
Discovering Preference Optimization Algorithms with and for Large Language Models

Chris Lu*

Sakana AI and FLAIR
chrislu@sakana.ai

Samuel Holt*

University of Cambridge
sih31@cam.ac.uk

Claudio Fanconi*

University of Cambridge
caf83@cam.ac.uk

Alex J. Chan[†]

University of Cambridge
ajc340@cam.ac.uk

Jakob Foerster[‡]

FLAIR, University of Oxford
jakob.foerster@eng.ox.ac.uk

Mihaela van der Schaar[‡]

University of Cambridge
mv472@cam.ac.uk

Robert Tjarko Lange[‡]

Sakana AI
robert@sakana.ai

Abstract

Offline preference optimization is a key method for enhancing and controlling the quality of Large Language Model (LLM) outputs. Typically, preference optimization is approached as an offline supervised learning task using manually crafted convex loss functions. While these methods are based on theoretical insights, they are inherently constrained by human creativity, so the large search space of possible loss functions remains under-explored. We address this by performing LLM-driven *objective discovery* to automatically discover new state-of-the-art preference optimization algorithms without (expert) human intervention. Specifically, we iteratively prompt an LLM to propose and implement new preference optimization loss functions based on previously evaluated performance metrics. This process leads to the discovery of previously unknown and performant preference optimization algorithms. The best performing of these we call *Discovered Preference Optimization* (DiscoPOP)¹, a novel algorithm that adaptively blends logistic and exponential losses. Experiments demonstrate the state-of-the-art performance of DiscoPOP and its successful transfer to held-out tasks.

1 Introduction

Training Large Language Models (LLMs) usually involves starting with a model pre-trained on large text corpora and then fine-tuning it to match human preferences. Pre-trained, and even instruction fine-tuned LLMs, can generate harmful, dangerous, and unethical completions [Carlini et al., 2021, Gehman et al., 2020]. To mitigate this and align an LLM with human values, we use human preference alignment through preference-ranked completion data. This approach has become an industry standard, popularized by reinforcement learning with human feedback (RLHF) [Christiano et al., 2017, RLHF], and more recently, by offline preference optimization algorithms like direct preference optimization [Rafailov et al., 2023, DPO] and sequence likelihood calibration [Zhao

*Equal Contribution.

[†]Work partially done at Spotify.

[‡]Equal Advising.

¹Code: <https://github.com/luchris429/DiscoPOP>.

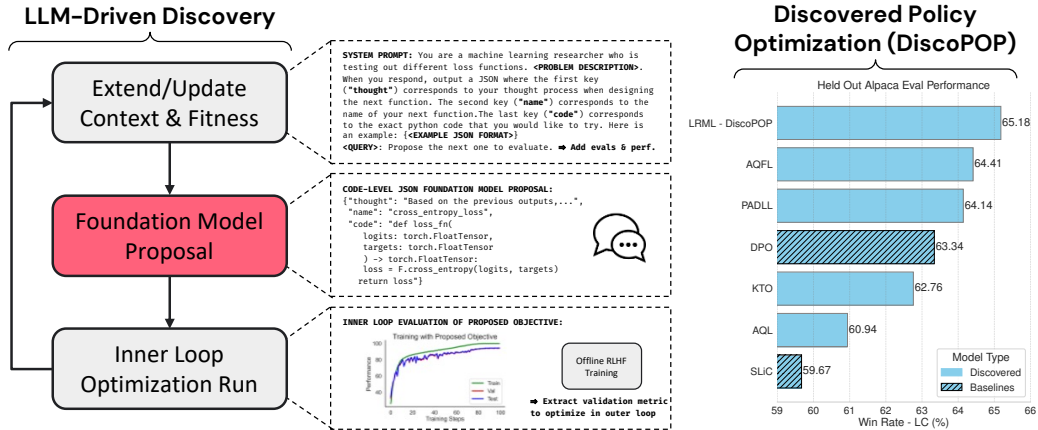


Figure 1: **Left.** Conceptual illustration of LLM-driven discovery of objective functions. We prompt an LLM to output new code-level implementations of offline preference optimization losses $\mathbb{E}_{(y_w, y_l, x) \sim \mathcal{D}} [f(\beta\rho)]$ as a function of the policy (π_θ) and reference model’s (π_{ref}) likelihoods of the chosen (y_w) and rejected (y_l) completions. Afterwards, we run an inner loop training procedure and evaluate the resulting model on MT-Bench. The corresponding performance is fed back to the language model, and we query it for the next candidate. **Right.** Performance of discovered objective functions on Alpaca Eval.

et al., 2023, SLiC], which cast the problem as a supervised learning objective. Many algorithms have been proposed in the literature for offline preference optimization, and it remains an open question which one performs best across tasks. While a strictly dominant algorithm may not exist, some algorithms likely exhibit generally improved performance. To date, all existing state-of-the-art preference optimization algorithms [Rafailov et al., 2023, Azar et al., 2023, Zhao et al., 2023] have been developed by human experts. Despite their advancements, these solutions are inherently constrained by human limitations, including creativity, ingenuity, and expert knowledge.

In this work, we aim to address these limitations by performing LLM-driven discovery to *automatically* generate new state-of-the-art preference optimization algorithms without continual expert human intervention in the development process. While previous works [Ma et al., 2023, Yu et al., 2023] have used LLMs to design environment-specific RL reward functions, we discover general-purpose objective functions which can be used across various preference optimization tasks. More specifically, we iteratively prompt an LLM to propose new preference optimization loss functions and evaluate them, with the previously proposed loss functions and their task performance metric (in our case, MT-Bench scores [Zheng et al., 2024]) as in-context examples. After performing this automatic discovery process, we catalogue high-performing loss functions and introduce a particularly strong one we call *Discovered Preference Optimization* (DiscoPOP), a new algorithm. To ensure robustness beyond MT-Bench, we validate DiscoPOP using AlpacaEval 2.0 [Dubois et al., 2024], showing an improvement in win rates against GPT-4 from DPO (11.23% \rightarrow 13.21%). Additionally, in separate, held-out, tasks such as summarization and controlled generation, models trained with the DiscoPOP loss outperform or perform competitively with existing preference optimization algorithms.

Contributions: ① We propose an LLM-driven objective discovery pipeline to discover novel offline preference optimization algorithms (Section 3). ② We discover multiple high-performing preference optimization losses. One such loss, which we call *Discovered Preference Optimization* (DiscoPOP), achieves strong performance across multiple held-out evaluation tasks of multi-turn dialogue (AlpacaEval 2.0), controlled sentiment generation (IMDb) and summarization (TL;DR) tasks. ③ We provide an initial analysis of DiscoPOP, which is a weighted sum of logistic and exponential losses, and discover surprising features. For example, DiscoPOP is non-convex.

2 Background

Preference Optimization. Consider a pre-trained language model policy π_θ and a dataset $\mathcal{D} = \{(x^i, y_w^i, y_l^i)\}_{i=1}^N$ consisting of prompts x and preference-ranked completions y_w and y_l . In this dataset, a human rater prefers y_w over y_l , denoted as $y_w \succ y_l$. The task is to align π_θ with the human values implicit in these preferences. Canonically, this has been achieved through reinforcement learning from human feedback [Christiano et al., 2017, RLHF], an approach that proceeds in two

phases: First, a *reward modelling* stage that learns a parameterized reward model r_ϕ . By assuming a Bradley-Terry model [Bradley and Terry, 1952] of preferences, the probability of the data can be expressed as $P(y_w \succ y_l) = \exp r_\phi(y_w, x) / (\exp r_\phi(y_w, x) + \exp r_\phi(y_l, x))$, and subsequently simply optimized over ϕ through the maximum likelihood principle. The second stage of *policy optimization* employs a reinforcement learning algorithm to train the language model against the learned reward. Usually, a KL penalty is introduced between the model and the pre-RL *reference* policy π_{ref} [Jaques et al., 2019, Stiennon et al., 2020] to prevent over-optimization and straying too far from the original policy, resulting in the final objective:

$$\max_{\pi_\theta} \underbrace{\mathbb{E}_{y \sim \pi_\theta, x \sim \mathcal{P}} [r_\phi(y, x)]}_{\text{reward maximization}} - \beta \underbrace{\mathbb{KL}(\pi_\theta, \pi_{ref})}_{\text{regularization}}. \quad (1)$$

Despite success in frontier models [Anthropic, 2023, Gemini-Team, 2023], deep RL has many implementations [Engstrom et al., 2019] and training challenges [Sutton, 1984, Razin et al., 2023] that hinder its adoption. To simplify the whole process, *direct preference optimization* [Rafailov et al., 2023, DPO] aims to forego both the reward modelling and online RL procedure. Rewriting (1) with a decomposition of the KL term into:

$$\max_{\pi_\theta} \mathbb{E}_{y \sim \pi_\theta, x \sim \mathcal{P}} \left[\underbrace{r_\phi(y, x)}_{\text{reward}} + \underbrace{\beta \log \pi_{ref}(y|x)}_{\pi_{ref} \text{ regularization}} \right] + \underbrace{\beta \mathcal{H}(\pi_\theta)}_{\text{policy entropy}}, \quad (2)$$

expresses the problem as an entropy-regularised RL bandit task [Ziebart et al., 2008], for which a known analytical solution exists: $\pi^*(y|x) = Z(x)^{-1} \pi_{ref}(y|x) \exp(\beta^{-1} r_\phi(y, x))$. By rearranging the reward, we can express the task as a binary classification problem based on the reward difference:

$$\min_{\pi_\theta} \mathbb{E}_{(y_w, y_l, x) \sim \mathcal{D}} \left[f \left(\underbrace{\beta \cdot \left(\log \frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)} \right)}_{r_\phi(y_w, x) - r_\phi(y_l, x)} \right) \right]. \quad (3)$$

Here, we define the log ratio difference as $\rho = \log \frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}$. In DPO, the function $f = -\log \sigma$ is derived as the negative log of the sigmoid function given the BT model assumptions. However, Tang et al. [2024] highlighted that more generally we can obtain a recipe for offline preference optimization algorithms by letting $f : \mathbb{R} \rightarrow \mathbb{R}$ be any scalar loss function. For example, setting $f(x) = (x - 1)^2$, the squared loss function [Rosasco et al., 2004] yields IPO [Azar et al., 2023], while employing the max-margin inspired hinge loss [Boser et al., 1992, Cortes and Vapnik, 1995] $f(x) = \max(0, 1 - x)$ produces SLiC [Zhao et al., 2023].

Meta-Optimization for Algorithm Discovery. The goal of *meta-optimization* (optimizing the optimization process) is to uncover novel learning algorithms using a data-driven process. Suppose that an algorithm uses an objective function f^γ to train a model for K iterations, where γ denotes a set of meta-parameters. Meta-optimization searches for an objective that maximizes the expected downstream performance $\max_\gamma \mathbb{E}[\eta(\pi_K) | \text{train}(f^\gamma)]$ where η is a downstream performance metric. Unlike previous methods that rely on a predefined parameterization of γ (e.g., a neural network [Hospedales et al., 2021] or domain-specific language [Alet et al., 2020]), we leverage LLMs to directly propose code-level objective functions in Python. This approach eliminates the need for a carefully designed search space and utilizes the extensive knowledge embedded in the LLM for flexible selection and mutation.

3 LLM-Driven Objective Discovery

Choosing an appropriate objective function is crucial for instilling capabilities into networks. Here, we detail our discovery process facilitated by LLM code-level objective function proposals:

Initial Context Construction. In the initial system prompt, we ‘burn-in’ the LLM using several established objective functions given in code and their corresponding performance. Furthermore, we provide problem details and an example of the output response format as a JSON dictionary.

LLM Querying, Parsing & Output Validation. We query the LLM, parse the response JSON, and run a set of unit tests (e.g. for valid output shapes) before starting a training run. If the parsing or unit tests fail, we resample a new solution after providing the error message as feedback to the LLM.

Performance Evaluation. The proposed objective function is then evaluated based on its ability to optimize a model for a predefined downstream validation task. We refer to the resulting performance metric as η .

Iterative Refinement. By using the performance provided as feedback, the LLM iteratively refines its proposals. In each iteration, the model synthesizes a new candidate loss function, exploring both variations of previously successful formulas and entirely new formulations that might improve upon the existing benchmarks. This iterative process is repeated for a specified number of generations or until convergence when a set of optimal loss functions is observed.

We summarise this general objective discovery process in Figure 1 and is shown in Algorithm 1.

Algorithm 1 LLM-Driven Objective Discovery

- 1: Initialize LLM with established loss functions and their performance *in context*.
 - 2: **repeat** for each generation i
 - 3: LLM proposes a new candidate objective function f_i
 - 4: Run unit tests to check the validity of the candidate and resample if needed.
 - 5: Evaluate the objective function using the performance metric η
 - 6: Update the LLM context with the performance data
 - 7: LLM refines generation strategy based on the feedback
 - 8: **until** convergence criteria are met or maximum generations are reached
-

Small case study: Discovering supervised classification loss functions. Consider the case of supervised classification on the CIFAR-10 dataset as a simple starting example. We train a simple ResNet-18 for 5 epochs using the objectives proposed by GPT-4 [OpenAI, 2023]. After each training run we provide the LLM with the corresponding validation accuracy and query it for the next PyTorch-based [Paszke et al., 2017] candidate objective function.

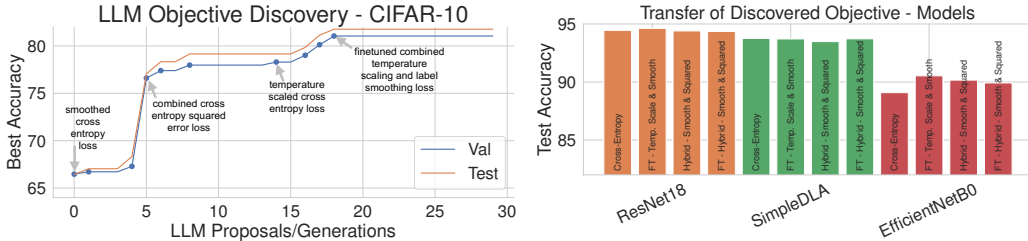


Figure 2: LLM-driven objective discovery for CIFAR-10 classification. **Left.** Performance across LLM-discovery trials. The proposals alternate between exploring new objective concepts, tuning the components, and combining previous insights. **Right.** The best three discovered objectives transfer to different network architectures and longer training runs (100 epochs).

