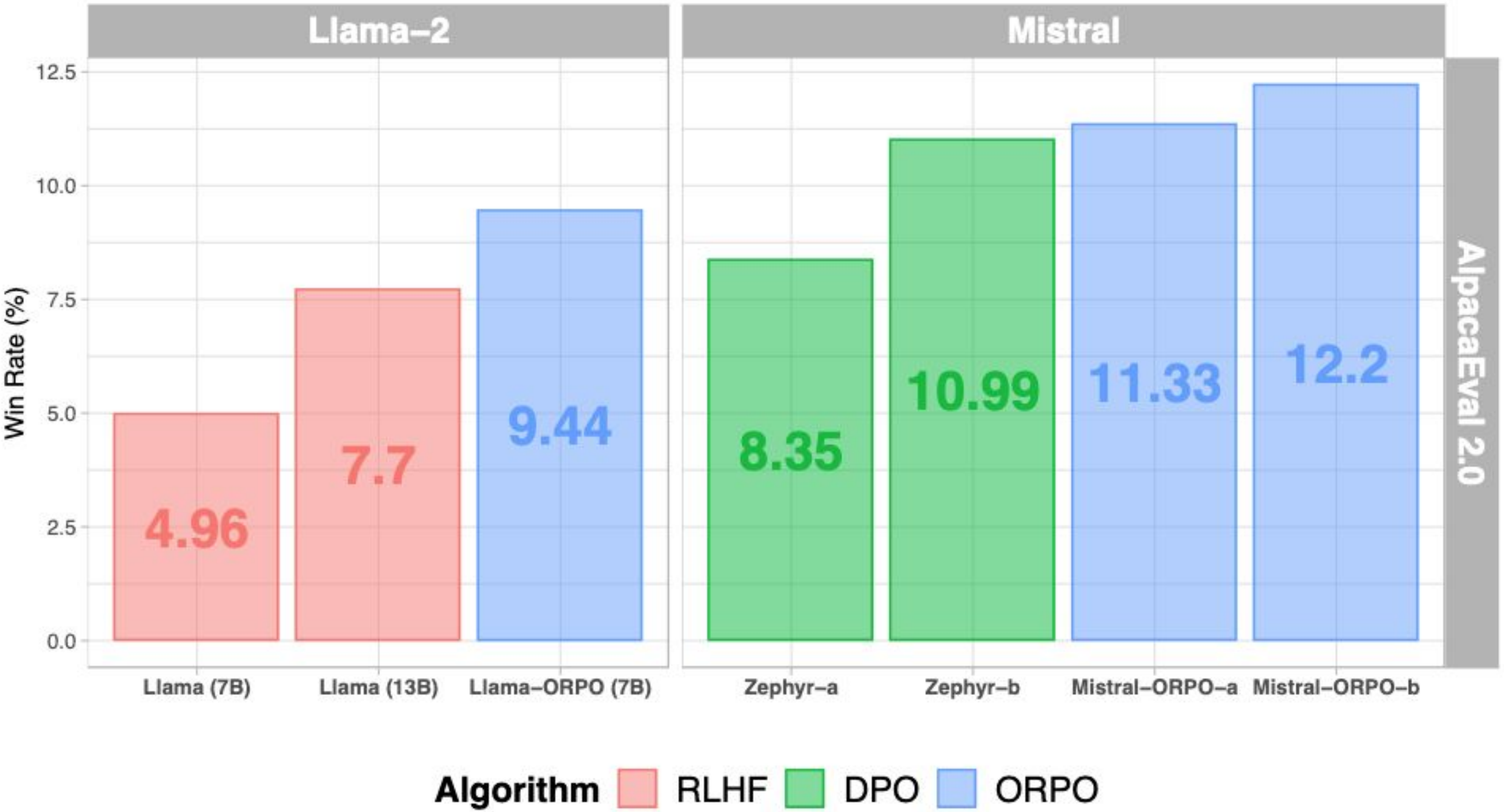
Training - Reward Model

- OPT 250M and 1.3B on each dataset for a single epoch

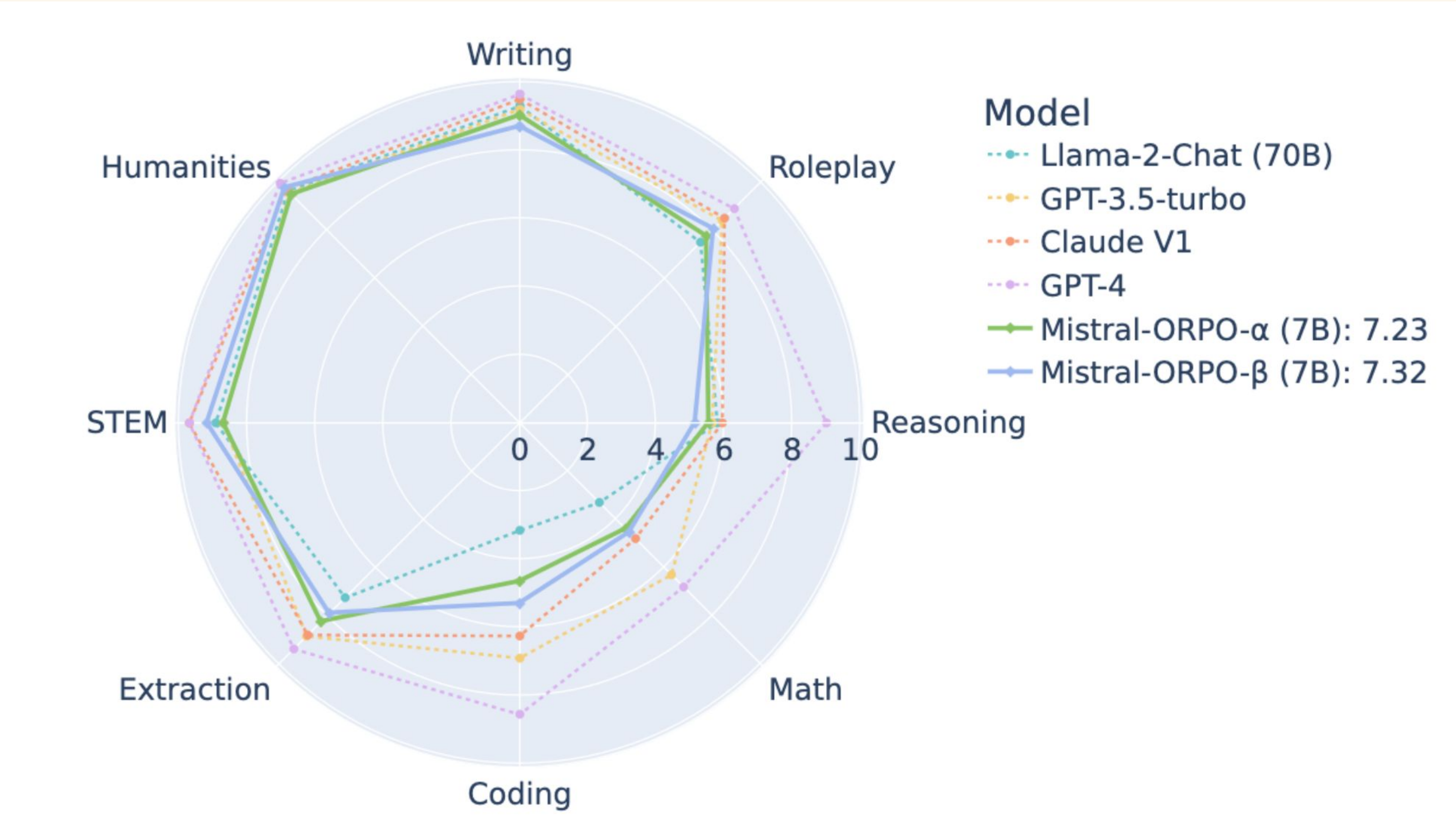
$$-\mathbb{E}_{(x,y_l,y_w)}\left[\log\sigma\left(r(x,y_w)-r(x,y_l)\right)\right] \qquad (11)$$

Results: AlpacaEval

Model Name	Size	$\text{AlpacaEval}_{1.0}$	$\text{AlpacaEval}_{2.0}$
Phi-2 + SFT	2.7B	48.37% (1.77)	0.11% (0.06)
Phi-2 + SFT + DPO	2.7B	50.63% (1.77)	0.78% (0.22)
Phi-2 + ORPO *(Ours)*	2.7B	**71.80% (1.59)**	**6.35% (0.74)**
Llama-2 Chat *	7B	71.34% (1.59)	4.96% (0.67)
Llama-2 Chat *	13B	81.09% (1.38)	7.70% (0.83)
Llama-2 + ORPO *(Ours)*	7B	**81.26% (1.37)**	**9.44% (0.85)**
Zephyr (α) *	7B	85.76% (1.23)	8.35% (0.87)
Zephyr (β) *	7B	90.60% (1.03)	10.99% (0.96)
Mistral-ORPO-α *(Ours)*	7B	87.92% (1.14)	11.33% (0.97)
Mistral-ORPO-β *(Ours)*	7B	**91.41% (1.15)**	**12.20% (0.98)**


