LLMs are Highly-Constrained Biophysical Sequence Optimizers

Angelica Chen[1,2] **Samuel D. Stanton** [2] **Robert G. Alberstein**[2] **Andrew M. Watkins**[2]
Richard Bonneau[2] **Vladimir Gligorijević**[2] **Kyunghyun Cho**[1,2,3] **Nathan C. Frey**[2]
{chena78,stantos5,freyn6}@gene.com
[1]New York University [2]Prescient Design, Genentech [3]CIFAR LMB

Abstract

Large language models (LLMs) have recently shown significant potential in various biological tasks such as protein engineering and molecule design. These tasks typically involve black-box discrete sequence optimization, where the challenge lies in generating sequences that are not only biologically feasible but also adhere to hard fine-grained constraints. However, LLMs often struggle with such constraints, especially in biological contexts where verifying candidate solutions is costly and time-consuming. In this study, we explore the possibility of employing LLMs as *highly-constrained bilevel optimizers* through a methodology we refer to as Language Model Optimization with Margin Expectation (LLOME). This approach combines both offline and online optimization, utilizing limited oracle evaluations to iteratively enhance the sequences generated by the LLM. We additionally propose a novel training objective – Margin-Aligned Expectation (MargE) – that trains the LLM to smoothly interpolate between the reward and reference distributions. Lastly, we introduce a synthetic test suite that bears strong geometric similarity to real biophysical problems and enables rapid evaluation of LLM optimizers without time-consuming lab validation. Our findings reveal that, in comparison to genetic algorithm baselines, LLMs achieve significantly lower regret solutions while requiring fewer test function evaluations. However, we also observe that LLMs exhibit moderate miscalibration, are susceptible to generator collapse, and have difficulty finding the optimal solution when no explicit ground truth rewards are available.

1 Introduction

Large language models (LLMs) have recently shown significant promise on various biophysical optimization tasks, such as protein engineering and molecule design. These tasks are often formulated as black-box discrete sequence optimization problems, wherein a solver must attempt to output a discrete sequence $\mathbf{x} \in \mathcal{X}$ that is *feasible* (*i.e.*, a biologically plausible sequence) and that fulfills a number of strict constraints, such as containing specific motifs. Yet despite their many successes, LLMs often struggle to generate outputs that fulfill hard fine-grained constraints (Garbacea & Mei, 2022). For instance, recent work indicates that many state-of-the-art LLMs cannot effectively generate text with a fixed number of words (Yuan et al., 2024), a particular prefix (Sun et al., 2023), or specific keywords (Chen et al., 2024c). This issue is even more pronounced in biological optimization problems, where constraints such as expressivity or low hydrophobicity are typically more complex and costly to check. Verifying that solution candidates meet these constraints often requires expensive and low throughput laboratory experiments. This lack of data presents a formidable challenge for LLMs, which often struggle to generate appropriately constrained sequences even in the presence of large training corpora.

AI for New Drug Modalities at NeurIPS 2024.

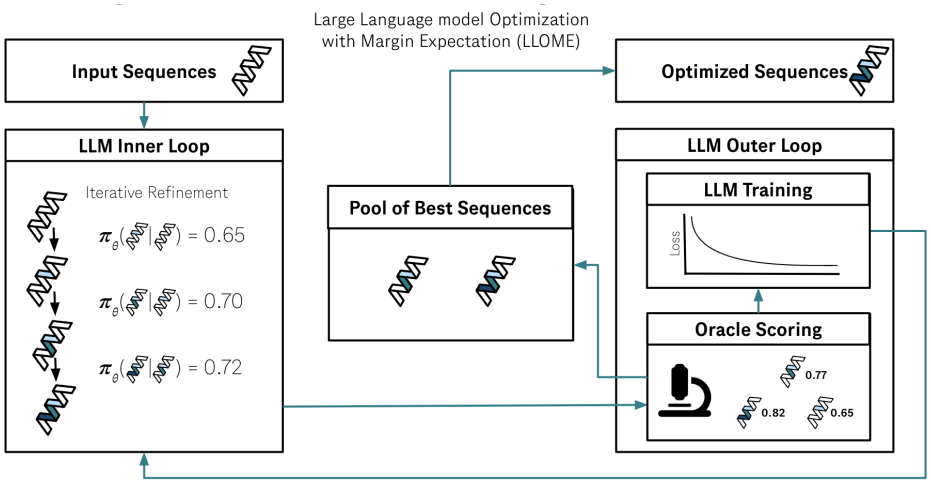


Figure 1: Large language model optimization with Margin Expectation (LLOME).

Although some recent work has focused on adapting LLMs to solve black-box optimization problems (Ma et al., 2024; Romera-Paredes et al., 2023; AhmadiTeshnizi et al., 2024; Chen et al., 2024a), few include (1) the evaluation of hard constraints, (2) the mixing of offline and online optimization loops, and (3) the use of oracle evaluations. Concerning (1), many commonly studied optimization problems, such as the Rosenbrock function (Nie et al., 2023), the Ackley function (Zhang et al., 2023b), and prompt optimization (Yang et al., 2024), require only that the optimizer find a solution with a high *fitness value*, and do not entail strict constraints. When stricter constraints are added, LLMs struggle to generalize without additional aids, such as auxiliary solvers or discriminator models that guide generation (Sun et al., 2023; Mittal et al., 2024; Dathathri et al., 2020; AhmadiTeshnizi et al., 2024). Concerning (2) and (3), laboratory validation often provides a limited number of oracle labels in realistic lab settings. The solver must then be capable of generating, ranking, and filtering its candidates down to a select few that are subsequently sent for oracle evaluation. Importantly, oracle labels are not available during this candidate generation stage. In contrast, many works either use *only* oracle labels, such as for mathematical optimization problems like linear regression where a solution is easy to verify (Nie et al., 2023; Yang et al., 2024), or simulation-based metrics that bear only a superficial resemblance to the complexity of real-world molecular design tasks (Ma et al., 2024). We instead explore LLM capabilities in a more realistic hybrid offline/online setting. During an outer offline optimization loop, we train our optimizer using oracle labels provided by either a seed dataset or the previous round of optimization. In the subsequent online inner optimization loop, our optimizer generates and iteratively refines its candidates without immediate access to the oracle. The optimizer's top candidates are then scored by the oracle, forming a small labelled training dataset for the next iteration of the outer loop.