Figure 2 depicts the performance of the proposed objective functions across the discovery process. The different discovered objectives all outperform the standard cross-entropy loss. Interestingly, we observe that the LLM-driven discovery alternates between several different exploration, fine-tuning, and knowledge composition steps: Initially, the LLM proposes a label-smoothed cross-entropy objective. After tuning the smoothing temperature, it explores a squared error loss variant, which improved the observed validation performance. Next, the two conceptually different objectives are combined, leading to another significant performance improvement. Hence, the LLM discovery process does not perform a random search over objectives previously outlined in the literature but instead composes various concepts in a complementary fashion. Furthermore, the discovered objectives also generalize to different architectures and longer training runs. In Appendix D.3 we show that this process of discovery is robust to the choice of sampling temperature and prompt/context construction.

4 Discovering Offline Preference Optimization Objectives

In this section, we run our LLM-driven discovery to automatically generate new state-of-the-art preference optimization algorithms.

4.1 Discovery Task - Multi-turn Dialogue on MT-Bench

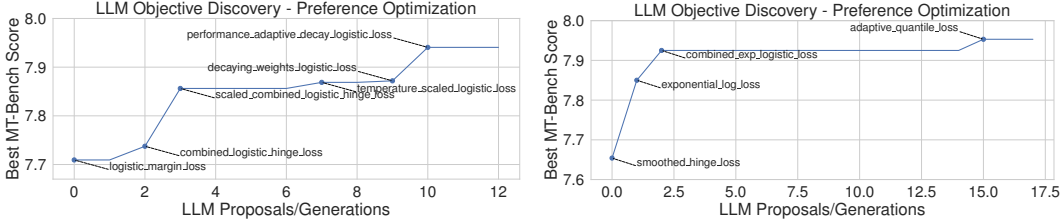


Figure 3: Examples of LLM Objective Discovery improvement across generations. The first and second runs are shown left and right respectively.

In this section we use our LLM-driven discovery method to discover new objective functions f for offline preference optimization, as defined in Section 2 and Equation (3). Specifically, at each generation i , GPT-4 generates PyTorch [Paszke et al., 2017] code of candidate objective function f_i . Each objective function takes as input the variables of $\{\log \pi_\theta(y_w|x), \log \pi_{\text{ref}}(y_w|x), \log \pi_\theta(y_l|x), \log \pi_{\text{ref}}(y_l|x)\}$, and returns a scalar. For each proposed objective f_i , we check if f_i is valid with a unit test.

For each *valid* generated objective function f_i , we finetune an LLM and then collect a performance evaluation score. Specifically, we build on top of the ‘alignment-handbook’ [Tunstall et al., 2023a] repository to finetune our models. Notably, this repository, when using DPO, reproduces ‘Zephyr 7B Gemma’² Tunstall and Schmid [2024], Tunstall et al. [2023b], which at the time of release, achieved state-of-the-art scores on MT-Bench for 7B models. ‘Zephyr 7B Gemma’ first takes gemma-7b [Gemma-Team et al., 2024] and finetunes it on the ‘deita-10k-v0-sft’ dataset [Liu et al., 2023] to produce ‘zephyr-7b-gemma-sft’³. It is then trained on the pairwise preference dataset of ‘Argilla DPO Mix 7K’⁴. When evaluating a new objective function, we replace DPO in this last step with the generated objective function, keeping the same hyperparameters. We show example runs in Figure 12 and provide further experimental details in Appendix B.

Once we have a trained LLM for the proposed objective function f_i , we evaluate that LLM on the popular multi-turn dialogue evaluation benchmark of MT-Bench [Zheng et al., 2024]. This is a multi-turn open-ended question set, which uses GPT-4 to assess the quality of the trained model’s responses, obtaining a high correlation with the popular Chatbot Arena [Zheng et al., 2024]. We provide further evaluation details in Appendix C.

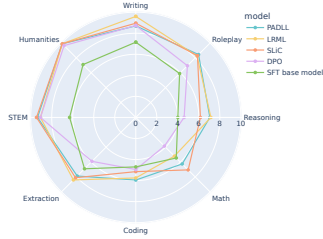


Figure 4: MT-Bench Discov-ered Objective Evaluations

4.2 Discovery Results

After evaluating approximately 100 objective functions, we catalogued the best-performing ones in Table 1. We tabulate the high-level objective forms here and provide the full objective loss functions and their associated code in Appendix E. Moreover, we also plot the best performing sub-task evaluations in Figure 4.

Table 1: **Discovery Task MT-Bench Evaluation Scores for each discovered objective function f .** We provide the baselines first, followed by a dashed line to separate the objective functions that were discovered. We provide details for each discovered objective function in Appendix E.

Name	Full Name	Objective f Function	Score (/ 10) ↑
DPO	Direct Preference Optimization	$\log(1 + \exp(-\beta\rho))$	7.888
DPO*	Official HuggingFace ‘zephyr-7b-gemma’ DPO model	$\log(1 + \exp(-\beta\rho))$	7.810
SLIC	Sequence Likelihood Calibration	$\text{ReLU}(1 - \beta\rho)$	7.881
KTO	Pairwise Kahneman-Tversky Optimization	see [Ethayarajh et al., 2024]	7.603
DBAQL	Dynamic Blended Adaptive Quantile Loss	$\sigma(\sqrt{\text{Var}[\beta\rho/\tau]}) \cdot f_{\text{dpo}}(\beta\rho/0.9) + (1 - \sigma(\sqrt{\text{Var}[\beta\rho/\tau]}) \cdot f_{\text{exp}}(\beta\rho \cdot 0.9)$	7.978
AQL	Adaptive Quantile Loss	$q \cdot f_{\text{dpo}}(\beta\rho) + (1 - q) \cdot f_{\text{slc}}(\beta\rho)$	7.953
PADLL	Performance Adaptive Decay Logistic Loss	$0.9 \cdot (1 - 0.5 \cdot \mathbb{1}[\rho < 0]) \cdot f_{\text{dpo}}(\beta\rho)$	7.941
AQFL	Adaptive Quantile Feedback Loss	$r \cdot f_{\text{dpo}}(\beta\rho) + (1 - r) \cdot f_{\text{slc}}(\beta\rho)$	7.931
CELL	Combined Exponential + Logistic Loss	$0.5 \cdot f_{\text{dpo}}(\beta\rho) + 0.5 \cdot f_{\text{exp}}(\beta\rho)$	7.925
LRML (DiscoPOP)	Log Ratio Modulated Loss	$(1 - \sigma(\beta\rho/\tau)) \cdot f_{\text{dpo}}(\beta\rho) + \sigma(\beta\rho/\tau) \cdot f_{\text{exp}}(\beta\rho)$	7.916
PFL	Policy Focused Loss	$1/2 \cdot f_{\text{dpo}}(\beta\rho) \cdot \mathbb{1}[\pi_w > \pi_r] + 2 \cdot f_{\text{slc}}(\beta\rho) \cdot \mathbb{1}[\pi_w \leq \pi_r]$	7.900

²<https://huggingface.co/HuggingFaceH4/zephyr-7b-gemma-v0.1>

³<https://huggingface.co/HuggingFaceH4/zephyr-7b-gemma-sft-v0.1>

⁴<https://huggingface.co/datasets/argilla/dpo-mix-7k>

5 Held-Out Evaluations

We next validate each of our discovered objective functions (shown in Table 1) on held-out tasks. We find that the Performance Adaptive Decay Loss (PADLL) and the Log Ratio Modulated Loss (LRML) consistently perform well. Because of its unconventional properties and performance, we refer to LRML as our discovered preference optimization, or *DiscoPOP*, algorithm.

We consider three different standard [Rafailov et al., 2023] open-ended text generation tasks each designed to evaluate different properties of the fine-tuned LLM policy π_θ where each LLM policy is trained with one of our discovered objective functions f on a preference dataset $\mathcal{D} = \{(x^i, y_w^i, y_l^i)\}_{i=1}^N$.

5.1 Single-turn Dialogue - Alpaca Eval 2.0

We evaluate the trained models on Alpaca Eval 2.0, [Li et al., 2023, Dubois et al., 2023, 2024]. This is a single-turn dialogue LLM-based automatic evaluation using GPT-4 to assess the win rate of the trained LLM policy’s completion compared to the of the underlying SFT base model. Alpaca Eval 2.0⁵, has been validated against 20K human annotations, and aims to reduce the length bias of Alpaca Eval 1.0; where using length controlled (LC) Alpaca Eval shows a correlation with Chatbot Area of 0.98, making it a popular benchmark with the highest correlation to Chatbot Arena [Dubois et al., 2024]. We also detail task training details in Appendix B.1.

Table 2: **Alpaca Eval 2.0 - Held Out Single Turn Dialogue Task.** Win rate of the discovered objective functions f evaluated on the Alpaca Eval 2.0 task against either GPT-4 or the SFT base model. Some of the discovered objective functions outperform the baselines, with the best bolded. We detail evaluation and error bars in Appendix C. We have highlighted the best scores with overlapping the standard errors.

Function	Win Rate (%) \uparrow	Win Rate - LC (%) \uparrow	Win Rate (%) \uparrow	Win Rate - LC (%) \uparrow
	vs. GPT-4		vs. SFT Checkpoint	
DPO	11.23 \pm 0.97	12.81 \pm 0.66	78.72 \pm 1.26	63.34 \pm 0.30
DPO*	11.99 \pm 1.00	14.73 \pm 0.71	75.75 \pm 1.31	59.88 \pm 0.41
SLiC	10.67 \pm 0.94	13; 16 \pm 0.69	75.05 \pm 1.34	59.67 \pm 0.42
KTO	12.57 \pm 1.00	13.58 \pm 0.67	78.81 \pm 1.25	62.76 \pm 0.31
DBAQL	10.68 \pm 0.92	11.41 \pm 0.57	72.06 \pm 1.42	54.40 \pm 0.38
AQL	11.11 \pm 0.96	13.63 \pm 0.68	76.34 \pm 1.30	60.94 \pm 0.36
PADLL	14.07 \pm 1.04	14.89 \pm 0.66	81.10 \pm 1.21	64.14 \pm 0.28
AQFL	13.63 \pm 1.05	15.55 \pm 0.71	79.32 \pm 1.23	64.41 \pm 0.34
CELL	10.27 \pm 0.93	12.26 \pm 0.61	71.75 \pm 1.39	57.48 \pm 0.34
LRML	13.21 \pm 1.02	14.78 \pm 0.67	79.27 \pm 1.24	65.18 \pm 0.32
PFL	8.15 \pm 0.83	10.67 \pm 0.57	68.27 \pm 1.44	56.14 \pm 0.43

We provide the Alpaca Eval 2.0 results in Table 2. As reference policies, we used GPT-4 for absolute comparison and the SFT-trained model for relative comparison. We observe that the discovered LRML (DiscoPOP), PADLL, and AQFL functions outperform the baselines and other discovered losses on the normal and length-controlled win rates. The differences in scores among these top-performing losses are not significant, except for the LC win rate against the SFT reference model, where DiscoPOP performs best.

5.2 Summarization (TL;DR)

We train an LLM policy to, given a forum post on Reddit x , generate a summarization y of the main points. We finetune ‘zephyr-7b-gemma-sft’ using 10% of the Reddit TL;DR summarization preference dataset [Völske et al., 2017] on each of the baseline and discovered objective functions. As a reference model, we again use ‘zephyr-7b-gemma-sft’. Further details on the training pipeline are outlined in Appendix B.2. To evaluate the quality of the summaries, we make use of the Alpaca Eval 2.0 library with a custom evaluation dataset existing of 694 test samples from the TL;DR dataset and a custom GPT-4 annotator template as described in Rafailov et al. [2023]. For additional details regarding the summarization evaluation see Appendix C.3.

⁵https://github.com/tatsu-lab/alpaca_eval

In Table 3 the PADLL loss and DPO loss perform best, with little difference from each other, on the summarization task in three out of four metrics. Additionally, the LRML - DiscoPOP function achieves scores slightly below the top performers, especially in the length-controlled win rates. In contrast to the single-turn dialogue task, the AQFL loss does not achieve high scores in the held-out evaluation.

Table 3: **TL;DR - Held Out Summarization Task** Win-rate of various preference optimization functions in the summarization task was evaluated with the Alpaca Eval 2.0 calculations, against a subset of the test set (694 samples). The baseline outputs are the human-generated preferences, and the model after SFT (see Appendix C for details). Note that the standard error in the LC win-rate has been rounded down because of values < 0.001 . We have highlighted the scores with means overlapping the standard error of the best score.

Function	Win Rate (%) \uparrow vs. Human Preference	Win Rate - LC (%) \uparrow	Win Rate (%) \uparrow vs. SFT Checkpoint	Win Rate - LC (%) \uparrow
DPO	88.27 \pm 1.07	82.82 \pm 0.00	54.38 \pm 1.52	54.64 \pm 0.00
SLiC	83.02 \pm 1.29	63.41 \pm 0.00	53.03 \pm 1.52	54.11 \pm 0.00
KTO	85.34 \pm 1.18	80.26 \pm 0.00	51.15 \pm 1.54	50.0 \pm 0.00
DBAQL	84.71 \pm 1.21	78.68 \pm 0.00	52.55 \pm 1.52	55.14 \pm 0.00
AQL	81.87 \pm 1.32	68.89 \pm 0.00	46.00 \pm 1.54	50.0 \pm 0.00
PADLL	88.54 \pm 1.05	76.13 \pm 0.00	55.34 \pm 1.52	55.64 \pm 0.00
AQFL	85.03 \pm 1.22	76.23 \pm 0.00	49.56 \pm 1.53	50.38 \pm 0.00
CELL	86.33 \pm 1.14	73.72 \pm 0.00	50.35 \pm 1.52	51.90 \pm 0.00
LRML	87.63 \pm 1.10	81.88 \pm 0.00	53.46 \pm 1.52	55.10 \pm 0.00
PFL	79.84 \pm 1.35	69.23 \pm 0.00	44.12 \pm 1.52	44.57 \pm 0.00

5.3 Positive sentiment generation (IMDb)

In this task, we train an LLM policy to generate movie review completions y with positive sentiment, where x is a prompt at the start of a movie review from the IMDb dataset [Maas et al., 2011]. We start with a GPT-2 [Radford et al., 2019] model, which had supervised fine-tuning on the IMDb dataset, and we perform preference optimization using the baseline and discovered objective loss functions. Details of the training implementations can be found in Appendix B.3. Inspired by Rafailov et al. [2023]’s experiments, we calculate the model rewards through a pre-trained sentiment classifier, which we use as a proxy for ground truth, as well as the KL-Divergence of the trained model and the reference model. Appendix C.4 provides further details into the evaluation for this task.