Llama-2
Mistral
AlpacaEval 2.0
Win Rate (%)
12.5
10.0
7.5
5.0
2.5
0.0
4.96
7.7
9.44
8.35
10.99
11.33
12.2
Llama (7B)
Llama (13B)
Llama-ORPO (7B)
Zephyr-a
Zephyr-b
Mistral-ORPO-a
Mistral-ORPO-b
Algorithm
RLHF
DPO
ORPO

Results: MT-Bench



Results: MT-Bench (varying λ)

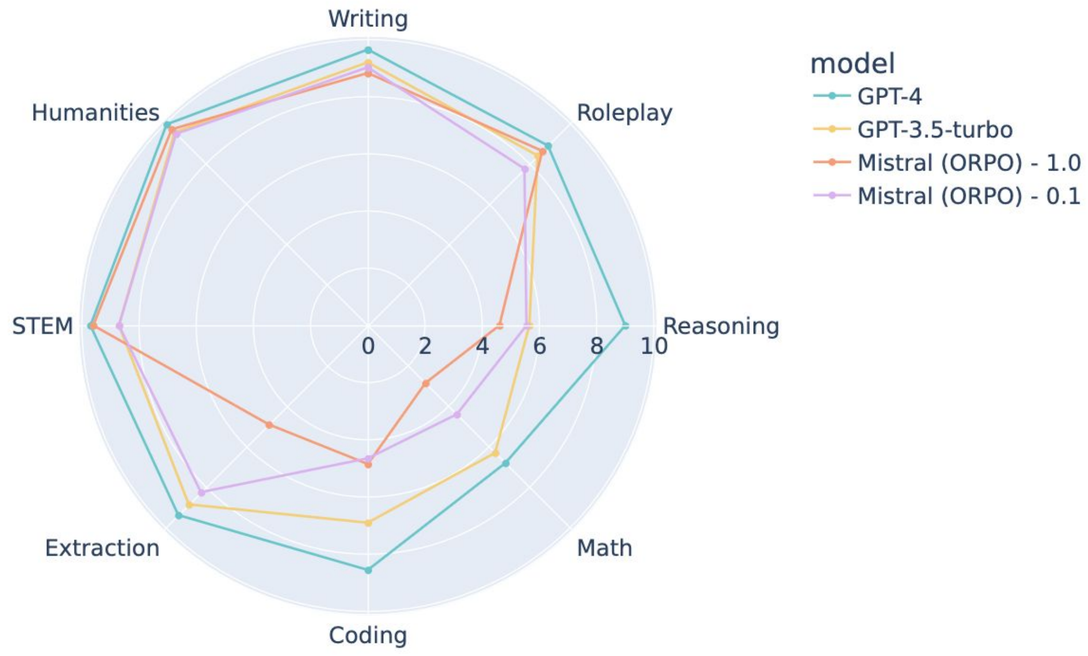


Figure 10: MT-Bench result comparison by differing $\lambda = 0.1$ and $\lambda = 1.0$.

Results: MT-Bench (varying λ)

$$\mathcal{L}_{ORPO} = \mathbb{E}_{(x,y_w,y_l)}\left[\mathcal{L}_{SFT} + \lambda \cdot \mathcal{L}_{OR}\right]$$