In summary, our contributions are as follows:

1. **Synthetic Test Suite:** Following a long-standing tradition of using synthetic test functions in the optimization literature, we design a suite of closed-form test functions with hard fine-grained constraints that draws upon the characteristics of real biophysical sequence optimization problems. Our test functions have well-characterized solutions, are non-trivial in difficulty, and reflect the non-additive, epistatic nature of biomolecular interactions.
2. **Exploring LLMs as Constrained Bilevel Optimizers:** We analyze LLMs' abilities to solve highly-constrained biophysical sequence optimization problems with sparse online feedback via **LLOME (Large Language model Optimization with Margin Expectation)**, a method for embedding an LLM in a bilevel optimization loop. In the *outer loop*, the model is trained on offline data provided by an oracle (*e.g.*, laboratory validation or the evaluation of the synthetic test function). In the *inner loop*, the model searches for novel and improved sequences without the use of oracle labels. We demonstrate that LLMs often outperform evolutionary baselines, finding lower-regret solutions given a fixed budget of function evaluations. We additionally explore a number of other fundamental questions about LLMs' optimization capabilities, including whether

LLMs can effectively select the best generations, how calibrated LLM predictions are, whether LLMs can iteratively extrapolate at inference-time without access to oracle labels, and when explicit reward values are required during training.

3. **Novel LLM Training Objective (MargE)**: To further explore whether LLMs become better bi-level optimizers when given explicit ground truth rewards, we also propose MargE (Margin-Aligned Expectation), a novel LLM training objective that smoothly interpolates between the target and reference distributions. This objective makes explicit use of the reward margin between the input and the target, which is lacking from common LLM offline training objectives such as supervised finetuning (SFT) and direct preference optimization (DPO). We show that MargE-trained LLMs find the optimal solution more efficiently than DPO- or SFT-trained LLMs.

2 Background

In this work, we consider pre-trained autoregressive large language models $\pi_\theta(x)$ parameterized by θ. π_θ defines a probability distribution over discrete tokens $x \in \mathcal{V}$ for vocabulary $\mathcal{V}$. We can also define the likelihood of sequences $\mathbf{x} \in \mathcal{V}^*$ as $\pi_\theta(\mathbf{x}) = \prod_{t=1}^{|\mathbf{x}|} \pi_\theta(x_t|x_{<t})$, where $\mathcal{V}^*$ is the set of all concatenations of tokens in $\mathcal{V}$.

Supervised Finetuning (SFT) After pre-training, LLMs are typically finetuned on some task-specific dataset $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})\}$ consisting of pairs of input $\mathbf{x}$ and target $\mathbf{y}$ sequences. During SFT, π_θ is trained to minimize the negative conditional log likelihood of examples from $\mathcal{D}$:

$$\theta_{\text{SFT}} = \arg\min_\theta -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\mathcal{D}} \log \pi_\theta(\mathbf{y}|\mathbf{x}) \tag{1}$$

Preference Learning In some settings, LLMs are further trained to align their output distributions to a *reward distribution*, typically encoded with a reward model $r(\mathbf{x}) : \mathcal{V}^* \to \mathbb{R}$. This is frequently accomplished with reinforcement learning, where the LLM is trained via a policy gradient method to maximize the expected rewards $\mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(\cdot|x)} r(x, y)$. The reward model is trained from a dataset of human preferences consisting of triples (x, y_w, y_l), where x is a prompt obtained from some offline dataset $\mathcal{X}$ and y_w and y_l are sampled from the current policy. The initial model is referred to as the *reference policy* π_{Ref} and y_w, y_l are assigned such that y_w is more preferred by human raters than y_l. More recently, practitioners have added a KL divergence constraint to this objective to prevent the LLM from quickly over-optimizing, yielding an objective known as *reinforcement learning from human feedback* (RLHF; Ziegler et al., 2019; Stiennon et al., 2020):

$$\theta_{\text{RLHF}} = \arg\max_\theta \mathop{\mathbb{E}}_{\substack{x\sim\mathcal{X} \\ y\sim\pi_\theta(\cdot|x)}} r(x, y) - \beta\mathbb{KL}(\pi_\theta \| \pi_{\text{Ref}}) \tag{2}$$

RLHF is commonly trained using Proximal Policy Optimization (PPO), which involves considerable engineering complexity due to the need to train and coordinate four models (π_θ, π_{Ref}, a reward model, and a critic model). Furthermore, RLHF-PPO is particularly sensitive to hyperparameter values and prone to training instabilities (Zheng et al., 2023b). To address some of these issues, an offline version of RLHF known as Direct Preference Optimization (DPO; Rafailov et al., 2023) was proposed. DPO skips reward modeling and directly trains on the preference triples with a contrastive objective:

$$\theta_{\text{DPO}} = \arg\min_\theta \mathop{\mathbb{E}}_{x,y_w,y_l\sim\mathcal{D}} -\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{Ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{Ref}}(y_l|x)}\right) \tag{3}$$

DPO often produces models with similar generative quality as RLHF-PPO (Rafailov et al., 2023), but involves notable tradeoffs such as faster over-optimization (Rafailov et al., 2024a) and a distribution mismatch between the training dataset and policy outputs (Chen et al., 2024b; Tang et al., 2024). Nevertheless, DPO has become one of the most prevalent algorithms for offline alignment of LLMs.

We provide further background on past work related to LLMs for optimization in Sec. 5.1.

