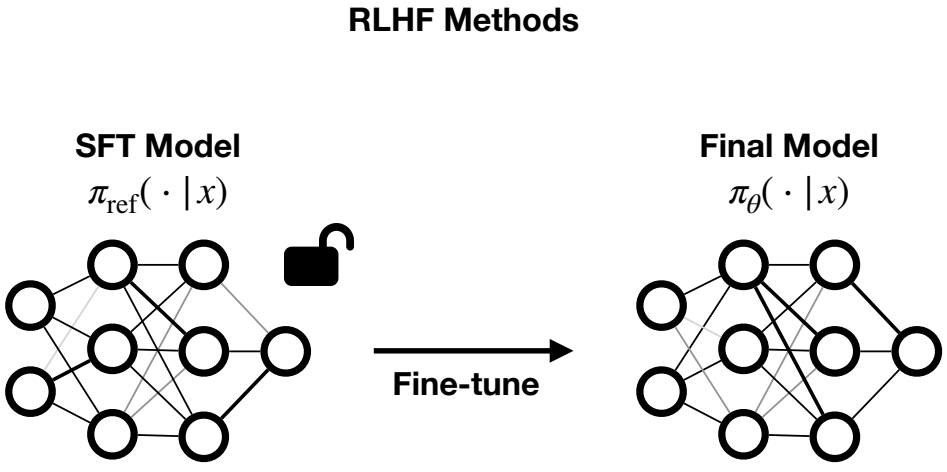
Preference Alignment with Flow Matching

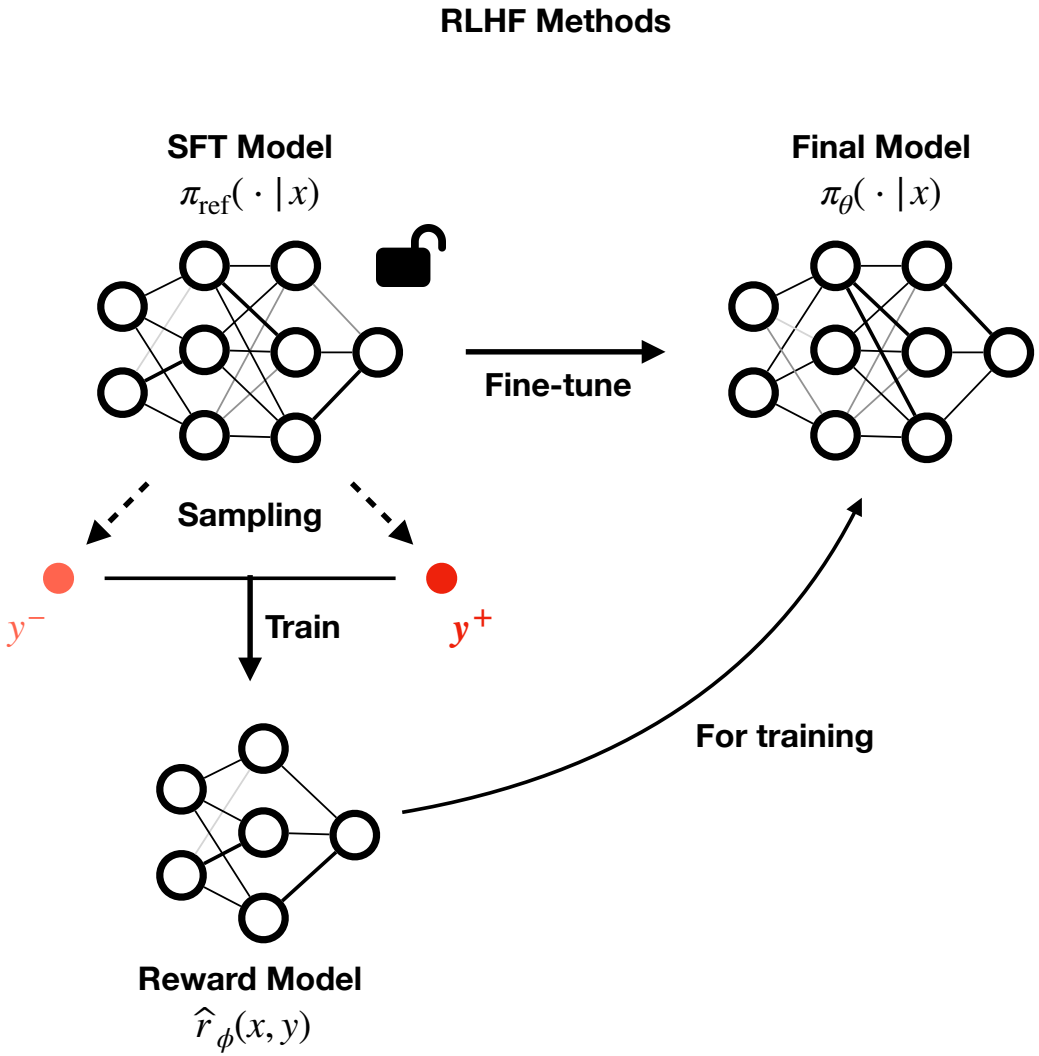
Minu Kim*, Yongsik Lee*, Sehyeok Kang, Jihwan Oh, Song Chong, Se-Young Yun