We provide results of models with converging β values in Figure 5 for LRML compared against DPO and SLiC, displaying the model rewards against the KL-Divergence to the reference model. In Figure 5a, the LRML-trained text generator outperforms the DPO model in terms of rewards and KL-divergence with low β values (0.025, 0.05, 0.1). At higher β values (0.5 and 1.0) both methods show trends of increased KL-Divergence and lower rewards, but generally, LRML maintains a higher reward than DPO. In Figure 5b, we note that LRML slightly outperforms DPO, SLiC, AQFL, and PADLL at $\beta \in \{0.05, 0.1\}$ in terms of reward. For larger β values (0.5 and 1.0), LRML shows similar trends of increased KL-Divergence and rewards like the other objective functions. A more detailed comparison between the individual discovered losses and the baselines can be found in Appendix Figure 8.

6 Analysis of DiscoPOP

We list all our discovered objectives in Table 1, as well as the code and mathematical representations in Appendix E. In this section, we now analyze the Log Ratio Modulated Loss, which we define as the DiscoPOP loss function, as it performs consistently high across the held-out evaluation tasks, and we provide some intuitive understanding of how it outperforms the existing state-of-the-art objectives.

6.1 Log Ratio Modulated Loss (DiscoPOP)

The Log Ratio Modulated Loss is a dynamically weighted sum of the logistic loss (as used in DPO) and the exponential loss. The weight of each is determined through a sigmoid calculation of the

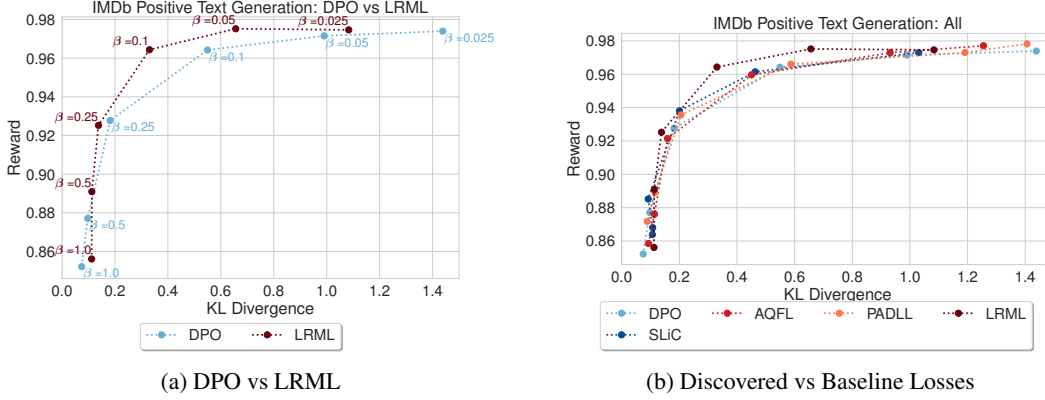


Figure 5: Frontiers of expected reward vs KL divergence for converging models for the LRML against DPO and SLiC objective function. The rewards and KL-divergence values are averaged over 10 generations with different seeds. The sweep is done over $\beta \in \{0.025, 0.05, 0.1, 0.25, 0.5, 1.0\}$. The optimal point is the top left corner, where the perfect reward is achieved with minimal divergence from the reference model.

difference of log-ratios (ρ). Mathematically, the LRML function can be described with a temperature parameter $\tau = 0.05$ as follows:

$$f_{lrml}(\beta\rho) = (\sigma(\beta\rho/\tau) - 1) \cdot f_{dpo}(\beta\rho) + \sigma(\beta\rho/\tau) \cdot f_{exp}(\beta\rho) \quad (4)$$

$$= (1 - \sigma(\beta\rho/\tau)) \cdot \log(1 + \exp(-\beta\rho)) + \sigma(\beta\rho/\tau) \cdot \exp(-\beta\rho) \quad (5)$$

If the difference of log ratios is zero ($\rho = 0$), which is at the start of the training when the model policy π_θ is equal to the reference policy π_{ref} , then the loss is equally balanced between the logistic and exponential loss. If $\rho \rightarrow \infty$, the model policy diverges from the reference policy and chosen outputs are preferred, then the exponential term dominates. This emphasizes larger differences more strongly. On the other hand, if $\rho \rightarrow -\infty$, the model policy diverges from the reference policy, and rejected outputs are preferred. In this case, the logistic loss can handle moderate differences well. The baseline objective losses and the LRML, the PADLL, and the AQFL functions are displayed in Figure 6, including their gradients. Surprisingly, the DiscoPOP function has a non-convex segment and negative gradients at the starting point $\rho = 0$. This is potentially helpful for introducing a curriculum or for stochasticity. Additional results and analysis of the discovered loss function can be found in Appendix D.6.

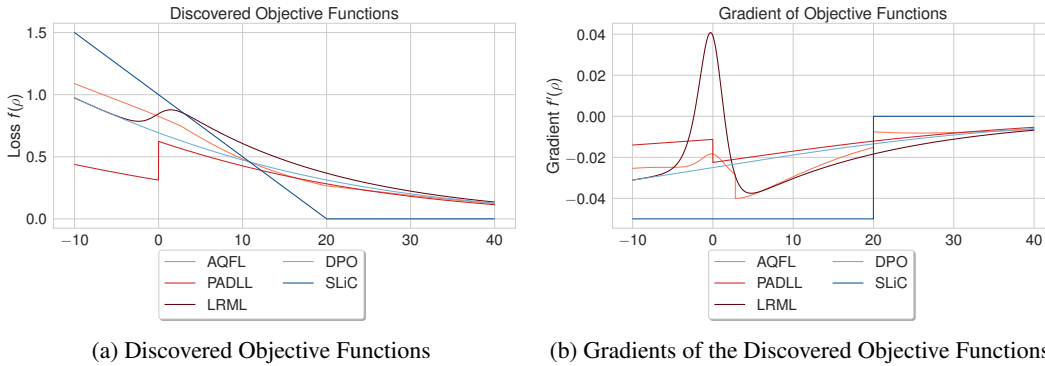


Figure 6: Figure 6a: Baseline objective functions DPO and SLiC, and the discovered ones, LRML, AQFL, and PADLL. Figure 6b: gradients of the objectives as a function of ρ and with fixed $\beta = 0.05$.

6.2 Limitations of DiscoPOP

While performing very well on single-turn text generation and text summarization, we observed during the IMDb experiment that LRML struggles to converge when β is too low ($\beta \leq 0.01$) or too high ($\beta \geq 2.5$), likely because $\beta \neq 0.05$ was never seen or used during the discovery process.

In Figure 9 and Figure 10 of the Appendix, we plot the LRML objective function for $\beta \in \{0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5, 5\}$ against DPO. When β is high, the DiscoPOP objective function takes the form of the DPO log sigmoid loss. During training on $\beta = 0.01$, we observed that DiscoPOP gets stuck in generating predominantly negative reviews (resulting in a reward score of ~ 0.15). We hypothesize that the loss is stuck in the local minima to the left with a negative difference in log ratios. While training with $\beta \in \{2.5, 5.0\}$, we observed that the model collapsed after a sharp spike in the loss and subsequently having loss value 0 and NaN outputs. This is potentially due to a large gradient in the non-convex part, which could be amended with gradient clipping.

7 Related Work

Evolution and Search with Large Language Models. LLMs provide a fast and automated way to create multiple candidate solutions for a problem stated in natural language [Song et al., 2024], which makes them powerful tools for driving population-based search procedures. Various recent works have applied this approach to coding problems [Romera-Paredes et al., 2024], neural architecture search [Chen et al., 2024a, Holt et al., 2024b], virtual robotic design settings [Lehman et al., 2023], reward functions [Ma et al., 2023, Yu et al., 2023], and algorithm heuristics [Liu et al., 2024]. Finally, recently LLMs have shown to be capable of acting as recombination operators for black-box optimization with Evolution Strategies [Lange et al., 2024] and for Quality-Diversity approaches [Lim et al., 2024].

Automated Discovery for Machine Learning. There are many other approaches to automating the discovery of generalizable machine learning algorithms. Some prior works explore the space of ML functions using genetic algorithms and a hand-crafted domain-specific language for reinforcement learning algorithms [Co-Reyes et al., 2021], curiosity algorithms [Alet et al., 2020], and optimizers [Chen et al., 2024b]. Other works instead parameterize a transferrable objective function using neural networks and optimize them with evolution strategies or meta-gradients. For example, Lu et al. [2022], Jackson et al. [2024b], Houthoof et al. [2018], Alfano et al. [2024], Kirsch et al. [2019], Oh et al. [2020], Jackson et al. [2024a] discover policy optimization objectives, Metz et al. [2022] evolves neural network optimizers, and Lange et al. [2023b,a] evolve blackbox optimizers. Moreover, automatically discovering closed-form functions (i.e., symbolic regression), works exist that leverage RL [Petersen et al., 2020], gradient descent [Kacprzyk et al., 2024], RL with evolution strategies [Mundhenk et al., 2021], pre-training transformers [Biggio et al., 2021] and hybrid combinations of pre-training transformers, which are further refined with RL and evolution strategies [Holt et al., 2023b].

Preference Optimization Algorithms. While the reduction to supervised learning makes DPO and alternatives easier to use, other approaches have sought to simplify the RL step, including using variants of REINFORCE [Ahmadian et al., 2024, Gemma-Team et al., 2024] as well as more fine-grained feedback [Wu et al., 2024] through preferences over individual steps in the reasoning process [Uesato et al., 2022, Lightman et al., 2023] or reward redistribution [Chan et al., 2024]. Others use iterative offline training interleaved with sampling from the policy model and obtaining a preference ranking from themselves [Xu et al., 2023], another judge LLM [Guo et al., 2024], or an oracle [Swamy et al., 2024].

8 Conclusion

Summary. In this paper, we proposed and used LLM-driven objective discovery to generate novel offline preference optimization algorithms. Specifically, we were able to discover high-performing preference optimization losses that achieve strong performance across held-out evaluation tasks, with the highest performing providing new insights into what an optimal objective may need to possess, such as being a blend of logistic and exponential losses and possibly being non-convex.

Limitations & Future work. There are multiple limitations to our current approach. First, we have only scratched the surface of how to generate LLM objective proposals effectively. Initial exploratory experiments using techniques such as temperature sampling or worst-to-best performance sorting in the context did not yield significant improvements. But one could imagine leveraging more information about the training runs and automatically tuning instruction prompt templates. E.g. by providing entire learning curve plots to a Visual Language Model (see Figure 13) or by meta-meta-optimizing [Lu et al., 2023] the LLM prompt. Second, the highest-performing loss re-purposed β

in the traditional sense, making it affect the functional behavior and the KL penalty of the model with respect to the base model. This motivates future work to study different forms, with perhaps multiple floating point parameters in the form, that each could be tuned separately. Although we provided an initial analysis sweep over this one single parameter and observed some instances of the functional behavior leading to instability of training the model, a further multi-parameter analysis, reformulating the objective, would be beneficial for future work. Finally, our work uses closed-source models (GPT-4) to generate code, which limits reproducibility and is costly to run. Future work could use the produced models *themselves* to generate code, resulting in code-level self-improvement.

Broader Impact and Ethical Considerations. This paper presents an LLM-driven discovery in-context learning pipeline that is used to generate better-performing novel offline preference optimization algorithms. However, misuse of the pipeline as a tool or training an LLM to produce undesirable, unethical, or harmful outputs could be possible by a user. Furthermore, due to the use of LLMs and training of LLMs, the outputs are susceptible to hallucinations, motivating all outputs of the LLMs to always have a content filter applied to the outputs. Finally, this work takes a small step towards code-level self-improvement in language models, which could potentially result in unintended behaviors.

Acknowledgments and Disclosure of Funding

This work was supported by Azure sponsorship credits granted by Microsoft’s AI for Good Research Lab and by Microsoft’s Accelerate Foundation Models Academic Research initiative. The hardware used for training was sponsored by GoodAI. SH is funded by AstraZeneca. AJC is funded by a Microsoft Research and EPSRC ICASE scholarship award. CL and RTL were supported by Sakana AI at the time of this work. The code can also be accessed at <https://github.com/samholt/DiscoPOP>.

References

- Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.
- Ferran Alet, Martin F Schneider, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Meta-learning curiosity algorithms. *arXiv preprint arXiv:2003.05325*, 2020.
- Carlo Alfano, Sebastian Towers, Silvia Sapora, Chris Lu, and Patrick Rebeschini. Meta-learning the mirror map in policy mirror descent. *arXiv preprint arXiv:2402.05187*, 2024.
- Anthropic. Model card and evaluations for claude models, 2023. URL <https://www-files.anthropic.com/production/images/Model-Card-Claude-2.pdf>.
- Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. A general theoretical paradigm to understand learning from human preferences. *arXiv preprint arXiv:2310.12036*, 2023.
- Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pages 936–945. Pmlr, 2021.
- Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.
- Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.
- Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650, 2021.