3 Evaluation

Although considerable effort has been focused on ML efforts for biological discovery in recent years, the rate of benchmark development has not kept pace (Tripp et al., 2021; Stanton et al., 2022). While

laboratory validation is the gold standard in evaluation, laboratory feedback cycles are slow and involve prohibitive costs of equipment and expert experimentalists. On the other hand, ML algorithm development requires rapid iteration and feedback, usually at a pace that is infeasible with *in vitro* validation. To close this gap, computational researchers often turn to simulation-based feedback (Cieplinski et al., 2023). However, this approach is also subject to steep tradeoffs between fidelity and latency (Kellogg et al., 2011; Barlow et al., 2018; Hummer et al., 2023).

In this work, we instead propose a class of synthetic test functions that bear strong geometric similarities to real-world biophysical sequence optimization problems. The practice of evaluating optimizers on synthetic test functions is a long-standing one (Molga & Smutnicki, 2005), based upon the principle that the test function need only correlate *geometrically* with the real sequence optimization problem. This approach has the advantage of being easy to verify but difficult to solve while still reflecting key characteristics of the fitness landscapes of real biophysical optimization tasks such as antibody affinity maturation.

To this end, we propose *Ehrlich functions*, a closed-form family of test functions for sequence optimization benchmarking. Given token vocabulary $\mathcal{V}$ and the set of sequences $\mathcal{V}^L$ formed of concatenations of L tokens in $\mathcal{V}$, we first define $\mathcal{F} \subset \mathcal{V}^L$, the set of *feasible sequences*. $\mathcal{F}$ is synthetically defined as the support of a discrete Markov process (DMP), more details of which are given in Appendix A.1. We also define a set of *c spaced motifs* that represent biophysical constraints in specific regions of a sequence. These motifs are expressed as pairs of vectors $\{(\mathbf{m}^{(1)}, \mathbf{s}^{(1)}), \cdots, (\mathbf{m}^{(c)}, \mathbf{s}^{(c)})\}$ where $\mathbf{m}^{(i)} \in \mathcal{V}^k$, $\mathbf{s}^{(i)} \in \mathbb{Z}_+^k$, and $k \leq L//c$. An Ehrlich function f then describes the degree to which a sequence $\mathbf{x} \in \mathcal{V}^L$ is feasible and possesses all motifs. For $q \in [1, k]$ bits of precision, f is expressed as:

$$f(x) = \begin{cases} \prod_{i=1}^{c} h_q(\mathbf{x}, \mathbf{m}^{(i)}, \mathbf{s}^{(i)}) & \text{if } \mathbf{x} \in \mathcal{F} \\ -\infty & \text{otherwise} \end{cases} \tag{4}$$

The function h_q defines the degree to which $\mathbf{x}$ fulfills the constraints, and is defined as follows:

$$h_q(\mathbf{x}, \mathbf{m}^{(i)}, \mathbf{s}^{(i)}) = \max_{l<L} \left(\sum_{j=1}^{k} \mathbb{1}\left[x_{l+s^{(j)}} = m_i^{(j)} \right] \right) // \binom{k}{q} / q \tag{5}$$

Setting $q = k$ corresponds to a dense signal which increments $h_q(\mathbf{x}, \mathbf{m}^{(i)}, \mathbf{s}^{(i)})$ whenever an additional element of any motif has been fulfilled. We provide additional details in Appendix A.1 about how to ensure that all motifs are jointly satisfiable and that there exists at least one feasible solution that attains the optimal value of 1.0. We also provide further evidence in Appendix A.1 that f is difficult to optimize for a standard evolutionary algorithm, even with small L, k, and c values.

4 Method

4.1 Bi-level Optimization

We formulate our task as a *sequence editing* problem, wherein the optimizer intakes a sequence x and must output an edited version y such that $f(y) > f(x)$. Not only is this the setting of common biophysical optimization tasks such as protein design, but LLMs have been shown to be highly effective at iterative refinement (Padmakumar et al., 2023; Madaan et al., 2024). We can then formulate the larger task of adapting LLMs for discrete sequence optimization as a bi-level optimization problem:

$$\begin{aligned} &\max_{\substack{x \sim \mathcal{X} \\ y \sim \pi_{\theta^*}(\cdot|x)}} \quad f(y) \\ &\text{subject to } \theta^* \in \arg\max_{\theta} \quad \mathop{\mathbb{E}}_{\substack{x \sim \mathcal{X} \\ y \sim \pi_{\theta^*}(\cdot|x)}} \quad f(y) \end{aligned} \tag{6}$$

The outer loop consists of finding θ^* that maximizes the expected score of sequences y generated by π_{θ^*} and the inner loop consists of finding the highest scoring y generated by π_{θ^*}. To more closely reflect the setting of real-world feedback cycles, we assume access to the oracle scoring function f only in the outer loop. This mimics the case where laboratory validation provides data for the model to be trained on. In the inner loop, the LLM must generate, rank, and filter candidate sequences without immediate access to f.

The high level algorithm is given in Algorithm 1. The optimizer is seeded with sequences generated by n_0 rounds of a genetic algorithm (details in Appendix A.3.3). These initial seeds are scored and used as the training dataset for the first iteration of the outer loop. After training, we enter the inner loop where π_{θ_i} generates and iteratively refines new candidate sequences using seeds sampled from its training dataset. During the iterative refinement process, the LLM iteratively ranks its outputs from the previous iterations and edits the highest ranking sequences. To avoid generator collapse, we automatically increment the sampling temperature if the average Hamming distance between the LLM inputs and outputs from the last round fall below a certain threshold. A subsample of k of these refined sequences are scored and used to create the training dataset for the next iteration of the outer loop. As such, no more than k new test function evaluations are required in each optimizer step. To create the training dataset for the next iteration, we adapt PropEn (Tagasovska et al., 2024), a dataset matching algorithm that implicitly guides the model towards improved regions of the dataset space. The detailed algorithms for DATASETFORMATTING, ITERATIVEREFINEMENT, and FILTER are given in Appendix A.3.