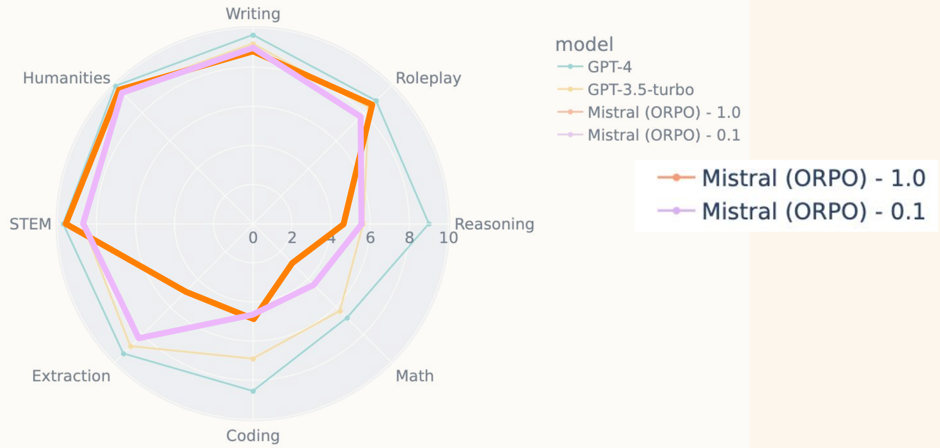


Figure 10: MT-Bench result comparison by differing $\lambda = 0.1$ and $\lambda = 1.0$.

In comparison to λ = 0.1, Mistral+ORPO (7B) with λ = 1.0 performs worse in extraction, math, and reasoning, which are the categories that generally require deterministic answers. On the other hand, it performs better in STEM, humanities, and roleplay, which ask the generations without hard answers.

Results: HH-RLHF

ORPO vs	SFT	+DPO	+PPO
OPT-125M	84.0 (0.62)	41.7 (0.77)	66.1 (0.26)
OPT-350M	82.7 (0.56)	49.4 (0.54)	79.4 (0.29)
OPT-1.3B	78.0 (0.16)	70.9 (0.52)	65.9 (0.33)

Table 2: Average win rate (%) and its standard deviation of ORPO and standard deviation over other methods on **HH-RLHF** dataset for three rounds. Sampling decoding with a temperature of 1.0 was used on the test set.

Results: UltraFeedback

ORPO vs	SFT	+DPO	+PPO
OPT-125M	73.2 (0.12)	48.8 (0.29)	71.4 (0.28)
OPT-350M	80.5 (0.54)	50.5 (0.17)	85.8 (0.62)
OPT-1.3B	69.4 (0.57)	57.8 (0.73)	65.7 (1.07)

Table 3: Average win rate (%) and its standard deviation of ORPO and standard deviation over other methods on **UltraFeedback** dataset for three rounds. Sampling decoding with a temperature of 1.0 was used.

Results: Reward Distribution

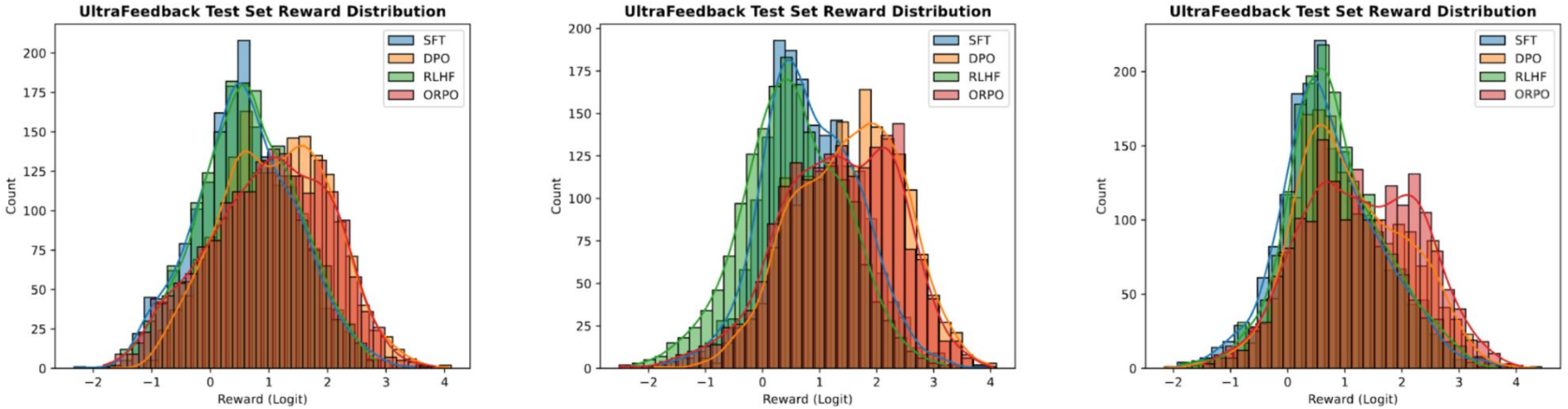


Figure 5: Reward distribution comparison between OPT-125M (left), OPT-350M (middle), and OPT-1.3B (right) trained with SFT (blue), RLHF (green), DPO (orange), and ORPO (red) on the test set of UltraFeedback using the RM-1.3B. While the rewards of the trained models are roughly normal and preference optimization algorithms (RLHF, DPO, and ORPO) tend to move the reward distribution in the positive direction, ORPO is on par or better than RLHF and DPO in increasing the expected reward. The same plot for the HH-RLHF dataset is in Appendix F.