KAIST AI

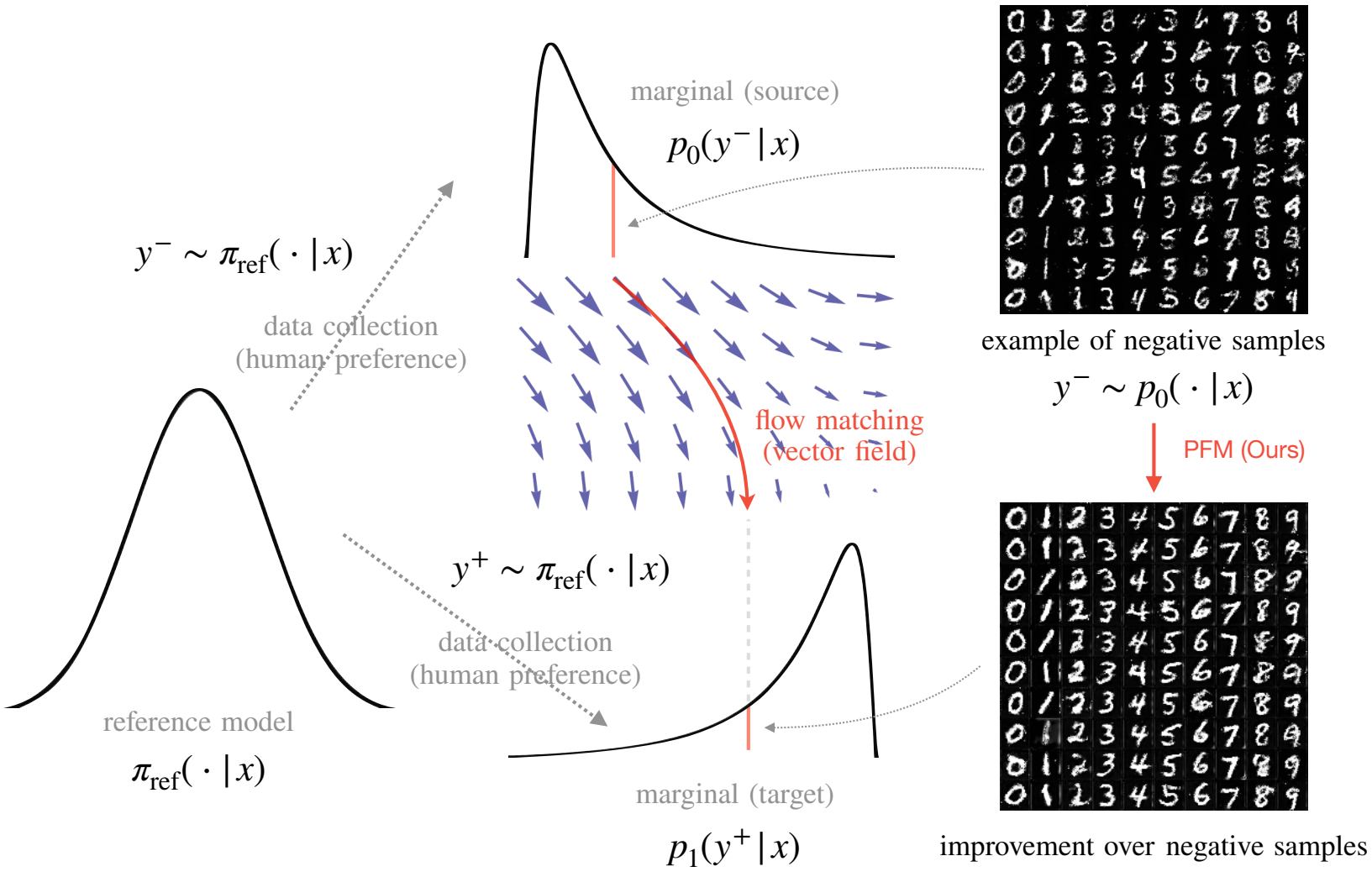
Aligning Black-Box Models with Preferences