4.2 LLM Training

To train the LLM in the outer loop of the optimization algorithm (step 9 of Algorithm 1), we consider standard offline LLM training algorithms such as supervised finetuning (SFT) and direct preference optimization (Rafailov et al., 2023, ;DPO), in addition to a novel objective MargE. We refer to these three variants of LLOME as LLOME-SFT, LLOME-DPO, and LLOME-MARGE, respectively. We briefly summarize SFT and DPO in Section 2 and describe the rationale for the design of MargE below.

Margin-Aligned Expectation (MargE) Training Although DPO has recently become a popular preference learning method due to its relative simplicity and competitive results, its offline contrastive objective suffers from a number of drawbacks. Firstly, since DPO optimizes for an off-policy objective, a DPO-trained model rapidly over-optimizes, resulting in generations that decline in quality despite continued improvements in offline metrics (Rafailov et al., 2024a; Chen et al., 2024b). DPO models also fail at ranking text according to human preferences (Chen et al., 2024b) and tend to decrease the probability mass assigned to preferred outputs (Pal et al., 2024; Rafailov et al., 2024b; Feng et al., 2024; Pang et al., 2024). As training continues, DPO generations also increase in length, even if the quality of the generations does not necessarily improve (Singhal et al., 2024). Additionally, when the reference model already performs well on a particular subset of the input domain, DPO cannot achieve the optimal policy without deteriorating performance on that subset (Hu et al., 2024). Lastly, DPO does not make use of absolute reward values – instead, it simply assumes that $r(x, y_w) > r(x, y_l)$ for all (x, y_w, y_l) in the training dataset, but does not use information about *how much better* y_w is than y_l.

RLHF, on the other hand, involves steep technical complexity and frequently exhibits training instabilities (Zheng et al., 2023a; Wang et al., 2024; Casper et al., 2023). Hu et al. (2024); Korbak et al. (2022b) additionally show that RLHF's objective interpolates between π_{Ref} and a degenerate distribution π^{δ} that places all probability mass on a single reward-maximizing sequence, thereby promoting generator collapse. Indeed, much past work (Kirk et al., 2024; Zhou et al., 2024; Padmakumar & He, 2024) has illustrated the low diversity of RLHF-generated text.

To attempt to resolve these issues, we propose Margin-Aligned Expectation (MargE), a training objective that smoothly interpolates between the reference model π_{Ref} and the optimal policy π^* and effectively incorporates information about the true reward margin. Like Hu et al. (2024), we propose an objective that takes the following general form:

$$\arg\min_{\theta} \left[\mathbb{KL}(\tilde{\pi}_\theta \| \pi^*) + \lambda \mathbb{KL}(\pi_{\text{Ref}} \| \tilde{\pi}_\theta)\right] \tag{7}$$

where π^* is the target reward distribution and $\tilde{\pi}_\theta$ is the length-normalized version of π_θ. If we apply importance sampling and use a Boltzmann function of the reward as π^*, we obtain:

$$\mathcal{L}_{\text{MargE}}(\pi_\theta, \pi_{\text{Ref}}; \mathbb{D}_x) = \mathop{\mathbb{E}}_{\substack{x\sim\mathbb{D}_x,\\ y\sim\pi_{\text{Ref}}(\cdot|x)}} \left[\frac{\pi_\theta(y|x)}{\pi_{\text{Ref}}(y|x)} \left(\frac{\log \pi_\theta(y|x)}{|y|} - r(x,y)\right) - \lambda \frac{\log \pi_\theta(y|x)}{|y|}\right]. \tag{8}$$

For the full derivation, see App. A.2. We additionally devise a reward function for the sequence edit task at hand. It may seem natural to devise a reward function that is exactly equal to the score function

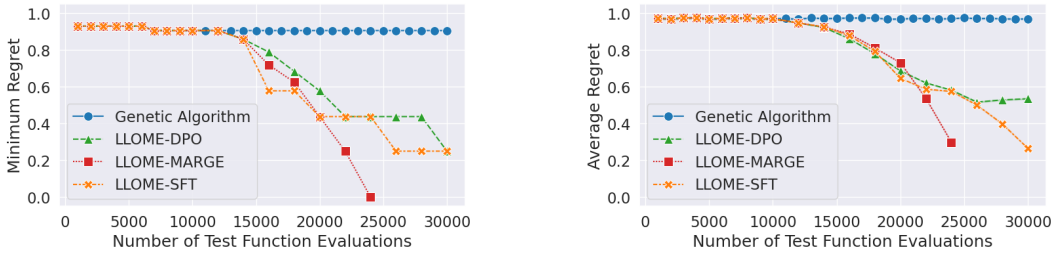
$f(x)$. However, the goal of our model is to produce a y with a **better** score than the input x – therefore, the reward should reflect whether the output is a relative improvement. However, simply setting $r(x, y) = f(y) - f(x)$ would also result in unintended behavior. Since a normalized Bradley-Terry policy is translation-invariant with respect to the rewards, we would have $\pi(y|x) = \pi(y|z)$ for all pairs $x, z \in \mathrm{dom}(f)$ for this particular reward function. (See Lemma 1 for a detailed derivation.) But if $f(x) \neq f(z)$, then y would constitute a greater improvement for one input than the other, which implies that we should not have $\pi(y|x) = \pi(y|z)$.

To avoid this issue, we instead devise a reward function that only rewards a policy for generating a sequence y with higher score than a seed sequence x and assigns 0 reward otherwise:

$$r(x, y) = \begin{cases} f(y) - f(x) & \text{if } f(y) > f(x) \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

In this case, we set $f(x) = 0$ for all $x \notin \mathcal{F}$ to avoid involving ∞'s in our calculations. When the policy produces an improved sequence (a sequence with higher score), the reward is equal to the difference in scores. Zero reward is given for sequences with equal or lower score. Notably, this reward function is similar to a margin loss.

5 Experiments



(a) Minimum regret achieved.