- Alex J Chan, Hao Sun, Samuel Holt, and Mihaela van der Schaar. Dense reward for free in reinforcement learning from human feedback. *arXiv preprint arXiv:2402.00782*, 2024.
- Angelica Chen, David Dohan, and David So. Evoprompting: Language models for code-level neural architecture search. *Advances in Neural Information Processing Systems*, 36, 2024a.
- Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in Neural Information Processing Systems*, 36, 2024b.
- Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.
- John D Co-Reyes, Yingjie Miao, Daiyi Peng, Esteban Real, Sergey Levine, Quoc V Le, Honglak Lee, and Aleksandra Faust. Evolving reinforcement learning algorithms. *arXiv preprint arXiv:2101.03958*, 2021.
- Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.
- Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. Ultrafeedback: Boosting language models with high-quality feedback. *arXiv preprint arXiv:2310.01377*, 2023.
- Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaFarm: A simulation framework for methods that learn from human feedback, 2023.
- Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaEval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.
- Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep RL: A case study on PPO and TRPO. In *International conference on learning representations*, 2019.
- Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. KTO: Model alignment as prospect theoretic optimization, 2024.
- Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. Real-toxicityprompts: Evaluating neural toxic degeneration in language models. *arXiv preprint arXiv:2009.11462*, 2020.
- Google DeepMind Gemini-Team. Gemini: A family of highly capable multimodal models, 2023.
- Gemma-Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on Gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.
- Shangmin Guo, Biao Zhang, Tianlin Liu, Tianqi Liu, Misha Khalman, Felipe Llinares, Alexandre Rame, Thomas Mesnard, Yao Zhao, Bilal Piot, et al. Direct language model alignment from online AI feedback. *arXiv preprint arXiv:2402.04792*, 2024.
- Samuel Holt, Alihan Hüyük, Zhaozhi Qian, Hao Sun, and Mihaela van der Schaar. Neural Laplace control for continuous-time delayed systems. In *International Conference on Artificial Intelligence and Statistics*, pages 1747–1778. PMLR, 2023a.
- Samuel Holt, Zhaozhi Qian, and Mihaela van der Schaar. Deep generative symbolic regression. In *The Eleventh International Conference on Learning Representations*, 2023b. URL <https://openreview.net/forum?id=o7koEEMA1bR>.
- Samuel Holt, Alihan Hüyük, and Mihaela van der Schaar. Active observing in continuous-time control. *Advances in Neural Information Processing Systems*, 36, 2024a.

- Samuel Holt, Tennison Liu, and Mihaela van der Schaar. Automatically learning hybrid digital twins of dynamical systems. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b. URL <https://openreview.net/forum?id=S0si0bSdU2>.
- Samuel Holt, Max Ruiz Luyten, and Mihaela van der Schaar. L2MAC: Large language model automatic computer for extensive code generation. In *The Twelfth International Conference on Learning Representations*, 2024c. URL <https://openreview.net/forum?id=EhrzQwsV4K>.
- Samuel Holt, Zhaozhi Qian, Tennison Liu, Jim Weatherall, and Mihaela van der Schaar. Data-driven discovery of dynamical systems in pharmacology using large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024d.
- Samuel I Holt, Zhaozhi Qian, and Mihaela van der Schaar. Neural laplace: Learning diverse classes of differential equations in the laplace domain. In *International Conference on Machine Learning*, pages 8811–8832. PMLR, 2022.
- Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9): 5149–5169, 2021.
- Rein Houthoofd, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, OpenAI Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *Advances in Neural Information Processing Systems*, 31, 2018.
- Matthew T Jackson, Minqi Jiang, Jack Parker-Holder, Risto Vuorio, Chris Lu, Greg Farquhar, Shimon Whiteson, and Jakob Foerster. Discovering general reinforcement learning algorithms with adversarial environment design. *Advances in Neural Information Processing Systems*, 36, 2024a.
- Matthew Thomas Jackson, Chris Lu, Louis Kirsch, Robert Tjarko Lange, Shimon Whiteson, and Jakob Nicolaus Foerster. Discovering temporally-aware reinforcement learning algorithms. *arXiv preprint arXiv:2402.05828*, 2024b.
- Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.
- Krzysztof Kacprzyk, Samuel Holt, Jeroen Berrevoets, Zhaozhi Qian, and Mihaela van der Schaar. ODE discovery for longitudinal heterogeneous treatment effects inference. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=pxI5IPeWgW>.
- Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.
- Robert Lange, Tom Schaul, Yutian Chen, Chris Lu, Tom Zahavy, Valentin Dalibard, and Sebastian Flennerhag. Discovering attention-based genetic algorithms via meta-black-box optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 929–937, 2023a.
- Robert Lange, Tom Schaul, Yutian Chen, Tom Zahavy, Valentin Dalibard, Chris Lu, Satinder Singh, and Sebastian Flennerhag. Discovering evolution strategies via meta-black-box optimization. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pages 29–30, 2023b.
- Robert Tjarko Lange, Yingtao Tian, and Yujin Tang. Large language models as evolution strategies. *arXiv preprint arXiv:2402.18381*, 2024.
- Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. Evolution through large models. In *Handbook of Evolutionary Machine Learning*, pages 331–366. Springer, 2023.
- Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaEval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 2023.

- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.
- Bryan Lim, Manon Flageat, and Antoine Cully. Large language models as in-context ai generators for quality-diversity. *arXiv preprint arXiv:2404.15794*, 2024.
- Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Forty-first International Conference on Machine Learning*, 2024.
- Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. *arXiv preprint arXiv:2312.15685*, 2023.
- Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning*, pages 22631–22648. PMLR, 2023.
- Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017. URL <https://api.semanticscholar.org/CorpusID:53592270>.
- Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468, 2022.
- Chris Lu, Sebastian Towers, and Jakob Foerster. Arbitrary order meta-learning with simple population-based evolution. In *ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference*. MIT Press, 2023.
- Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.
- Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.
- Luke Metz, James Harrison, C Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, et al. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022.
- T Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P Santiago, Daniel M Faissol, and Brenden K Petersen. Symbolic regression via neural-guided genetic programming population seeding. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, pages 24912–24923, 2021.
- Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado P van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070, 2020.
- OpenAI. Gpt-4 technical report, 2023.
- Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
- Brenden K Petersen, Mikel Landajuela Larma, Terrell N Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2020.

- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.
- Noam Razin, Hattie Zhou, Omid Saremi, Vimal Thilak, Arwen Bradley, Preetum Nakkiran, Joshua Susskind, and Etai Littwin. Vanishing gradients in reinforcement finetuning of language models. *arXiv preprint arXiv:2310.20703*, 2023.
- Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.
- Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural computation*, 16(5):1063–1076, 2004.
- Xingyou Song, Yingtao Tian, Robert Tjarko Lange, Chansoo Lee, Yujin Tang, and Yutian Chen. Position paper: Leveraging foundational models for black-box optimization: Benefits, challenges, and future directions. *arXiv preprint arXiv:2405.03547*, 2024.
- Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.
- Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. University of Massachusetts Amherst, 1984.
- Gokul Swamy, Christoph Dann, Rahul Kidambi, Zhiwei Steven Wu, and Alekh Agarwal. A minimaximalist approach to reinforcement learning from human feedback. *arXiv preprint arXiv:2401.04056*, 2024.
- Yunhao Tang, Zhaohan Daniel Guo, Zeyu Zheng, Daniele Calandriello, Rémi Munos, Mark Rowland, Pierre Harvey Richemond, Michal Valko, Bernardo Ávila Pires, and Bilal Piot. Generalized preference optimization: A unified approach to offline alignment. *arXiv preprint arXiv:2402.05749*, 2024.
- Lewis Tunstall and Philipp Schmid. Zephyr 7b gemma. <https://huggingface.co/HuggingFaceH4/zephyr-7b-gemma-v0.1>, 2024.
- Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Shengyi Huang, Kashif Rasul, Alexander M. Rush, and Thomas Wolf. The alignment handbook. <https://github.com/huggingface/alignment-handbook>, 2023a.
- Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. Zephyr: Direct distillation of lm alignment, 2023b.
- Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.
- Michael Völske, Martin Potthast, Shahbaz Syed, and Benno Stein. Tl; dr: Mining reddit to learn automatic summarization. In *Proceedings of the Workshop on New Frontiers in Summarization*, pages 59–63, 2017.
- Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, and Nathan Lambert. TRL: Transformer Reinforcement Learning. URL <https://github.com/huggingface/trl>.

- Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A Smith, Mari Ostendorf, and Hannaneh Hajishirzi. Fine-grained human feedback gives better rewards for language model training. *Advances in Neural Information Processing Systems*, 36, 2024.
- Jing Xu, Andrew Lee, Sainbayar Sukhbaatar, and Jason Weston. Some things are more cringe than others: Preference optimization with the pairwise cringe loss. *arXiv preprint arXiv:2312.16682*, 2023.
- Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.
- Yao Zhao, Rishabh Joshi, Tianqi Liu, Misha Khalman, Mohammad Saleh, and Peter J Liu. Slic-hf: Sequence likelihood calibration with human feedback. *arXiv preprint arXiv:2305.10425*, 2023.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2024.
- Zhanhui Zhou, Zhixuan Liu, Jie Liu, Zhichen Dong, Chao Yang, and Yu Qiao. Weak-to-strong search: Align large language models via searching over small language models. *arXiv preprint arXiv:2405.19262*, 2024.
- Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Appendix

Table of Contents

A LLM-Driven Objective Discovery Implementation Details	17
A.1 Prompts	17
B Training Details	19
B.1 Discovery Task - Single-turn Dialogue	19
B.2 TL;DR Summarization	19
B.3 IMDb Positive Text Generation	20
C Evaluation Metrics	20
C.1 MT-Bench	20
C.2 Alpaca Eval	21
C.3 TL;DR Summarization Win-Rate	21
C.4 IMDb Rewards vs KL-Divergence	21
D Additional Results	22
D.1 Frontiers of Expected Reward vs KL Divergence	22
D.2 Loss Sweeps for Different Beta Parameters	23
D.3 Discovery Robustness with respect to LLM Hyperparameters	24
D.4 LLM-Driven Discovery Analysis	24
D.5 Visual Language Models for Objective Discovery	24
D.6 Additional Analysis of DiscoPOP	25
E Discovered Objective Functions	25
E.1 DBAQL: Dynamic Blended Adaptive Quantile Loss	26
E.2 AQL: Adaptive Quantile Loss	27
E.3 PADLL: Performance Adaptive Decay Logistic Loss	27
E.4 AQFL: Adaptive Quantile Feedback Loss	28
E.5 CELL: Combined Exponential + Logistic Loss	28
E.6 LRML: Log Ratio Modulated Loss	29
E.7 PFL: Policy Focused Loss	29
F Extended Related Work	30
G Full Run Log	31

A LLM-Driven Objective Discovery Implementation Details

A.1 Prompts

We use the following system prompt to generate the model responses:

```
You are a machine learning researcher who is testing out different
RLHF loss functions. When you respond, output a JSON where the
first key ("thought") corresponds to your thought process when
designing the next function. The second key ("name") corresponds
to the name of your next function. Finally, the last key ("code")
corresponds to the exact python code that you would like to try.
Here is an example:

{
  "thought": "Based on the previous outputs, I should try the direct
  preference optimization algorithm.",
  "name": "dpo",
  "code": "def sigmoid_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
  ) -> torch.FloatTensor:
    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.beta * logits)
    return losses"
}
```

You are deeply familiar with binary classification losses from the literature. Be creative and reference prior literature when possible.

You must use the exact function interface used above. Feel free to define extra hyperparameters within your function as constants. Do not make them attributes of self.

Note that 'self.beta = 0.05'.

RLHF loss functions train on a dataset of pairs of preferred and rejected completions. 'policy_chosen_logps' refers to the policy's log probabilities of the preferred completion, and 'policy_rejected_logps' refers to the policy's log probabilities of the rejected completion. 'reference_chosen_logps' and 'reference_rejected_logps' refer to the same for the reference (base) model.

The user will then return to you a fitness that corresponds to the performance of the resulting model on a downstream task. Your goal is to maximize performance.

We then provide the first user prompt as such:

```
Here are some results we've obtained:

[
  {
    "code": "
def logistic_log_loss(
  self,
  policy_chosen_logps: torch.FloatTensor,
```

```

    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    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.beta * logits)
    return losses
    ",
    "fitness": 7.8875
},
{
    "code": "
def hinge_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    losses = torch.relu(1 - self.beta * logits)
    return losses
    ",
    "fitness": 7.88125
},
{
    "code": "
def ipo_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    losses = (logits - 1 / (2 * self.beta)) ** 2
    return losses
    ",
    "fitness": 7.84
},
{
    "code": "
def kto_pair_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    chosen_KL = (policy_chosen_logps - reference_chosen_logps).mean().
    clamp(min=0)
    rejected_KL = (policy_rejected_logps - reference_rejected_logps).
    mean().clamp(min=0)

    chosen_logratios = policy_chosen_logps - reference_chosen_logps
    rejected_logratios = policy_rejected_logps -
    reference_rejected_logps
    # As described in the KTO report, the KL term for chosen (rejected
    ) is estimated using the rejected (chosen) half.

```

```

    losses = torch.cat(
        (
            1 - F.sigmoid(self.beta * (chosen_logratios - rejected_KL)
        ),
            1 - F.sigmoid(self.beta * (chosen_KL - rejected_logratios)
        ),
    ),
    0,
)
return losses
",
"fitness": 7.603125
}
]

Please generate the next one.

```

Upon testing the generated code, if an error is encountered, we provide the following prompt, where ‘error’ is the text containing the system error:

```

Code not valid. Error:
{error}
Please generate the next one.

```

Upon successful completion, we return the following user prompt, where ‘val’ is the MT-Bench score:

```

Fitness: {val}.
Please generate the next one.