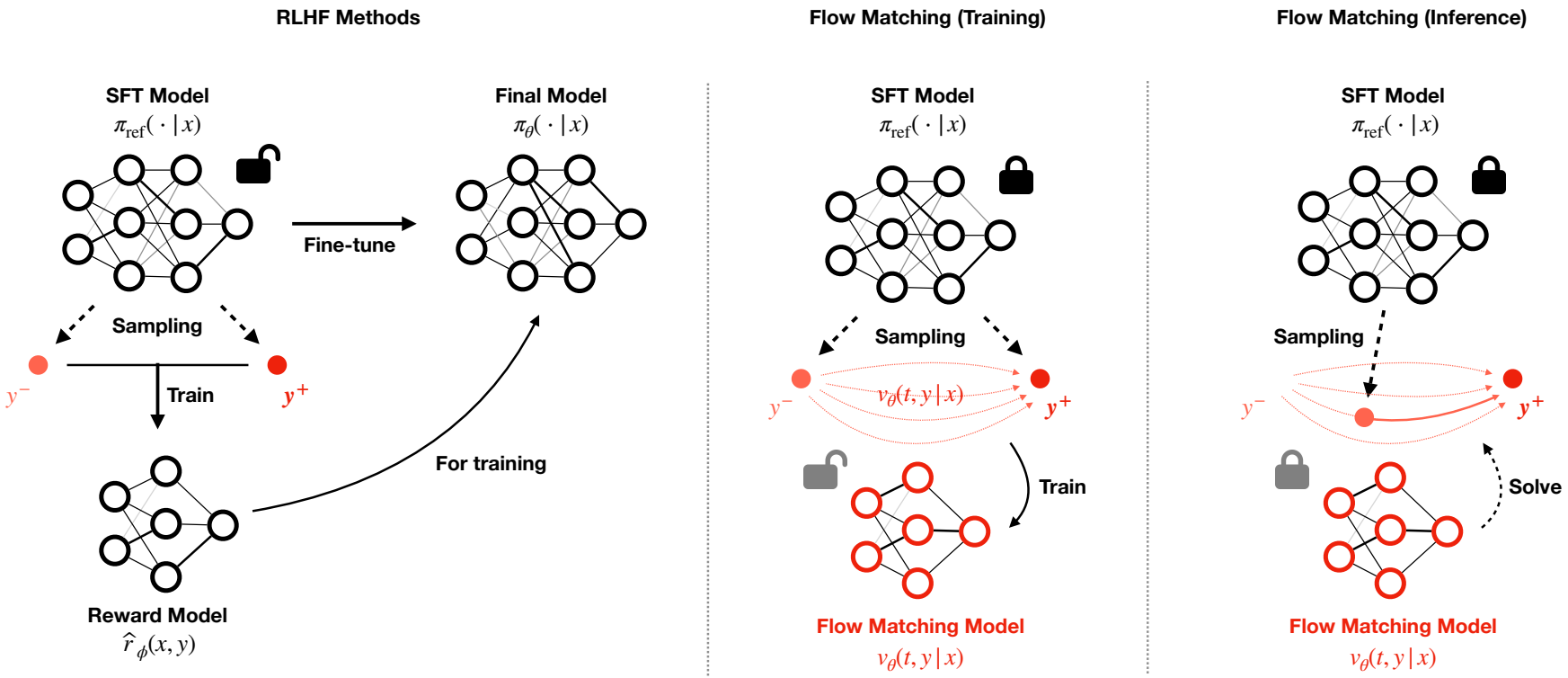
Aligning Black-Box Models with Preferences