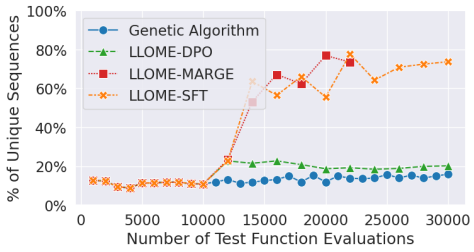
(b) Average regret.

Figure 2: Minimum regret achieved as a function of the number of test function evaluations (2a) and average regret of the selected generations in each round of the LLM inner loop (2b). The LLOME-MARGE line ends early because it achieves the optimal solution by 24K test function evaluations. The first 10K test function evaluations of every method correspond to the seeds provided by the genetic algorithm. The shaded regions in (2b) represent the mean $\pm$ two standard errors.

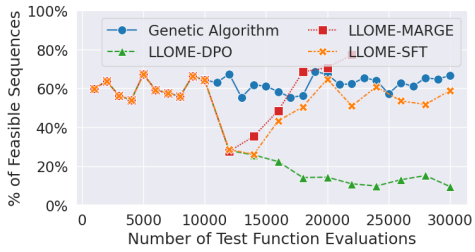
LLOME Details We compare the performance of three different variants of LLOME (LLOME-SFT, LLOME-MARGE, LLOME-DPO) against that of a genetic algorithm. The details of the genetic algorithm are given in Appendix A.3.3 and the training details for SFT, DPO, and MargE are given in Appendix A.4. Each LLOME variant is seeded with data from 10 rounds of the genetic algorithm (*i.e.*, $n_0 = 10$). For LLOME-MARGE and LLOME-DPO, one round of SFT is trained before continuing with MargE and DPO training in future iterations. All three variants use the pre-trained model PYTHIA 2.8B (Biderman et al., 2023) as the base model. During the TRAIN step of each iteration of LLOME (step 9 of Alg. 1), we train the current checkpoint for one epoch. Lastly, we use $k = 2000$ test function evaluations per iteration of LLOME.

Ehrlich Function Details Unless stated otherwise, we evaluate LLOME on an Ehrlich function with $L = 32$, $|\mathcal{V}| = 32$, $c = 4$, and $k = q = 4$.

LLMs generate sequences with low regret using few oracle labels. In Figure 2, we show both the average and minimum regret of LLOME. While all three LLOME variants achieve significantly lower minimum regret values than the genetic algorithm does, LLOME-MARGE is the only variant to discover the optimal solution and achieve a minimum regret of 0. However, the regret is based solely off the score of the output sequence (*i.e.* $\mathrm{regret}(x, y) = 1 - f(y)$ for $y \sim \pi_\theta(\cdot \mid x)$) and does not directly reflect how well the model improves the score of the input sequence. The reward, as given by Eq. 9, provides a more clear picture of the LLM's ability to improve upon the input



(a) Percent of sequences generated by each method that are unique.



(b) Percent of unique sequences generated by each method that are also feasible.

Figure 3: The percentage of generated sequences that are unique or feasible. The lines for LLOME-MARGE end early because LLOME-MARGE discovers the optimal solution early.

sequence. In Figure 11, we show the average and maximum reward achieved by each LLOME variant. While all three variants achieve similar maximum reward throughout all iterations, LLOME-SFT and LLOME-MARGE achieve significantly higher average reward than LLOME-DPO.

SFT- and MargE-trained LLMs generate unique and feasible sequences, but DPO-trained LLMs struggle. Successfully solving a highly-constrained optimization problem requires that the LLM be able to generate a diverse array of feasible sequences. Compared to the genetic algorithm, Figure 3a shows that all three LLOME variants produce a higher percentage of unique sequences. However, Figure 3b indicates that LLOME-SFT and LLOME-MARGE often experience an initial dip in the percentage of feasible outputs before learning how to generate sequences in the feasible space. LLOME-DPO, however, does not improve either the diversity or the feasibility of its outputs even when provided with more oracle labels. Like other past works (Pal et al., 2024; Rafailov et al., 2024b; Feng et al., 2024; Pang et al., 2024), we observe that the likelihood assigned by the DPO-trained LLM to both y_w and y_l continuously declines throughout training, implying that probability mass is moved to sequences outside of the training data distribution. Since the percentage of infeasible sequences generated by LLOME-DPO increases over multiple iterations, it is likely that DPO moves some probability mass to infeasible regions of the sample space. As such, DPO may be ill-suited for training LLMs to adhere to strict constraints.

Although LLMs are capable of generating new feasible sequences, we also find that they suffer from the classic explore versus exploit tradeoff. When the LLM makes a larger edit to the input sequence, the output is less likely to be feasible (Fig. 12). For both LLOME-SFT and LLOME-MARGE, an edit larger than 0.3 Hamming distance away from the input is less than 20% likely to be feasible. Since the Δx threshold we set for the PropEn dataset formatting algorithm (Alg. 2) was 0.25, this is unsurprising. The LLM has never been trained on edits of larger than 0.25 Hamming distance. Overall, LLOME-MARGE exhibits the best tradeoff of all methods, producing the largest proportion of unique and feasible sequences for the smallest budget of test function evaluations (Fig. 3) and for moderately sized edits of Hamming distance ≤ 0.3 (Fig. 12).

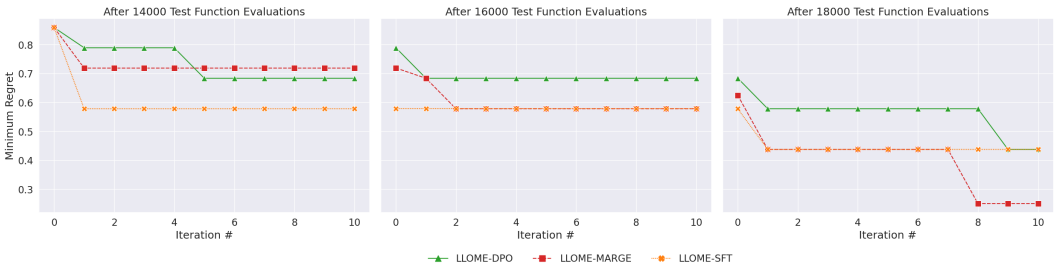


Figure 4: Minimum regret of sequences generated during the LLM inner loop, at each iteration of the iterative refinement process. The titles reflect the number of oracle labels that each LLM has been trained on. We defer the remaining plots for all other rounds of LLOME to App. A.5, Fig. 13.