```

B Training Details

B.1 Discovery Task - Single-turn Dialogue

For each valid generated objective function f_i , we use it to train an LLM and then collect a performance evaluation score. Specifically, we follow the same process when training and evaluating all objective functions, starting with a pre-trained supervised fine-tuned (SFT) 7 billion gemma model of ‘zephyr-7b-gemma-sft’ This is a 7 billion base version gemma [Gemma-Team et al., 2024] model supervised-fine-tuned on the ‘deita-10k-v0-sft’ dataset [Liu et al., 2023]. Starting with this model, we train it on the pairwise preference dataset of ‘Argilla DPO Mix 7K’; which attempts to create a high-quality preference dataset by filtering only highly rated chosen responses from the datasets of a multi-turn dataset, instruction following dataset [Longpre et al., 2023] and a diverse preference dataset that covers truthfulness, honesty and helpfulness [Cui et al., 2023]. For each training run, we trained all the parameters of the starting model, using a fixed $\beta = 0.05$. We used the same fixed hyper-parameters for all training runs unless explicitly noted. Specifically, we used a learning rate of $5e-7$, bfloat16 floating-point format, two epochs, a batch size per device of two, a gradient accumulation step of 8, a cosine learning rate scheduler, and AdamW optimization algorithm [Loshchilov and Hutter, 2017]. We use the popular TRL transformers library [von Werra et al.], adapting the offline preference optimization objective function to train all models. The models were trained on 8 Nvidia A100 GPUs. An individual training run takes approximately 30 minutes. We provide training and evaluation statistics for discovered objective functions in Figure 7. We also provide the equivalent code implementation at <https://github.com/vanderschaarlab/DiscoPOP>.

B.2 TL;DR Summarization

To determine if the discovered objective functions generalize well also to other tasks, we use them to preference optimize an LLM for text summarization. Specifically, we start again with a pre-trained supervised fine-tuned (SFT) 7 billion gemma model of ‘zephyr-7b-gemma-sft’, and we optimized it with the objective function f_i on a subsample of the Reddit TL;DR summarization preference dataset

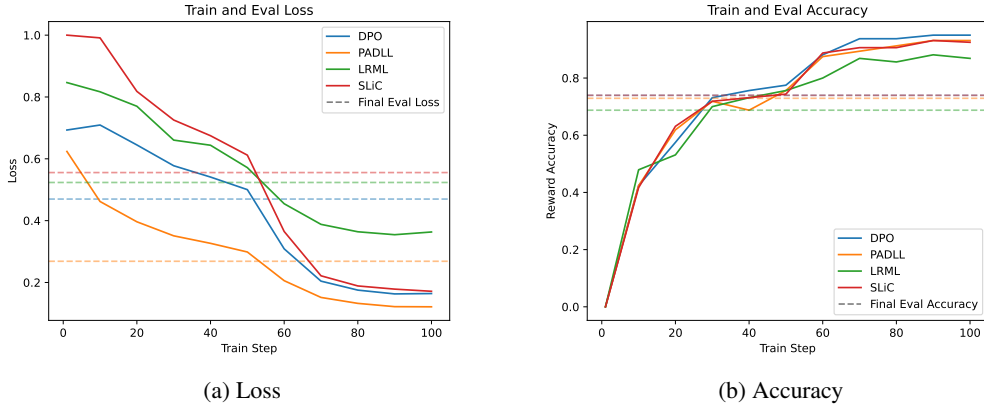


Figure 7: Training and eval statistics of DPO, SLiC, PADLL, and LRML. The losses are not directly comparable to each other, as they are calculated differently for each model. Interestingly, eval results are not strongly correlated with the downstream MT-Bench scores, as LRML achieves the worst accuracy.

[Völske et al., 2017]⁶. More precisely we use the first 10% of the dataset for preference optimization, which amounts to around 8’000 training samples. During training the hyperparameters are kept the same as in the single-turn dialogue task, explained in subsection B.1, except that LLMs were trained 4 Nvidia A100 GPUS using a gradient accumulation step of 16. An individual training run takes approximately 1.5 hours.

B.3 IMDb Positive Text Generation

Another popular generalization task for preference optimization [Rafailov et al., 2023] is to fine-tune a small LLM to generate positive text for movie review, based on the IMDb sentiment dataset [Maas et al., 2011]⁷ following the the work of Zhou et al. [2024]. As a starting model, we use a GPT2 model [Radford et al., 2019], that was supervised fine-tuned on the IMDb dataset⁸. Subsequently, we apply the baseline and discovered objective function f_i for preference optimization. The goal of the LLM is to give a short prompt of 2-8 tokens, which indicate the start of a movie review, to generate a positive review. As we are interested in the effect of β on the rewards and KL-Divergence, we train the objective functions over a sweep of $\beta \in \{0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5, 5\}$. Every LLM is trained for three epochs, using the AdamW optimizer, with an initial learning rate of $5.0e-5$, a warm-up scheduler of 0.1, and a cosine learning rate scheduler. The models are trained on 4 Nvidia A100 GPUs, using a gradient accumulation step of 8, and a batch size per device of 2. The training takes around 30 minutes.

C Evaluation Metrics

C.1 MT-Bench

To assess the fitness of the discovered preference optimization loss function during the discovery phase, we evaluate the trained LLMs on the MT-Bench [Zheng et al., 2024] benchmark. The evaluation benchmark consists of 80 high-quality multi-turn questions from various disciplines. The goal is to assess LLM’s ability to follow instructions and keep the flow of a conversation. A larger LLM, in our case GPT-4, is then used as a judge to score the quality of the answers with a number from 0 (lowest) to 10 (highest). Scores are given based on the quality of the LLM’s first-turn answer (single-turn) and first and second answers (multi-turn). Finally, the MT-Bench score is the average

⁶https://huggingface.co/datasets/CarperAI/openai_summarize_comparisons

⁷https://huggingface.co/datasets/ZHZisZZ/imdb_preference

⁸<https://huggingface.co/lvwerra/gpt2-imdb>

of single-turn and multi-turn scores. For answer generation and evaluation, we used the FastChat library⁹ and its standard sampling and temperature parameters, provided by Zheng et al. [2024].

C.2 Alpaca Eval

Alpaca Eval 2.0 [Li et al., 2023, Dubois et al., 2023, 2024] is also a popular benchmark for evaluating LLMs. This is a single-turn dialogue LLM-based automatic evaluation using a stronger LLM, here GPT-4 Turbo, to assess the win rate of the trained LLM policy’s completion compared to either GPT-4 or the of the underlying SFT base model. Specifically, Alpaca Eval 2.0 has been validated against 20K human annotations and aims to reduce the length bias of Alpaca Eval, where using length-controlled (LC) Alpaca Eval shows a correlation with Chatbot Arena of 0.98, making it a popular benchmark with the highest correlation to Chatbot Arena [Dubois et al., 2024]. The Alpaca evaluation dataset consists of 841 high-quality instructions from different data sets. The library¹⁰ provided by Dubois et al. [2024] calculates the win-rate (percentage were the trained policy is preferred over the reference policy, first introduced in Alpaca Eval 1.0), and a length-controlled win-rate, where a linear model is fitted to de-bias for length of the prompt and instruction difficulty. We used a temperature of 0.7, sampling, and a maximum number of new tokens of 1024 to generate the answers. Furthermore, the library provides the standard error of the mean, which indicates the confidence of the win-rate and LC win-rate.

C.3 TL;DR Summarization Win-Rate

To evaluate how well the discovered objective functions generalize to the summarization task, we use the Alpaca Eval 2.0 library, similar to subsection C.2. Instead of using the Alpaca evaluation dataset, we create a custom dataset of 694 samples from the IMDb preference test dataset. Additionally, we change the prompt of the annotator LLM, to fit the "Summarization GPT-4 win rate prompt (C)" as described in Rafailov et al. [2023]. The (LC) win-rate is calculated against either the existing human-chosen test sample or against the summary generated by the SFT reference model. For a summary generation, we apply a temperature parameter of 0.7, sampling, and a maximum of 256 new tokens. Moreover, we stop the summarization after the "\n" token to avoid nonsensical generations. Furthermore, as we cannot calculate an instruction difficulty for the length-controlled win-rate, we omit this term from the linear model (This has only a small impact on the metric). In addition to the win-rates we also provide the standard error as a measure of confidence.

C.4 IMDb Rewards vs KL-Divergence

For the positive text generation, we do not require an LLM judge compared to MT-Bench, Alpaca Eval 2.0, and TL;DR evaluation, as we take a pre-trained sentiment classifier¹¹ as ground truth reward scorer. The LLMs apply sampling and a maximum of 60 new tokens for the positive text generation. The rewards and KL-divergence are averaged over 10 different generations from the trained LLMs.

⁹<https://github.com/lm-sys/FastChat>

¹⁰https://github.com/tatsu-lab/alpaca_eval

¹¹<https://huggingface.co/siebert/sentiment-roberta-large-english>

D Additional Results

D.1 Frontiers of Expected Reward vs KL Divergence

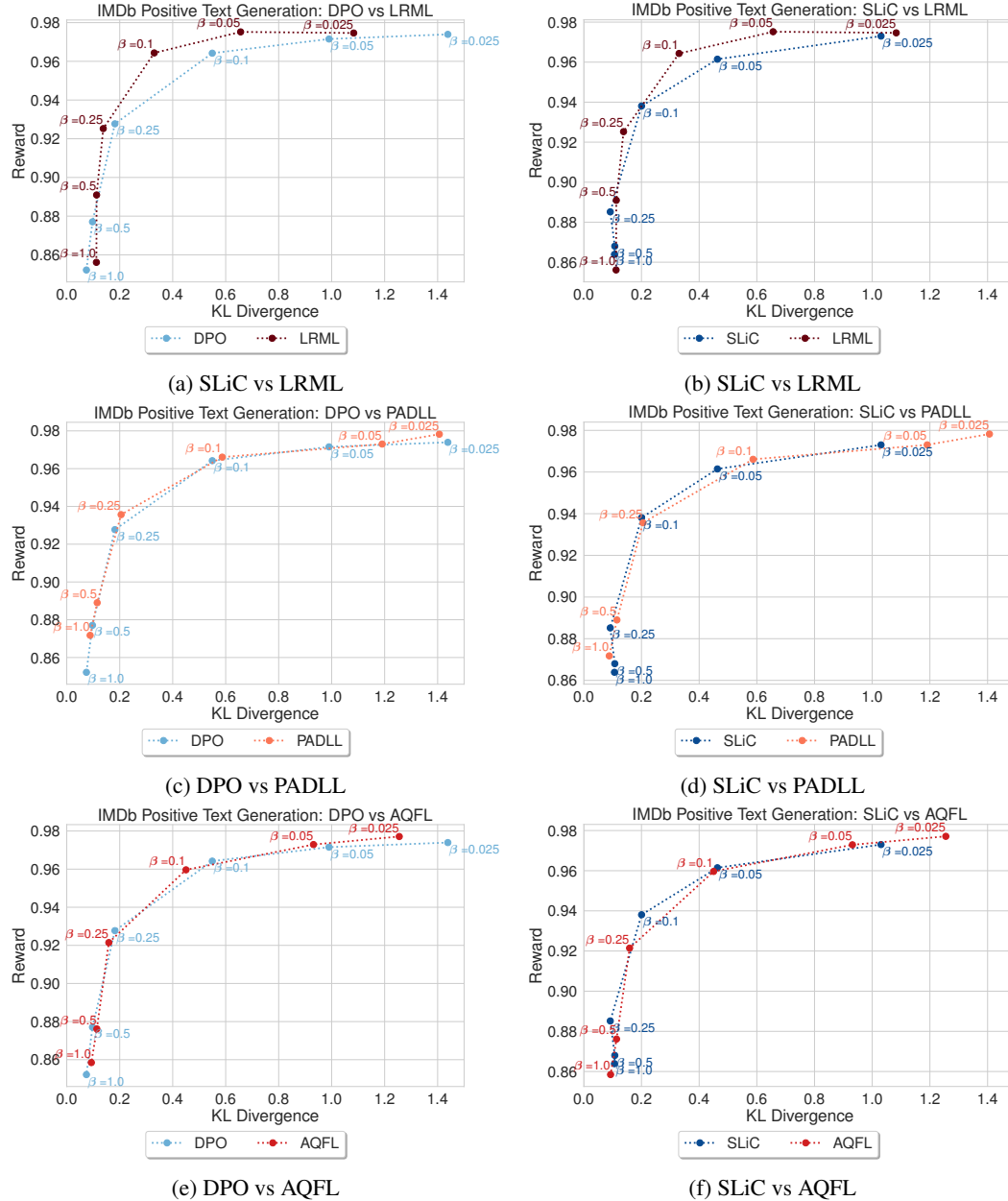


Figure 8: Frontiers of expected reward vs KL divergence after convergence for the baseline functions and all the discovered ones. The rewards and KL divergence values are averaged over 10 generations with different seeds. The sweep is done over $\beta \in \{0.025, 0.05, 0.1, 0.25, 0.5, 1, \}$. The optimal point is the top left corner, where perfect reward is achieved with minimal divergence from the reference model, to avoid reward hacking.

D.2 Loss Sweeps for Different Beta Parameters

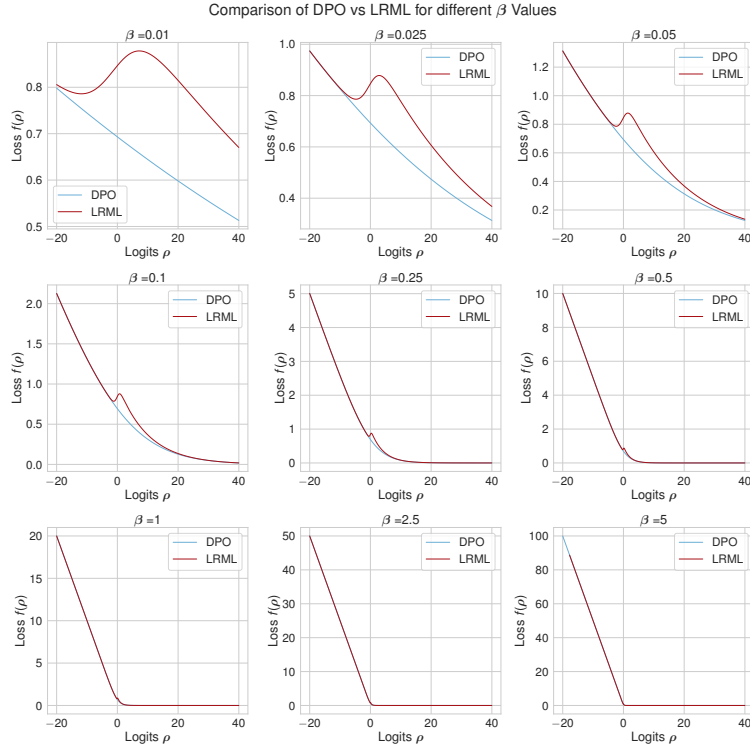


Figure 9: DPO and LRML objective function over $\beta \in \{0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5, 5\}$.