Flow Matching Transports Probability Distributions

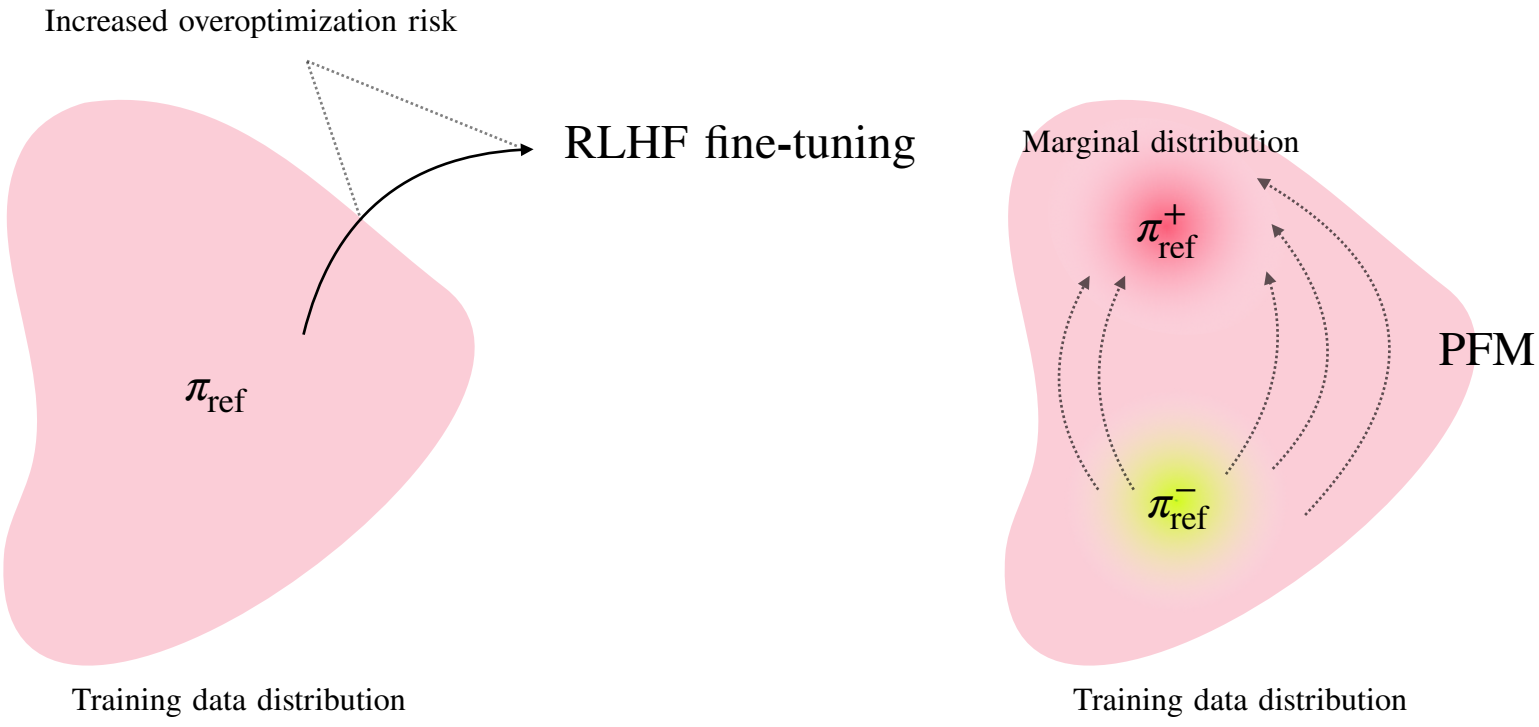


Preference Alignment with Flow Matching

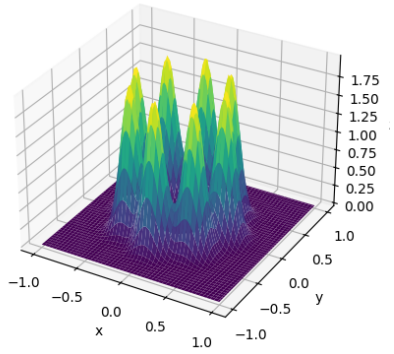


	RLHF	DPO	IPO	Ours
Reward Model Free	✗	✓	✓	✓
No Reward Assumptions (e.g. BT)	✗	✗	✓	✓
Applicable to Black-Box Models	✗	✗	✗	✓

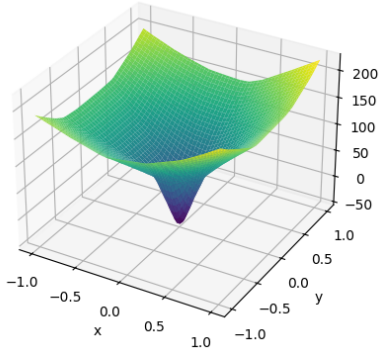
Mitigating Reward Overoptimization from Distribution Shift