Results: Lexical Diversity

	Per Input↓	**Across Input↓**
Phi-2 + SFT + DPO	**0.8012**	0.6019
Phi-2 + ORPO	0.8909	**0.5173**
Llama-2 + SFT + DPO	**0.8889**	0.5658
Llama-2 + ORPO	0.9008	**0.5091**

Table 4: Lexical diversity of Phi-2 and Llama-2 fine-tuned with DPO and ORPO. Lower cosine similarity is equivalent to higher diversity. The highest value in each column within the same model family is bolded.

Results: Computational Efficiency

- ORPO does not require a reference model which in RLHF and DPO is the model trained with SFT used as the baseline model for updating the parameters.
- DPO and RLHF require two SFT models: a frozen reference model (KL term) and the model being trained.
- ORPO only has one model: the model being trained with SFT. This requires half the forward passes of DPO or RLHF.

Limitations

- Lacks comparisons with alignment algorithms other than PPO and DPO
- Only trained models up to 7B parameters

Future Work

- Evaluate performance on models larger than 7B parameters
- Evaluate impact of ORPO on pretrained model
- Expand to consecutive preference alignment algorithms

Related Work

- Alignment with Reinforcement Learning (RLHF)
- Alignment without Reward Model (DPO)
- Alignment with Supervised Fine-Tuning (filtered datasets)

Extra Slides

Why not Probability Ratio?

$$\mathbf{PR}_\theta(y_w, y_l) = \frac{P_\theta(y_w|x)}{P_\theta(y_l|x)} \quad (16)$$

The probability ratio leads to more extreme discrimination of the disfavored responses than the odds ratio.

The excessive margin could lead to the unwarranted suppression of logits for tokens in disfavored responses within the incorporated setting, potentially resulting in issues of degeneration.