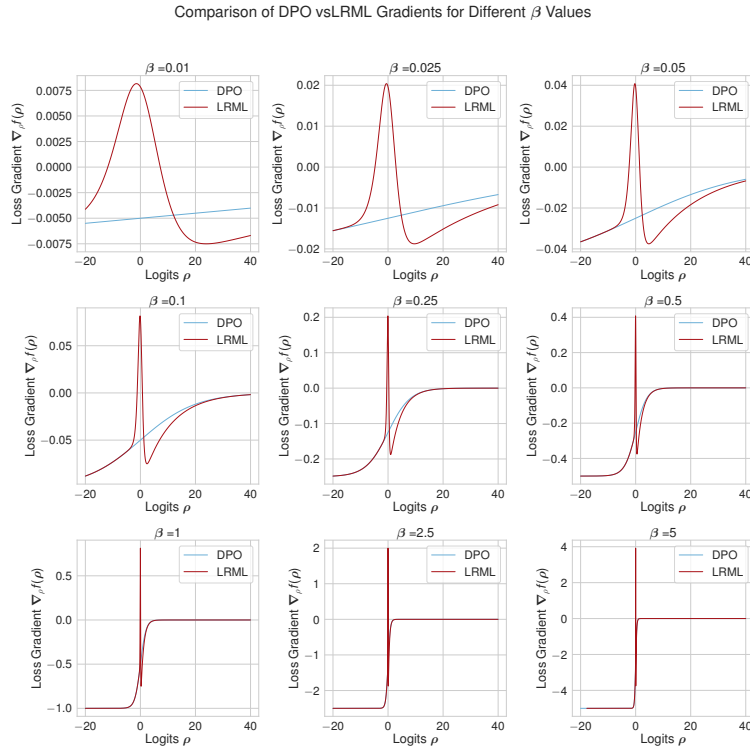


Figure 10: DPO and LRML gradient function over $\beta \in \{0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5, 5\}$.

D.3 Discovery Robustness with respect to LLM Hyperparameters

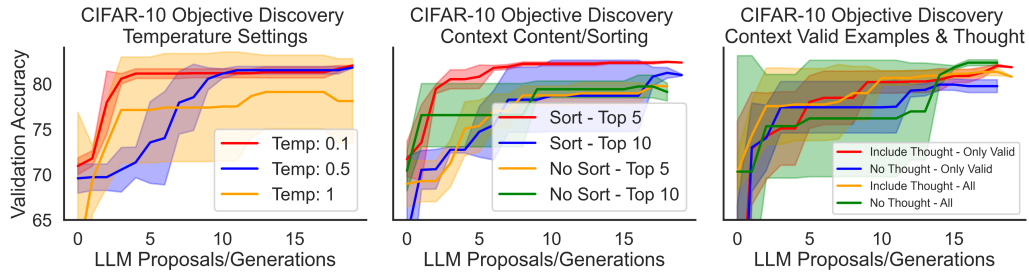


Figure 11: Robustness of the LLM-driven discovery process. **Left.** We compare different sampling temperatures $\{0.1, 0.5, 1.0\}$. **Middle.** The default configuration includes all objective proposals and evaluations in chronological order. Here we also explore using only the top- K performing objectives unsorted and sorted by their performance. **Right.** We also investigate whether using a "thought" as part of the context and whether to include non-valid code and error messages improves performance. The discovery process for CIFAR-10 objectives (5 epochs) is robust to these settings. The results are averaged across 3 independent runs.

D.4 LLM-Driven Discovery Analysis

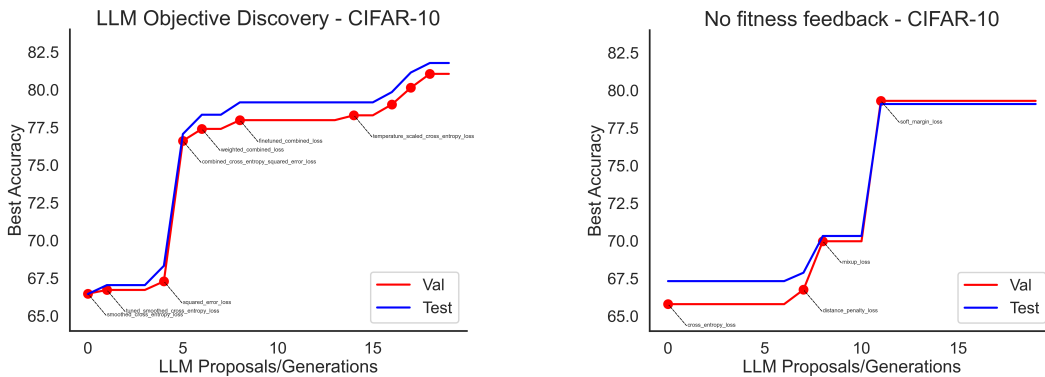


Figure 12: LLM-driven discovery for CIFAR-10 loss functions with (left) and without (right) providing fitness feedback. Note that without feedback, it performs worse but also is unable to *refine* its ideas, resulting in fewer improvements across generations.

D.5 Visual Language Models for Objective Discovery

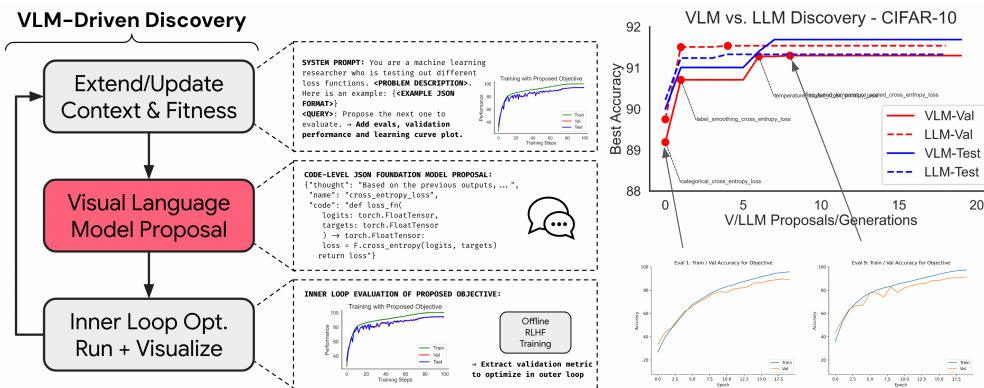


Figure 13: Objective Discovery with a Visual Language Model (VLM) for CIFAR-10 (20 epochs). We provide a plot of the training and validation accuracy across training as context components to the VLM (GPT-4-Turbo).

D.6 Additional Analysis of DiscoPOP

We performed further analysis to improve our understanding of the discovered loss function. We hypothesize that the local optimum of the DiscoPOP loss could catch noisy or incorrect data points. By inspecting the DiscoPOP loss values and log ratios of the training preference pairs in the IMDb dataset, we see that 1.35% of training points fall there (see Figure 14). Although we use the binary preference labels from the IMDb dataset¹² for training, the dataset also includes a positivity reward score for each completion, calculated by a separate reward model. When we analyze the data points between the local optima, we find that the positive and negative completions are significantly closer in absolute reward difference than those outside the local optimum (See Table 4). This implies that the preference labels on those points are likely more challenging to distinguish and help empirically validate our hypothesis.

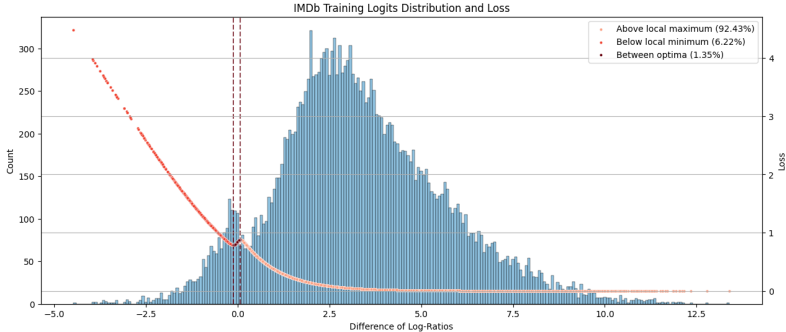


Figure 14: Distribution of β -scaled difference of log-ratios (left y-axis) and corresponding DiscoPOP loss value (right y-axis) of the training samples on the IMDb positive review generation task. The DiscoPOP function has a local minimum at $f_{lrml}(-2.3714) = 0.785929$ and a local maximum at $f_{lrml}(1.44012) = 0.87829$. The number of samples within the two local optima corresponds to 1.35% of the training set.

Description	Domain $\beta\rho$	Mean	95%-CI	p-value	% of training set
Between optima	$[-2.3714, 1.44012]$	0.981	(0.830, 1.131)	-	1.35
Full range	\mathbb{R}	3.861	(3.818, 3.904)	$< 0.01\%$	100
Outside optima	$\mathbb{R} \setminus [-2.3714, 1.44012]$	3.9	(3.857, 3.944)	$< 0.01\%$	98.65
Below local minimum	$(-\infty, -2.3714)$	4.086	(4.041, 4.131)	$< 0.01\%$	6.22
Above local maximum	$(1.44012, \infty)$	1.141	(1.076, 1.206)	4.29%	92.43

Table 4: The IMDb positive review preference dataset also provides “golden reward” scores for the chosen responses r_w and rejected responses r_l , calculated with a strong sentiment classifier ($\log p(\text{pos}) - \log p(\text{neg})$). We analysed the absolute difference in reward scores between the chosen and rejected responses $|r_w - r_l|$ across the different domains of the DiscoPOP function and report the statistics here. Training samples within the optima have a significantly lower mean absolute difference in rewards compared to the other regions in the DiscoPOP loss function. The samples stuck within the optima are the training triplets where the chosen and rejected responses are “closest” to each other and likely the most “noisy” labels.

E Discovered Objective Functions

To describe the discovered losses mathematically, we define three existing preference optimization losses here:

$$f_{dpo}(\beta\rho) = -\log(\sigma(\beta\rho)) = -\log\left(\frac{1}{1 + \exp(-\beta\rho)}\right) = \log(1 + \exp(-\beta\rho)) \quad (6)$$

$$f_{slic}(\beta\rho) = \text{ReLU}(1 - \beta\rho) \quad (7)$$

¹²https://huggingface.co/datasets/ZHZisZZ/imdb_preference

$$f_{exp}(\beta\rho) = \exp(-\beta\rho) \quad (8)$$

Moreover, we display the code of the discovered losses as it is output by the LLM. In addition, we provide a mathematical representation of each, which we have adapted to be consistent with β being the KL-Divergence regularization parameter. This is because the generated code for LRML, DBAQL, AQL, AQFL, and PFL did not uphold the β ought to be multiplied with the difference of log-ratios before any further calculations. If this was not upheld, it could lead to the loss function changing shapes based on the KL-regularization term, and therefore, models could not converge or potentially collapse. In future work, we should constrain the exploring LLM to uphold the β multiplication with the input before any other calculations are done with the difference of log-ratios ρ . As the meta-exploration was done with a set $\beta = 0.05$, and we wish to keep consistent with this scale of regularization, we have adapted the losses by dividing ρ values used in intermediate calculations with a scalar $\tau = 0.05$.

In the IMDb experiment in Section 5, we have thus used the corrected version of codes for the discovered losses based on the provided mathematical representation, as we were most interested in the effect of the KL-divergence compared to the model rewards.

E.1 DBAQL: Dynamic Blended Adaptive Quantile Loss

MT-Bench Score: 7.978

```
def dynamic_blended_adaptive_quantile_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    import torch.nn.functional as F
    # Constants for the loss function
    starting_quantile = 0.5
    quantile_adapt_rate = 0.01
    temperature = 0.9
    dynamic_blend_rate = 1.0

    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    logits_variability = logits.var()

    # Calculate an adaptive quantile based on a moving target
    moving_quantile = starting_quantile + quantile_adapt_rate * (torch
        .sigmoid(logits.mean()) - starting_quantile)

    # Calculate dynamic blending coefficient based on logits
    variability
    dynamic_blend_coef = torch.sigmoid(logits_variability) *
    dynamic_blend_rate

    # Prepare components of the blended loss
    logistic_loss = -F.logsigmoid(self.beta * logits / temperature)
    exp_loss = torch.exp(-self.beta * logits * temperature)

    # Blend the losses dynamically
    losses = dynamic_blend_coef * logistic_loss + (1 -
    dynamic_blend_coef) * exp_loss
    return losses
```

$$f_{dbaql}(\beta\rho) = \sigma(\text{Var}[\beta\rho/\tau]) \cdot f_{dpo}(\beta\rho/0.9) + (1 - \sigma(\text{Var}[\beta\rho/\tau])) \cdot f_{exp}(\beta\rho \cdot 0.9) \quad (9)$$

$$\tau = 0.05 \quad (10)$$

E.2 AQL: Adaptive Quantile Loss

MT-Bench Score: 7.953

```
def adaptive_quantile_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    percentile = 0.5 # Start with the median quantile
    moving_quantile_weight = 0.01 # Weight for updating the moving
    quantile
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios

    moving_quantile = percentile + moving_quantile_weight * (torch.
    sigmoid(logits.mean()) - percentile)

    quantile_weights = torch.sigmoid(-self.beta * (logits -
    moving_quantile))

    logistic_losses = -F.logsigmoid(self.beta * logits)
    hinge_losses = torch.relu(1 - self.beta * logits)

    # Blend the logistic and hinge losses based on the dynamic
    quantile weight
    losses = quantile_weights * logistic_losses + (1 -
    quantile_weights) * hinge_losses
    return losses
```

$$f_{aqi}(\beta\rho) = q \cdot f_{apo}(\beta\rho) + (1 - q) \cdot f_{stic}(\beta\rho) \quad (11)$$

$$q = \sigma(\tau m_2 - \beta\rho) \quad (12)$$

$$m_2 = 0.5 + 0.01 \cdot (\mathbb{E}[\sigma(\beta\rho/\tau)] - 0.5) \quad (13)$$

$$\tau = 0.05 \quad (14)$$

E.3 PADLL: Performance Adaptive Decay Logistic Loss

MT-Bench Score: 7.941

```
def performance_adaptive_decay_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    base_decay = 0.9
    mismatch_penalty = 0.5 # Penalty decay for mismatched choices

    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    mismatches = (logits < 0).float() # Identify mismatches

    adaptive_decay = base_decay * (1 - mismatches * mismatch_penalty)
    weighted_losses = adaptive_decay * -F.logsigmoid(self.beta *
    logits)
    return weighted_losses
```

$$f_{padll}(\beta\rho) = \delta_{\text{adpt}} \cdot f_{dpo}(\beta\rho) \quad (15)$$

$$= \delta_{\text{base}} \cdot (1 - \mathbb{1}_{[\rho < 0]} \cdot \tau) \cdot f_{dpo}(\beta\rho) \quad (16)$$

$$= \delta_{\text{base}} \cdot (1 - \mathbb{1}_{[\rho < 0]} \cdot \tau) \cdot \log(1 + \exp(-\beta\rho)) \quad (17)$$

$$= 0.9 \cdot (1 - \mathbb{1}_{[\rho < 0]} \cdot 0.5) \cdot \log(1 + \exp(-\beta\rho)) \quad (18)$$