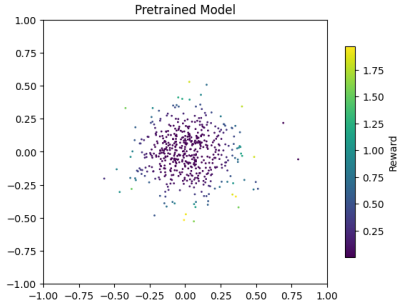
Illustrative Example



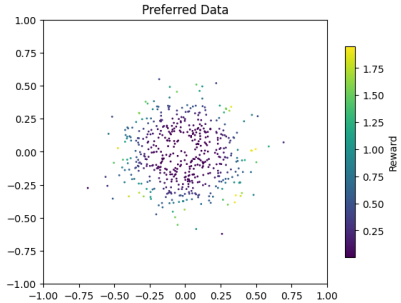
(a) True reward



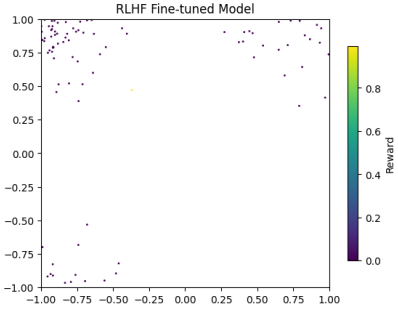
(b) Learned reward



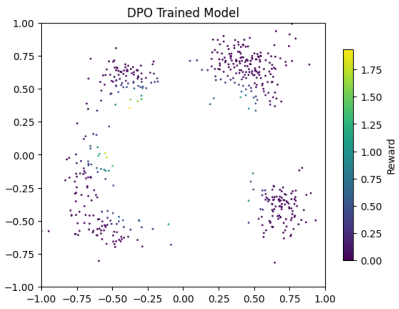
(c) Pre-trained



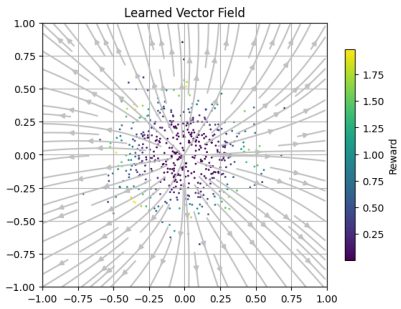
(d) Preferred



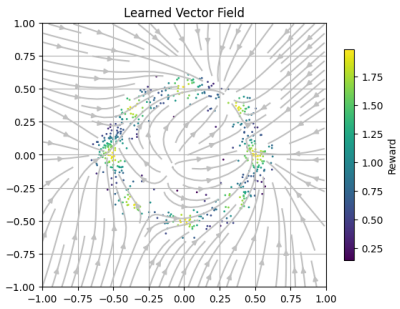
(e) RLHF



(f) DPO



(g) PFM



(h) PFM (5 iter.)