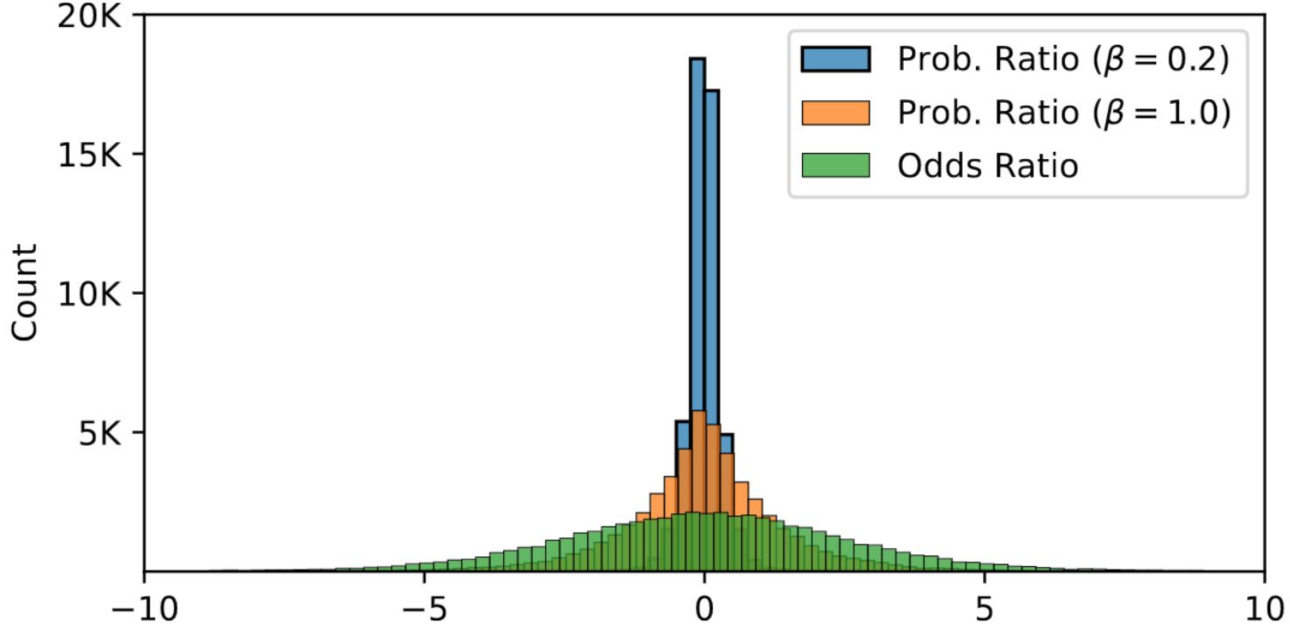


Figure 6: Sampled distribution of $\log \mathbf{PR}(X_2|X_1)$ and $\log \mathbf{OR}(X_2|X_1)$. $\log \mathbf{OR}(X_2|X_1)$ has a wider range given the same input probability pairs (X_1, X_2).

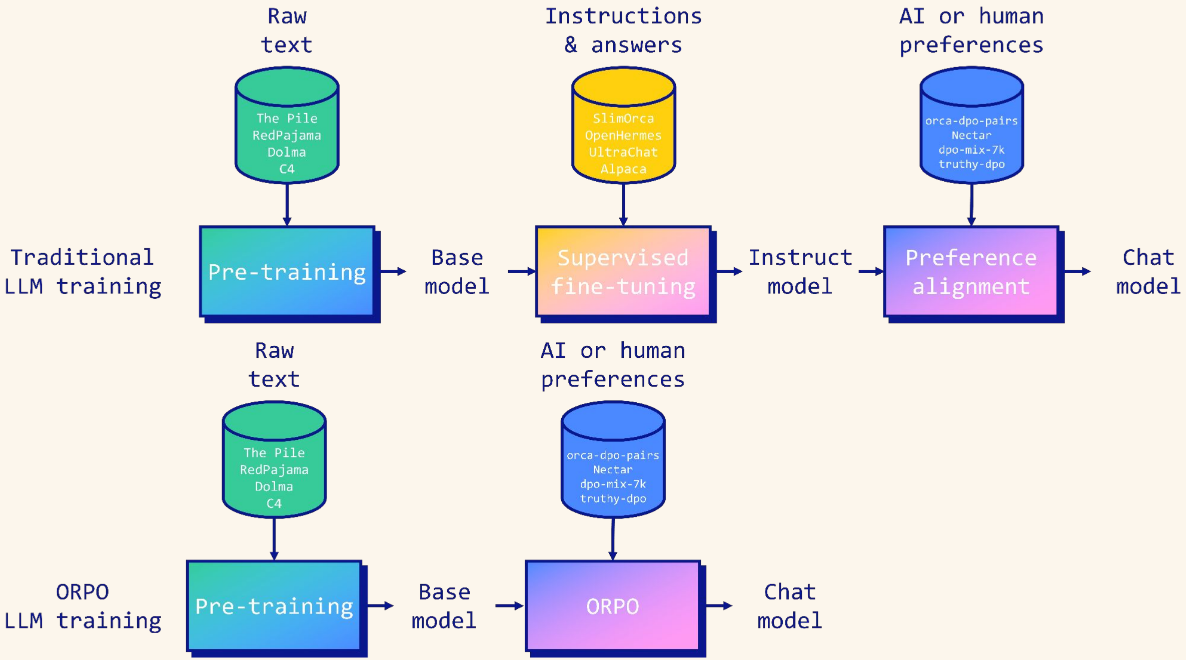
Weighting Value λ

$$\mathcal{L}_{ORPO} = \mathbb{E}_{(x,y_w,y_l)}\left[\mathcal{L}_{SFT} + \lambda \cdot \mathcal{L}_{OR}\right]$$

Figure 9: The log probability trend by λ. With larger λ (e.g., $\lambda = 1.0$), $\mathcal{L}_{OR}$ gets more influential in fine-tuning the models with ORPO.

Fine-tuning Llama 3 with ORPO

https://huggingface.co/blog/mlabonne/orpo-llama-3



Math Notebook

https://colab.research.google.com/drive/1dgR8O4pbuvecVEkfq6xf_dHwVn2G9Kjf?usp=sharing