This loss can also be rewritten as:

$$f_{padll}(\beta, \rho) = \begin{cases} \delta_{\text{pos}} \cdot f_{dpo}(\beta\rho), & \text{if } \rho \geq 0 \\ \delta_{\text{neg}} \cdot f_{dpo}(\beta\rho), & \text{if } \rho < 0 \end{cases}, \text{ where } \delta_{\text{pos}} > \delta_{\text{neg}} > 0 \quad (19)$$

E.4 AQFL: Adaptive Quantile Feedback Loss

MT-Bench Score: 7.931

```
def adaptive_quantile_feedback_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    import torch.nn.functional as F

    quantile_update_rate = 0.05
    distance_scale = 0.1

    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    logits_std = logits.std()

    adaptive_quantile = logits_std * torch.sigmoid(-logits).mean()
    adaptive_quantile += quantile_update_rate * (torch.sigmoid(logits).
    mean()) - adaptive_quantile)

    distance_from_quantile = (logits - adaptive_quantile).abs()
    blend_rate = torch.sigmoid(distance_scale * distance_from_quantile)

    logistic_losses = -F.logsigmoid(self.beta * logits)
    hinge_losses = torch.relu(1 - self.beta * logits)

    losses = blend_rate * logistic_losses + (1 - blend_rate) *
    hinge_losses
    return losses
```

$$f_{aqfl}(\beta\rho) = r \cdot f_{dpo}(\beta\rho) + (1 - r) \cdot f_{slie}(\beta\rho) \quad (20)$$

$$r = \sigma(0.1 * d) \quad (21)$$

$$d = |\beta\rho/\tau - m_2| \quad (22)$$

$$m_2 = m_1 + 0.05 \cdot (\sigma(\mathbb{E}[\beta\rho/\tau] - m_1)) \quad (23)$$

$$m_1 = \mathbb{E}[\sigma(-\beta\rho/\tau)] \cdot \sqrt{\text{Var}[\beta\rho/\tau]} \quad (24)$$

$$\tau = 0.05 \quad (25)$$

E.5 CELL: Combined Exponential + Logistic Loss

MT-Bench Score: 7.925

```

def combined_exp_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    exp_losses = torch.exp(-self.beta * logits)
    log_losses = -F.logsigmoid(self.beta * logits)
    # Combine the losses with a tunable mixing coefficient
    alpha = 0.5
    losses = alpha * exp_losses + (1 - alpha) * log_losses
    return losses

```

$$f_{cell}(\beta\rho) = 0.5 \cdot f_{dpo}(\beta\rho) + 0.5 \cdot f_{exp}(\beta\rho) \quad (26)$$

E.6 LRML: Log Ratio Modulated Loss

MT-Bench Score: 7.916

```

def log_ratio_modulated_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    # Modulate the mixing coefficient based on the log ratio
    magnitudes
    log_ratio_modulation = torch.sigmoid(logits)
    logistic_component = -F.logsigmoid(self.beta * logits)
    exp_component = torch.exp(-self.beta * logits)
    # Blend between logistic and exponential component based on log
    ratio modulation
    losses = logistic_component * (1 - log_ratio_modulation) +
    exp_component * log_ratio_modulation
    return losses

```

$$f_{lrml}(\beta\rho) = (1 - \sigma(\beta\rho/\tau)) \cdot f_{dpo}(\beta\rho) + \sigma(\beta\rho/\tau) \cdot f_{exp}(-\beta\rho) \quad (27)$$

$$\tau = 0.05 \quad (28)$$

E.7 PFL: Policy Focused Loss

MT-Bench Score: 7.900

```

def policy_focused_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:

```

```

focus_scale = 2.0 # Scale to emphasize or de-emphasize based on
the correctness of predictions

pi_logratios = policy_chosen_logps - policy_rejected_logps
ref_logratios = reference_chosen_logps - reference_rejected_logps
logits = pi_logratios - ref_logratios
is_correct = policy_chosen_logps > policy_rejected_logps

logistic_losses = -F.logsigmoid(logits)
hinge_losses = torch.relu(1 - logits)

focused_loss = torch.where(
    is_correct,
    logistic_losses / focus_scale, # De-emphasize correct
predictions
    hinge_losses * focus_scale # Emphasize incorrect predictions
)
return focused_loss

```

Interestingly, the PFL generated function code did not include any β values in the loss function. We have added it to the corrected code for the IMDb experiment, as well as to the mathematical expression below. We account for the consistency of the logit values by dividing with $\tau = 0.05$, which is the same value as β in the discovery process.

$$f_{pfl}(\beta\rho) = 1/2 \cdot f_{dpo}(\beta\rho/\tau) \cdot \mathbb{1}_{[\pi_w > \pi_r]} + 2 \cdot f_{slc}(\beta\rho/\tau) \cdot \mathbb{1}_{[\pi_w \leq \pi_r]} \quad (29)$$

$$\tau = 0.05 \quad (30)$$

F Extended Related Work

In the following, we provide an extended related work of the related work included in the main paper.

Evolution and Search with Large Language Models. LLMs provide a fast and automated way to create multiple candidate solutions for a problem stated in natural language [Song et al., 2024], which makes them powerful tools for driving population-based search procedures. Various recent works have applied this approach to coding problems [Romera-Paredes et al., 2024], neural architecture search [Chen et al., 2024a, Holt et al., 2024b,d], virtual robotic design settings [Lehman et al., 2023], reward functions [Ma et al., 2023, Yu et al., 2023], and algorithm heuristics [Liu et al., 2024]. Finally, recently LLMs have shown to be capable of acting as recombination operators for black-box optimization with Evolution Strategies [Lange et al., 2024] and for Quality-Diversity approaches [Lim et al., 2024]. Additionally, prior work has shown LLM multi-agent frameworks’ ability to excel at large code-generation tasks [Holt et al., 2024c], which we could envision enabling scaling up searching for larger candidate solutions for more complicated objectives or algorithms than the existing approaches that fit within the LLMs output context window.

Automated Discovery for Machine Learning. There are many other approaches to automating the discovery of generalizable machine learning algorithms. Some prior works explore the space of ML functions using genetic algorithms and a hand-crafted domain-specific language for reinforcement learning algorithms [Co-Reyes et al., 2021], curiosity algorithms [Alet et al., 2020], and optimizers [Chen et al., 2024b]. Other works instead parameterize a transferrable objective function using neural networks and optimize them with evolution strategies or meta-gradients. For example, Lu et al. [2022], Jackson et al. [2024b], Houthoofd et al. [2018], Alfano et al. [2024], Kirsch et al. [2019], Oh et al. [2020], Jackson et al. [2024a] discover policy optimization objectives, Metz et al. [2022] evolves neural network optimizers, and Lange et al. [2023b,a] evolve blackbox optimizers. Moreover, automatically discovering closed-form functions (i.e., symbolic regression), works exist that leverage RL [Petersen et al., 2020], gradient descent [Kacprzyk et al., 2024], RL with evolution strategies [Mundhenk et al., 2021], pre-training transformers [Biggio et al., 2021] and hybrid combinations of pre-training transformers, which are further refined with RL and evolution strategies [Holt et al., 2023b]. Furthermore, more general-purpose ML approaches exist to perform sequential modeling of

processes [Holt et al., 2022], which can be readily used in model-based RL approaches [Holt et al., 2023a, 2024a].

Preference Optimization Algorithms. While the reduction to supervised learning makes DPO and alternatives easier to use, other approaches have sought to simplify the RL step, including using variants of REINFORCE [Ahmadian et al., 2024, Gemma-Team et al., 2024] as well as more fine-grained feedback [Wu et al., 2024] through preferences over individual steps in the reasoning process [Uesato et al., 2022, Lightman et al., 2023] or reward redistribution [Chan et al., 2024]. Others use iterative offline training interleaved with sampling from the policy model and obtaining a preference ranking from themselves [Xu et al., 2023], another judge LLM [Guo et al., 2024], or an oracle [Swamy et al., 2024].

G Full Run Log

We provide a full run below, formatted for readability.

```
=====
thought
Since the logistic log loss currently has the highest fitness and is a
  direct optimization of the log likelihood ratio, let's try a
  variation that includes a margin in the log likelihood ratios,
  inspired by the concept of a margin in SVM hinge loss. Introducing
  a margin could create a buffer that leads to more robust learning
  , as seen in SVMs.
name
logistic_margin_loss
code
def logistic_margin_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    margin = 0.1
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits_with_margin = pi_logratios - ref_logratios - margin
    losses = -F.logsigmoid(self.beta * logits_with_margin)
    return losses

=====
Fitness: 7.709375.
Please generate the next one.
=====
thought
The logistic margin loss outperformed the hinge and ipo losses but did
  not surpass the original logistic log loss. Perhaps introducing
  an adaptive margin based on the confidence in the predictions
  could yield improvements. The margin could be set proportional to
  the absolute difference between the policy's and the reference's
  log probabilities. This could potentially penalize incorrect
  confident predictions more heavily.
name
adaptive_margin_logistic_loss
code
def adaptive_margin_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
```

```

    confidence_margin = torch.abs(policy_chosen_logps -
    policy_rejected_logps) - torch.abs(reference_chosen_logps -
    reference_rejected_logps)
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits_with_margin = pi_logratios - ref_logratios -
    confidence_margin
    losses = -F.logsigmoid(self.beta * logits_with_margin)
    return losses

=====
Fitness: 7.51875.
Please generate the next one.
=====
thought
The adaptive margin did not yield an improvement. It might have
introduced too much variability or penalization. Let's go back to
a fixed structure but combine elements from both logistic loss and
hinge loss, with the idea of encouraging a larger margin when the
decision is correct, while still having a soft, probabilistic
interpretation. This combined loss could retain the benefits of
both worlds.
name
combined_logistic_hinge_loss
code
def combined_logistic_hinge_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    logistic_losses = -F.logsigmoid(self.beta * logits)
    hinge_losses = torch.relu(1 - self.beta * logits)
    combined_losses = logistic_losses + hinge_losses
    return combined_losses

=====
Fitness: 7.7375.
Please generate the next one.
=====
thought
The combined logistic and hinge loss slightly outperformed the
baseline logistic log loss but still did not achieve a significant
improvement. Considering the nature of RLHF, where we prefer the
model to be correct but not overly confident, we can explore a
scaled version of the combined loss where the logistic part is
emphasized more heavily, as it intrinsically accounts for
confidence due to its probabilistic nature.
name
scaled_combined_logistic_hinge_loss
code
def scaled_combined_logistic_hinge_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    logistic_scale = 2.0 # Emphasize logistic loss more
    hinge_scale = 0.5 # De-emphasize hinge loss

```



```

pi_logratios = policy_chosen_logps - policy_rejected_logps
ref_logratios = reference_chosen_logps - reference_rejected_logps
logits = pi_logratios - ref_logratios

logistic_losses = logistic_scale * (-F.logsigmoid(self.beta *
logits))
hinge_losses = hinge_scale * (torch.relu(1 - self.beta * logits))

combined_losses = logistic_losses + hinge_losses
return combined_losses

=====
Fitness: 7.85625.
Please generate the next one.
=====
thought
The scaled combined loss has shown an improvement, which indicates
that balancing the contribution between the logistic and hinge
components is beneficial. To further expand on this idea, we could
try a form of annealing where early in training we use more of
the logistic component, encouraging better probability estimation,
and later on a switch gradually to the hinge component,
emphasizing decisions with a margin. We'll simulate this by using
a weighting factor that changes linearly through the logits,
giving lower logits (earlier, less confident predictions) a higher
weight for the logistic loss.
name
annealed_combined_logistic_hinge_loss
code
def annealed_combined_logistic_hinge_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios

    logistic_scale = torch.sigmoid(-logits) # Higher weight for lower
    logits
    hinge_scale = torch.sigmoid(logits) # Higher weight for
    higher logits

    logistic_losses = logistic_scale * (-F.logsigmoid(self.beta *
logits))
    hinge_losses = hinge_scale * (torch.relu(1 - self.beta * logits))

    combined_losses = logistic_losses + hinge_losses
    return combined_losses

=====
Fitness: 7.66875.
Please generate the next one.
=====
thought
The annealed combined loss did not yield the expected improvement and
performed worse than the scaled version. It's possible the
transition from logistic to hinge was too aggressive. For the next
iteration, let's return to a simpler concept and try a variation
of the logistic loss with a squared term, similar to squared hinge
loss, which might penalize incorrect predictions more smoothly
than the linear term used in the standard logistic loss.
name