Theoretical Analysis

Theorem 1 (Characterization of the Marginal)

Let p_1 denote the marginal distribution of the positively-labeled samples y^+. Then the marginal distribution p_1 obtained from the preference model $\mathbb{P}(y > y'|x)$ is an optimizer of the optimization problem

$$p_1 = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{y\sim\pi}\left(\log \mathbb{E}_{y'\sim\pi_{\text{ref}}}\left(\mathbb{P}(y > y')\right)\right) - \mathbb{D}_{\text{KL}}\left(\pi \middle\| \pi_{\text{ref}}\right). \tag{1}$$

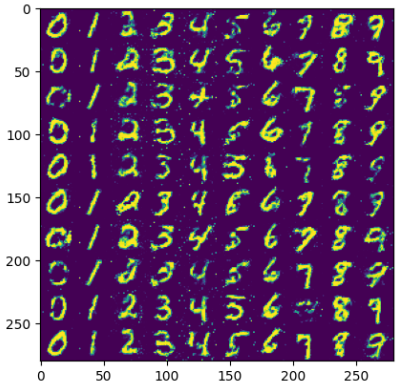
Theorem 2 (Convergence of Iterative Method)

Assume $\pi_{\text{ref}} \in L^2$ and $\mathbb{P}(y > y') \in L^2$. Consider an iterative update of the marginal distribution p_1. Then, the iteration converges to the uniform distribution of points y where the value $\mathbb{E}_{y^-\sim\pi_{\text{ref}}}\left(\mathbb{P}(y > y^-)\right)$ is the largest, i.e.,

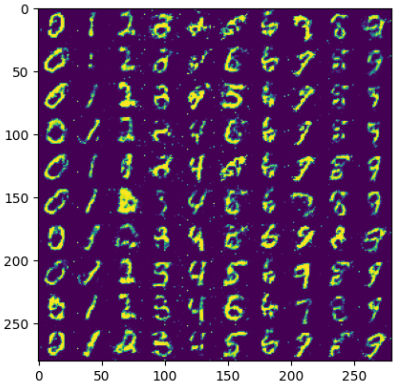
$$p_1^{(\infty)} \to U\left(\left\{y : y \in \underset{y}{\operatorname{argmax}}\, \mathbb{E}_{y^-\sim\pi_{\text{ref}}}\left(\mathbb{P}(y > y^-)\right)\right\}\right), \tag{2}$$

where U stands for uniform distribution.

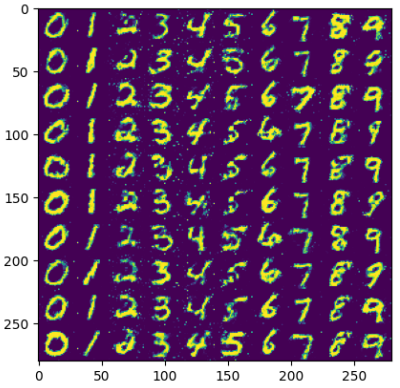
Experimental Results: Conditional Image Generation



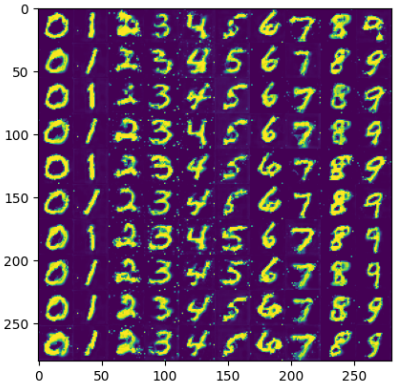
(a) $y \sim \pi_{\text{ref}}$ (0.8389)



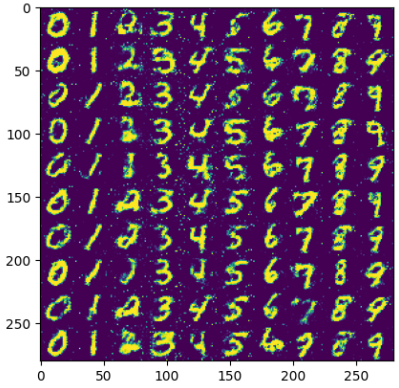
(b) y^- (0.3951)