At inference time, LLMs can iteratively extrapolate beyond their training distributions. Extrapolating beyond the training distribution is a well-known machine learning problem (Bommasani et al., 2021; Press et al., 2022; Li et al., 2024), especially without explicit guidance provided at inference time. Although some prior work has shown LLMs to be effective at iteratively generating sequences that monotonically increase a particular attribute to values beyond the training distribution (Chan et al., 2021; Padmakumar et al., 2023), much of this work focuses on simpler tasks such as increasing the positive sentiment of text or decreasing the $\Delta\Delta G$ of a well-studied protein. Since optimizing an Ehrlich function requires satisfying multiple constraints in addition to generating sequences that lie within a small feasible region, we posit that this evaluation is a more challenging assessment of LLMs' inference-time extrapolation capabilities.

We display the iterative refinement results of LLOME's inner loop in Fig. 4, which suggest that LLMs iteratively produce edits that significantly reduce regret. However, the first few edits frequently improve the sequence whereas later edits are less likely to be helpful. This suggests that LLM's inference-time extrapolative capabilities are limited – without further training or explicit guidance, LLMs may be unable to continuously improve a given sequence beyond a certain threshold. By alternating between optimizing the model's parameters and optimizing the model's outputs, we provide a sample-efficient method for iteratively bootstrapping the model's extrapolative abilities using its own generations.

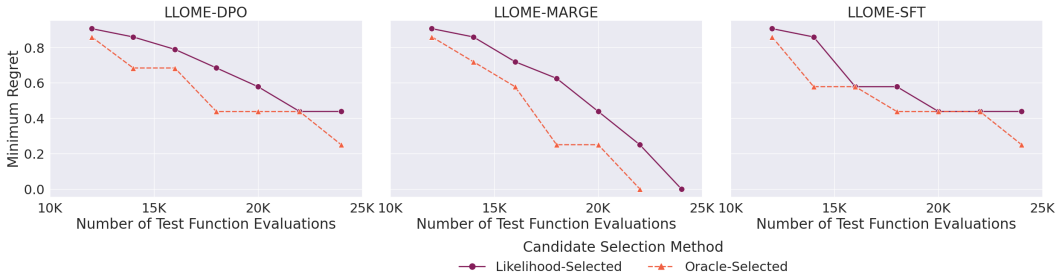
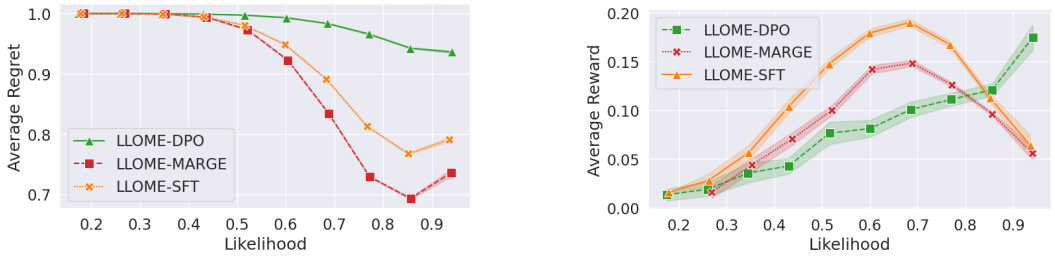


Figure 5: Minimum regret of candidates selected by either LLM likelihood or the oracle. Since the first 10K test function evaluations are seeds derived from the genetic algorithm, we show only candidates generated after the first 10K.



(a) Model likelihood vs. regret of generations.

(b) Model likelihood vs. reward of generations.

Figure 6: Calibration curves comparing likelihood versus the average regret (6a) and reward (6b) of each method's generated sequences. Shaded regions represent the mean $\pm$ two standard errors.

LLMs are moderately effective at ranking their own outputs. The iterative refinement process requires that the LLM rank and filter its own outputs (Alg. 4), but we have not yet considered how effective the LLM is at selecting the best candidates. When compared to oracle selection (*i.e.* using the ground-truth Ehrlich score to select candidates), we find that the likelihood method often selects high-scoring candidates, but not necessarily the *highest* scoring ones (Fig. 5). To better understand this gap between likelihood selection and oracle selection, we examine the calibration curves of average likelihood versus regret and reward on LLOME-generated sequences in Fig. 6. Although it appears that LLOME-DPO is the only variant to exhibit a calibration curve that monotonically decreases and increases with regret and reward, respectively, LLOME-DPO also produces the fewest sequences with regret outside of the training distribution. Calibration becomes more difficult when

evaluated on more sequences outside the training distribution, as is the case for LLOME-MARGE and LLOME-SFT. While SFT- and MargE-trained LLMs generally assign higher likelihood to lower-regret sequences than DPO-trained LLMs (Fig. 6a), they also exhibit some degree of overconfidence. Indeed, we observe that the likelihood and reward of SFT- and MargE-generated sequences appear to be *inversely* correlated for likelihoods exceeding 0.7 (Fig. 6b). Since these higher-likelihood sequences also correspond to the lowest regret values (Fig. 6a), we hypothesize that LLMs may become increasingly miscalibrated as their outputs extend beyond the training distribution.