```

```

squared_logistic_loss
code
def squared_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    logistic_losses = -F.logsigmoid(self.beta * logits)
    squared_losses = logistic_losses ** 2
    return squared_losses

=====
Fitness: 7.60062893081761.
Please generate the next one.
=====
thought
The squared logistic loss did not perform as well as hoped, possibly
due to excessive penalization of difficult samples. This time, let
's integrate a weighting mechanism that focuses on hard examples
while using the logistic component. Inspired by the focal loss
used in object detection, which gives more weight to harder,
misclassified examples, we can apply a similar mechanism to
emphasize learning from examples where the policy significantly
differs from the reference.

name
focal_logistic_loss
code
def focal_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    gamma = 2.0 # Focusing parameter for modulating the loss
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    sigmoids = F.sigmoid(logits)
    focal_weights = (1 - sigmoids) ** gamma # Focus more on harder
examples
    logistic_losses = -focal_weights * F.logsigmoid(self.beta * logits
)
    return logistic_losses

=====
Fitness: 7.840625.
Please generate the next one.
=====
thought
The focal logistic loss seems to have a moderate effect, indicating
that prioritizing hard examples has some merit. To build on this,
an alternative could be to apply temperature scaling to the logits
before computing the logistic loss. Temperature scaling is often
used in model calibration and can soften the probability
distribution, which might work well with RLHF where overconfidence
in predictions is undesirable.

name
temperature_scaled_logistic_loss
code

```

```

def temperature_scaled_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    temperature = 2.0 # Temperature > 1.0 softens the logits
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    tempered_logits = (pi_logratios - ref_logratios) / temperature
    logistic_losses = -F.logsigmoid(self.beta * tempered_logits)
    return logistic_losses

=====
Fitness: 7.86875.
Please generate the next one.
=====
thought
The temperature scaling seemed to have a positive impact, possibly due
to better-calibrated probability estimates. To further explore
this direction, we can try using label smoothing, which encourages
the model not to be too confident about its predictions by
preventing it from assigning full probability to a single class.
It's a technique commonly used in classification tasks and might
be beneficial for RLHF.
name
label_smoothed_logistic_loss
code
def label_smoothed_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    label_smoothing = 0.1 # Epsilon for label smoothing
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    smooth_positive = 1.0 - label_smoothing
    smooth_negative = label_smoothing / 2.0
    targets = torch.ones_like(logits) * smooth_positive
    losses = F.binary_cross_entropy_with_logits(
        self.beta * logits, targets, reduction='none'
    ) + smooth_negative * F.binary_cross_entropy_with_logits(
        -self.beta * logits, torch.zeros_like(logits), reduction='none'
    )
    return losses.mean()

=====
Code not valid. Error:
Expected loss shape to be per input (e.g. (10,)), got torch.Size([])
Please generate the next one.
=====
thought
Since the label smoothing implementation had an issue with the loss
shape being incorrect, it seems the loss reduction was erroneously
computed over all inputs instead of keeping the per-input format.
Let's adjust the implementation to make sure the loss retains the
correct shape, by performing the mean operation separately for
the positive and negative parts and then combining them, weighted
appropriately.
name

```

```

corrected_label_smoothed_logistic_loss
code
def corrected_label_smoothed_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    label_smoothing = 0.1 # Epsilon for label smoothing
    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    smooth_positive = 1.0 - label_smoothing
    smooth_negative = label_smoothing / 2.0
    positive_targets = torch.ones_like(logits) * smooth_positive
    negative_targets = torch.zeros_like(logits) * smooth_negative
    positive_losses = F.binary_cross_entropy_with_logits(
        self.beta * logits, positive_targets, reduction='none'
    )
    negative_losses = F.binary_cross_entropy_with_logits(
        self.beta * logits, negative_targets, reduction='none'
    )
    return (positive_losses + negative_losses) / 2

=====
Fitness: 6.425.
Please generate the next one.
=====
thought
The corrected label smoothing implementation didn't work as intended
and significantly decreased the fitness. It seems that label
smoothing may not align well with the objective of RLHF. In light
of this, let's explore a different direction by introducing a
decaying weight on older samples. The idea is to give higher
importance to the more recent decisions made by the policy, under
the assumption that they may be more aligned with the current
state of the policy.

name
decaying_weights_logistic_loss
code
def decaying_weights_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    decay_rate = 0.9 # Weight decay for older samples
    batch_size = policy_chosen_logps.size(0)
    decay_weights = decay_rate ** torch.arange(batch_size - 1, -1, -1)
    decay_weights = decay_weights.to(policy_chosen_logps.device)

    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    losses = decay_weights * -F.logsigmoid(self.beta * logits)
    return losses / decay_weights.sum() # Normalizing by sum of
weights

=====
Fitness: 7.871875.
Please generate the next one.
=====
thought

```

While the decaying weights logistic loss provided a slight increase in fitness, it suggests that emphasizing more recent samples can be beneficial, but the approach might need some refinement. We could potentially improve this by making the decay adaptive based on the performance of each choice. The idea would be to give less weight to choices that are heavily mismatched with the reference, under the hypothesis that these could be outliers or errors.

```

name
performance_adaptive_decay_logistic_loss
code
def performance_adaptive_decay_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    base_decay = 0.9
    mismatch_penalty = 0.5 # Penalty decay for mismatched choices

    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    mismatches = (logits < 0).float() # Identify mismatches

    adaptive_decay = base_decay * (1 - mismatches * mismatch_penalty)
    weighted_losses = adaptive_decay * -F.logsigmoid(self.beta *
logits)
    return weighted_losses

```

=====

Fitness: 7.940625.

Please generate the next one.

=====

thought

The performance-adaptive decay approach provided a slight improvement in fitness, suggesting that dynamically adjusting the loss based on the correctness of predictions is a promising direction. To take this further, let's create a hybrid approach which combines the performance-adaptive decay with a form of margin-based loss. This will aim to reduce the weights of not only the incorrect predictions but also those that are correct yet lack confidence, thereby promoting a more decisive policy.

name

hybrid_performance_margin_decay_logistic_loss

code

```

def hybrid_performance_margin_decay_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    base_decay = 0.9
    margin = 0.2
    mismatch_penalty = 0.5 # Penalty decay for mismatched choices

    pi_logratios = policy_chosen_logps - policy_rejected_logps
    ref_logratios = reference_chosen_logps - reference_rejected_logps
    logits = pi_logratios - ref_logratios
    margin_logits = logits - margin
    mismatches = (margin_logits < 0).float() # Identify mismatches
with margin

    adaptive_decay = base_decay * (1 - mismatches * mismatch_penalty)

```

```

    weighted_losses = adaptive_decay * -F.logsigmoid(self.beta *
margin_logits)
    return weighted_losses

=====
Fitness: 7.6125.
Please generate the next one.
=====
thought
The hybrid approach with the performance margin decay did not achieve
the desired effectiveness. Introducing a margin may have been too
punitive on correct predictions that are close to the boundary. To
maintain the balance, let's explore the idea of using a triplet-
style loss, which is popular in embedding learning. By treating
the chosen policy logs as the anchor and the reference chosen and
rejected logs as positive and negative examples, respectively, we
can encourage the chosen policy decisions to be closer to the
reference chosen decisions and farther away from the reference
rejected decisions.

name
triplet_style_logistic_loss
code
def triplet_style_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    alpha = 0.1 # Margin for the triplet loss
    positive_distance = -F.logsigmoid(self.beta * (policy_chosen_logps
- reference_chosen_logps))
    negative_distance = -F.logsigmoid(self.beta * (policy_chosen_logps
- reference_rejected_logps))
    triplet_loss = F.relu(positive_distance - negative_distance +
alpha)
    return triplet_loss

=====
Code not valid. Error:
isnan(): argument 'input' (position 1) must be Tensor, not NoneType
Please generate the next one.
=====
thought
It seems there was an error in the triplet-style loss function, which
suggests that the computation may have been incorrect or
incomplete. To correct this, we need to ensure that all
intermediate steps yield non-empty tensors. Triplet loss can still
be a useful approach as it inherently contains the concept of a
margin while allowing for a dynamic balance between the chosen and
rejected log probabilities. Let's revise the code with added
checks to prevent any NoneType issues and ensure that the loss is
computed correctly.

name
revised_triplet_style_logistic_loss
code
def revised_triplet_style_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    alpha = 0.1 # Margin for the triplet loss
    # Compute distances

```

```

    positive_distance = policy_chosen_logps - reference_chosen_logps
    negative_distance = policy_chosen_logps - reference_rejected_logps
    # Calculate the triplet loss
    triplet_loss = F.relu(positive_distance - negative_distance +
alpha)
    # Convert triplet loss into a logistic-style loss
    logistic_triplet_loss = -F.logsigmoid(self.beta * triplet_loss)

    return logistic_triplet_loss.mean() # Ensure the loss is averaged

```

=====

Code not valid. Error:
Expected loss shape to be per input (e.g. (10,)), got torch.Size([])
Please generate the next one.

=====

thought

The revised triplet-style logistic loss still encountered an issue, likely due to improper application of the operations and reduction at the end, which should be on a per-example basis. Triplet loss typically works with distances embedded in a space, and here we are dealing with log probabilities, so the direct application may not be correct. For the next attempt, let's simplify: we'll reinforce the relationship between the chosen and rejected log probabilities directly by using their difference, promoting a wider margin while still within the logistic loss framework.

name

reinforced_margin_logistic_loss

code

```

def reinforced_margin_logistic_loss(
    self,
    policy_chosen_logps: torch.FloatTensor,
    policy_rejected_logps: torch.FloatTensor,
    reference_chosen_logps: torch.FloatTensor,
    reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
    margin = 0.1 # Margin introduced for reinforcing the difference
    # Calculate log probability differences
    chosen_difference = policy_chosen_logps - reference_chosen_logps
    rejected_difference = policy_rejected_logps -
reference_rejected_logps
    # Apply the margin to differences
    reinforced_chosen = F.relu(chosen_difference + margin)
    reinforced_rejected = F.relu(rejected_difference - margin)
    # Compute logistic losses with reinforced margins
    losses = -F.logsigmoid(self.beta * reinforced_chosen) - F.
logsigmoid(-self.beta * reinforced_rejected)
    return losses.mean(0) # Maintain loss shape as per input

```

=====

Code not valid. Error:
Expected loss shape to be per input (e.g. (10,)), got torch.Size([])
Please generate the next one.

=====

thought

There was an error in the implementation of the reinforced margin logistic loss, likely because the mean operation was again placed incorrectly. The loss should be returned without any aggregation to maintain the per-input structure. Keeping this in mind, we need to correctly apply a margin to reinforce the separation between policy decisions and reference decisions.

name

corrected_reinforced_margin_logistic_loss

code

```

def corrected_reinforced_margin_logistic_loss(
    self,

```

```

policy_chosen_logps: torch.FloatTensor,
policy_rejected_logps: torch.FloatTensor,
reference_chosen_logps: torch.FloatTensor,
reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
margin = 0.1 # Margin for reinforcing the separation
# Calculate log probability differences
chosen_difference = policy_chosen_logps - reference_chosen_logps
rejected_difference = policy_rejected_logps -
reference_rejected_logps
# Reinforce chosen logits by adding a margin
reinforced_chosen = chosen_difference + margin
# Reinforce rejected logits by subtracting a margin
reinforced_rejected = rejected_difference - margin
# Compute logistic losses for reinforced logits
chosen_losses = -F.logsigmoid(self.beta * reinforced_chosen)
rejected_losses = -F.logsigmoid(-self.beta * reinforced_rejected)
# Combine losses without applying mean
return chosen_losses + rejected_losses

=====
Fitness: 7.525.
Please generate the next one.
=====
thought
The approach of applying a static margin did not yield the expected
improvements. It's worth considering a different perspective;
instead of focusing on introducing margins or applying decays, let
's attempt to directly optimize the policy's certainty.
Intuitively, the policy should be more confident when choosing
actions similar to the reference and less confident otherwise. A
potential approach would be to scale the logistic loss by the
absolute difference between the policy's choice and the reference
choice, thus directly tying the loss to the policy's certainty in
its decision.
name
certainty_scaled_logistic_loss
code
def certainty_scaled_logistic_loss(
self,
policy_chosen_logps: torch.FloatTensor,
policy_rejected_logps: torch.FloatTensor,
reference_chosen_logps: torch.FloatTensor,
reference_rejected_logps: torch.FloatTensor,
) -> torch.FloatTensor:
# Compute the absolute certainty differences
chosen_certainty_diff = torch.abs(policy_chosen_logps -
reference_chosen_logps)
rejected_certainty_diff = torch.abs(policy_rejected_logps -
reference_rejected_logps)
# Calculate the mean certainty difference
mean_certainty_diff = (chosen_certainty_diff +
rejected_certainty_diff) / 2
# Compute the logistic loss
losses = -F.logsigmoid(self.beta * (policy_chosen_logps -
policy_rejected_logps))
# Scale the loss by the certainty difference
certainty_scaled_losses = losses * mean_certainty_diff
return certainty_scaled_losses

=====
Fitness: 7.33125.
Please generate the next one.

```


NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: Please see Sections 2-8.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: We discuss limitations in Section 8 under the heading **Limitations & Future Work**

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory Assumptions and Proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: We do not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental Result Reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide details on all our experiments in the main paper and appendix, as well as providing code for reproducing them at <https://github.com/samholt/DiscoPOP>.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide code for reproducing results at <https://github.com/samholt/DiscoPOP>.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental Setting/Details

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Details are given in Sections 4, 5, and 6.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment Statistical Significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We provide details on standard errors in the results tables.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments Compute Resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: Details are given in Appendix B.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code Of Ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines?>

Answer: [Yes]

Justification: We reviewed the NeurIPS Code of Ethics and ensured the research conforms to it.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader Impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: We discuss this in Section 8 under the heading **Broader Impact and Ethical Considerations**.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The paper poses no such risk.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All creators are properly credited.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.

- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: The paper does not release new assets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.