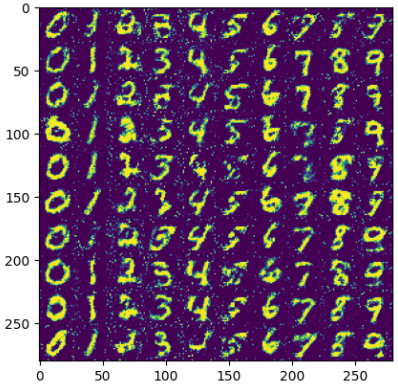
(c) y^+ (0.9976)



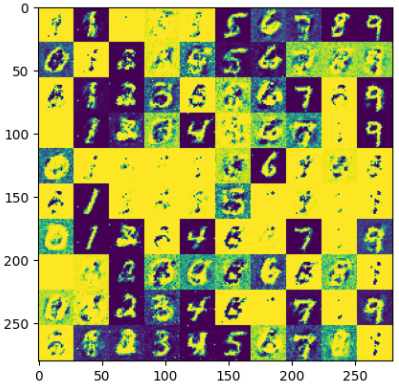
(d) PFM (0.9841)



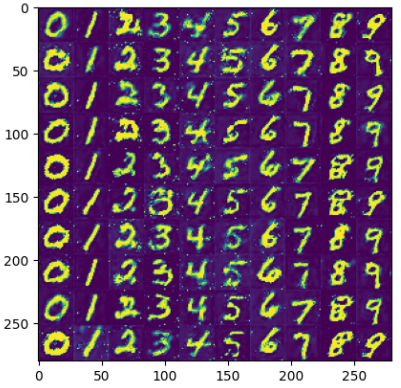
(e) RLHF (0.9171)



(f) DPO (0.8936)



(g) DPO $\beta\downarrow$ (0.5397)



(h) PFM $\times 2$ (**0.9996**)

Experimental Results: Controlled Text Generation

Table 1: Average preference scores of 100 IMDB test instances.

Reference	PFM	PFM×5
-0.3607	0.6399	**2.7469**
RLHF fine-tuned	RLHF + PFM	RLHF + PFM×5
2.0156	2.178	**2.7894**

Table 2: GPT-4 win rate vs. RLHF (PPO) fine-tuned policy.