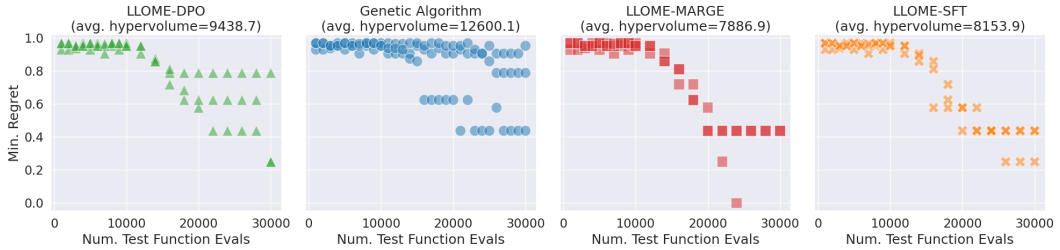
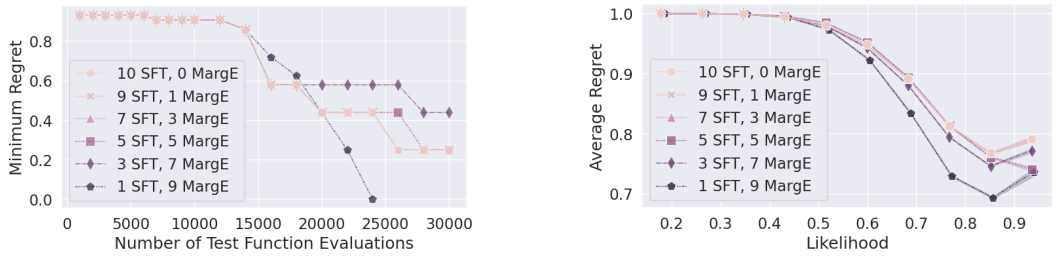


Figure 7: Pareto frontiers of evaluation budget vs. minimum regret for a variety of LLOME hyperparameter settings, Ehrlich functions, and seed datasets. The average hypervolume refers to the average (number of test function evaluations $\times$ minimum regret). Lower hypervolume is better.

How sensitive is LLOME to hyperparameters, test function difficulty, and seed dataset? An important aspect of an optimization algorithm is how robust it is to hyperparameter settings, problem difficulty, and choice of seed examples. To explore the robustness of LLOME, we evaluate LLOME on two different Ehrlich functions, two seed datasets (generated by two instances of the genetic algorithm with different p_m values), and varying budgets of test function evaluations. The Ehrlich functions f_1 and f_2 have parameters $L = 32, |\mathcal{V}| = 32, c = k = q = 4$ and $L = 16, |\mathcal{V}| = 32, c = k = q = 4$. The genetic algorithms have $p_m = 0.005$ and $p_m = 0.01$, and the budget of test function evaluations varies from 1K to 30K. We present the Pareto frontiers and corresponding hypervolumes of these experiments in Fig. 7. While all three LLOME variants exhibit significantly lower average hypervolume than the genetic algorithm, LLOME-MARGE is the most sample efficient.



(a) Minimum regret for multi-stage SFT+MargE training.

(b) Calibration curve of likelihood vs. regret for multi-stage SFT+MargE training.

Figure 8: Minimum regret and calibration curves for multi-stage LLM training.

When is explicit reward information required during training? Since we have observed that LLOME-SFT and LLOME-MARGE perform similarly up to the point of 20K test function evaluations (Fig. 2a), we might hypothesize that incorporating explicit reward values into the training objective is only necessary once the LLM is closer to the optimum. Since MargE requires a larger memory footprint than SFT (due to the need to store and compute likelihoods with both π_θ and π_{Ref}), training the LLM with further rounds of SFT before switching to MargE would be more efficient than relying mostly on MargE training. We test this hypothesis via a multi-stage pipeline, where early rounds of LLOME use SFT training, and later rounds use MargE. We keep the total number of LLM training rounds constant at 10 (unless the LLM finds the optimal solution earlier) and vary the proportion of rounds that use SFT versus MargE, as shown in Fig. 8. We refer to the pipeline with i rounds of SFT training followed by j rounds of MargE training as LLOME-SFT$_i$-MARGE$_j$. We find that LLOME-SFT$_1$-MARGE$_9$ not only is the sole pipeline to find the optimal solution (Fig. 8a), but also

achieves the best calibration curve (Fig. 8b). In contrast, switching from SFT to MargE training at an intermediate point results in the worst performance. It appears that SFT and MargE may have conflicting loss landscapes – if we first train for a few rounds with the SFT loss, subsequently switching to MargE training impedes the LLM's progress. We additionally explore in App. A.5 the effects of incorporating the reward in different ways during training.

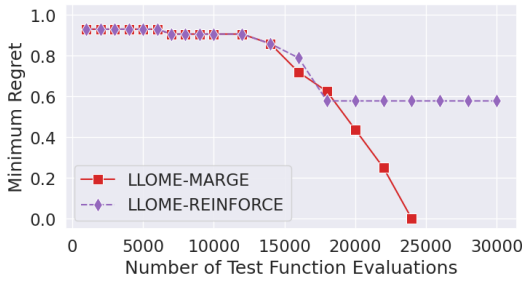


Figure 9: Minimum regret achieved by LLOME-MARGE and LLOME-REINFORCE.