PFM	PFM×5	RLHF + PFM	RLHF + PFM×5
2%	85%	99%	**100%**

Experimental Results: Offline RL

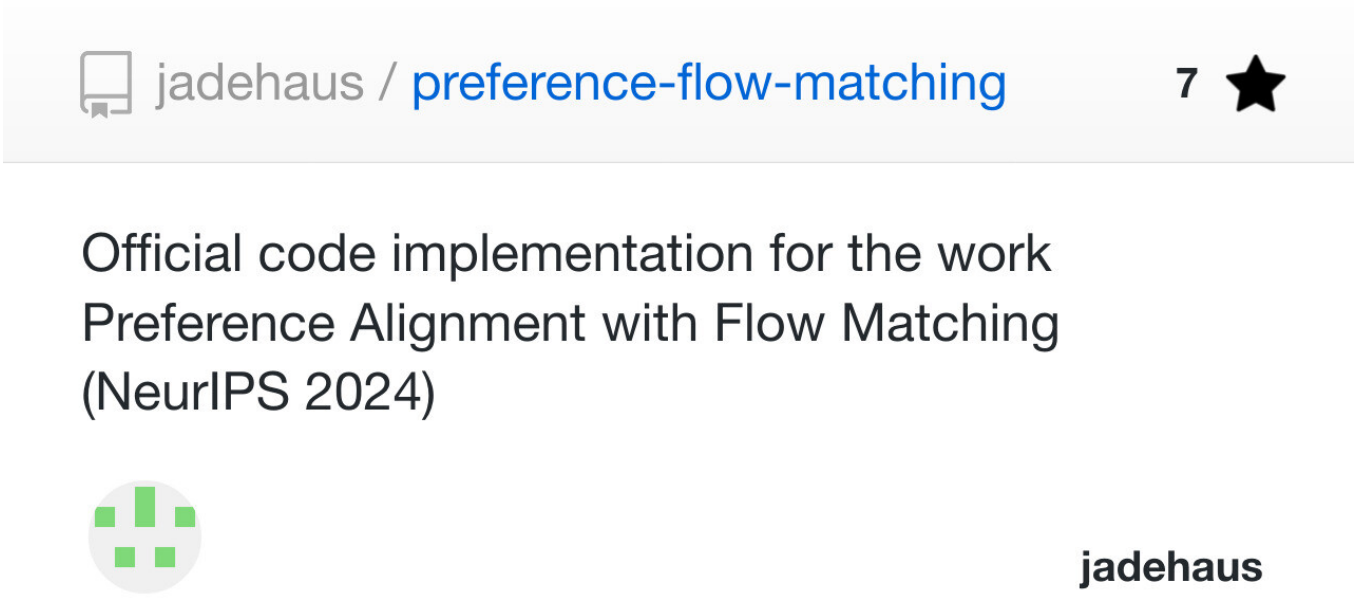
Dataset	Behavior Cloning	DPO Fine-tuned	PFM (Ours)	Marginal BC
ant-random-v2	31.59 ± 0.05	31.52 ± 0.08	**31.62 ± 0.13**	25.01 ± 4.64
ant-medium-v2	90.16 ± 21.48	95.04 ± 13.93	96.73 ± 2.47	**99.67 ± 1.57**
ant-expert-v2	125.83 ± 24.07	**134.96 ± 3.76**	132.20 ± 2.69	99.29 ± 34.74
hopper-random-v2	3.17 ± 0.25	3.23 ± 0.25	**7.69 ± 0.08**	5.48 ± 4.46
hopper-medium-v2	52.83 ± 5.03	53.47 ± 3.92	**58.76 ± 2.62**	40.44 ± 1.69
hopper-expert-v2	111.27 ± 0.48	111.51 ± 0.92	**111.70 ± 0.77**	32.39 ± 0.1
halfcheetah-random-v2	2.25 ± 0.01	2.26 ± 0.01	**2.26 ± 0.0**	2.21 ± 0.02
halfcheetah-medium-v2	40.97 ± 0.89	41.94 ± 0.68	**43.49 ± 0.88**	38.79 ± 1.27
halfcheetah-expert-v2	91.02 ± 1.24	**92.15 ± 0.76**	90.05 ± 0.83	4.77 ± 2.5
walker2d-random-v2	1.47 ± 0.1	1.38 ± 0.08	1.77 ± 0.13	**2.45 ± 0.38**
walker2d-medium-v2	60.35 ± 18.16	**74.05 ± 12.05**	72.59 ± 15.8	65.29 ± 12.58
walker2d-expert-v2	**108.62 ± 0.39**	108.38 ± 0.28	108.36 ± 0.21	15.8 ± 0.54
Random Average	9.62	9.60	**10.84**	8.79
Medium Average	61.08	66.13	**67.89**	61.05
Expert Average	109.19	**111.75**	110.58	38.06
D4RL Average	59.97	62.49	**63.10**	35.97

Source Code Available

```python
from flow import OptimalTransportConditionalFlowMatching

reference_policy = ... # Any base (reference) model or your interest
preference_dataset = ... # Your preference dataset, yielding pairs (x, y^+,
y^-).
# Flow matching model. (e.g., UNet, MLP)
flow_model = ...

# For training PFM, simply call `fit` method.
flow_matching = OptimalTransportConditionalFlowMatching(flow_model)
flow_matching.fit(
  dataset,
  conditional=True # Set to `True` if contexts x are given.
)

# For evaluation, simply modify the generated output from the reference policy
# using the `compute_target` method.
context = ... # e.g. a prompt
generated_sample = reference_policy(context) # e.g. a response
improved_sample = flow_matching.compute_target(
  generated_sample,
  context=context
)
```

https://github.com/jadehaus/preference-flow-matching

Source Code Available



https://github.com/jadehaus/preference-flow-matching