Does fulfilling the Strong Interpolation Criteria (SIC) matter for LLM training? We previously proposed a new LLM training objective, MargE, for the purposes of easily incorporating reward information while fulfilling SIC (Hu et al., 2024). However, there also exist policy gradient methods such as REINFORCE (Williams, 1992) that directly optimize for maximal reward without the steep complexity of RLHF (Stiennon et al., 2020; Ziegler et al., 2019). Although these policy gradient methods do not fulfill SIC, they have been shown to perform comparably to RLHF (Stiennon et al., 2020; Ziegler et al., 2019) and RL-free variants such as DPO (Rafailov et al., 2023) and RAFT (Dong et al., 2023). The primary difference between MargE and a policy gradient method like REINFORCE, then, is that MargE attempts to smoothly interpolate between π^* and π_{Ref}, whereas the KL-regularized REINFORCE algorithm interpolates between π^δ (the degenerate reward-maximizing distribution) and π_{Ref}. To test whether this difference has a meaningful impact on an LLM's optimization capabilities, we compare LLOME-MARGE with LLOME-REINFORCE, where we use a KL-regularized version of REINFORCE with the same importance sampling and self-normalization techniques as in MargE:

$$\mathcal{L}_{\text{REINFORCE}}(\pi_\theta, \pi_{\text{Ref}}; \mathbb{D}_x) = \mathop{\mathbb{E}}_{\substack{x\sim\mathbb{D}_x,\\ y\sim\pi_{\text{Ref}}(\cdot|x)}} \left[-\frac{\pi_\theta(y|x)}{\pi_{\text{Ref}}(y|x)} r(x,y) \log \pi_\theta(y|x) - \lambda \frac{\log \pi_\theta(y|x)}{|y|} \right] \tag{10}$$

The implications of this objective are intuitive – greater weight is placed on the loss for higher reward examples, and vice versa. However, in the limit of $\lambda \to 0$, we have $\pi_{\theta^*_{\text{REINFORCE}}} = \arg\max_{\pi_\theta} \mathbb{E}_{y\sim\pi_\theta(\cdot|x)} r(x,y)/\pi_{\text{Ref}}(y|x)$, which is a degenerate distribution that places all probability mass on $(x', y') \in \arg\max_{(x,y)} r(x,y)/\pi_{\text{Ref}}(y|x)$, thereby violating SIC. By inspecting Fig. 9, we observe that although LLOME-REINFORCE does decrease the regret to some degree, it plateaus early and does not reach optimality.

5.1 Related Work

Our work combines insights from multiple areas of research, including discrete sequence black-box optimization, controllable text generation, and LLMs for optimization and scientific discovery.

Discrete Sequence Black-Box Optimization Many algorithms for discrete sequence optimization take inspiration from *directed evolution* (Arnold, 1998), a combination of random mutagenesis (a means to generate variants of the current solution) and high throughput screening (discriminative selection pressure). Researchers have explored many types of variant generators, including random mutagenesis (Back, 1996; Sinai et al., 2020), reinforcement learning (Angermueller et al., 2020), denoising with explicit discriminative guidance (Stanton et al., 2022; Maus et al., 2022; Gruver et al., 2024), and denoising with implicit discriminative guidance (Tagasovska et al., 2024). While these algorithms are all very general *in principle*, in practice a substantial amount of effort is required to actually implement these algorithms for new problems due to changes in the problem setup. In this work we investigate whether LLMs offer a path to a solution that can rapidly be applied to new problems with minimal engineering effort.

LLMs for Optimization and Scientific Discovery Much of the work on LLMs for optimization has been inspired by the design of classic black-box optimizers (BBO) such as evolutionary algorithms, bayesian optimizers, and random search methods. BBOs are characterized by a lack of information about the true objective function. Instead, only inputs and their corresponding objective values are provided to the optimizer, with no gradient or priors about the objective. Since these optimization problems are often expressible in formal mathematical, logical, or symbolic terms, many initial attempts at LLMs for optimization used the LLMs to first translate a natural language description of the problem into code or a modeling language, prior to passing this formalization into an auxiliary solver (Ramamonjison et al., 2022; Ahmed & Choudhury, 2024; Mittal et al., 2024; AhmadiTeshnizi et al., 2024). This approach was often quite effective, as it utilized the wealth of past work on LLMs for named entity recognition (Amin & Neumann, 2021; Ushio & Camacho-Collados, 2021; Wang et al., 2023b; Yan et al., 2019; Arkhipov et al., 2019, ; NER), semantic parsing (Drozdov et al., 2023; Gupta et al., 2018; Shaw et al., 2019; Shao et al., 2020; Rongali et al., 2020; Shi et al., 2021; Chen et al., 2022a; Stengel-Eskin et al., 2021), and code generation (Chen et al., 2021; Nijkamp et al., 2023b,a; Zhang et al., 2023c; Poesia et al., 2022). The NL4OPT competition (Ramamonjison et al., 2022), for example, decomposed LLMs-for-BBO into two subtasks: (1) the translation of a natural language description into a graph of entities and relations between the entities, and (2) a formalization of the optimization problem into a canonical logical reperesentation that could be solved by many commercial solvers. Winning approaches often employed ensemble learning (He et al., 2022; Wang et al., 2023a; Ning et al., 2023; Doan, 2022), adversarial learning (Wang et al., 2023a; Ning et al., 2023), and careful data pre-/post-processing and augmentation (He et al., 2022; Ning et al., 2023; Jang, 2022). Later work demonstrated that pre-trained LLMs like GPT-4 could achieve competitive performance on NLP4OPT without the NER stage (Ahmed & Choudhury, 2024), though the F1 score still trailed behind the state-of-the-art translation+NER approaches from the winning NL4OPT entries. Outside of NL4OPT, Gao et al. (2022), He-Yueya et al. (2023), AhmadiTeshnizi et al. (2024), and Mittal et al. (2024) use an LLM to formalize math, mixed integer linear programming, and combinatorial reasoning problems from a natural language description before offloading the solving to a Python interpreter or symbolic solver. Each work notes that the LLM performs better in this decomposed framework than through prompting alone.

Yet other approaches tackle LLMs-for-optimization directly with the LLM, without additional solvers. As Song et al. (2024) argues, LLMs offer both powerful in-context learning abilities and a flexible natural-language interface capable of expressing a wide variety of problems. Many techniques embed the LLM in an evolutionary algorithm, using the LLM as a mutation or crossover operator (Chen et al., 2024a; Guo et al., 2024b; Meyerson et al., 2024; Lehman et al., 2023; Nasir et al., 2024; Liu et al., 2024a; Lange et al., 2024; Liu et al., 2024b; Romera-Paredes et al., 2023). In this setting, the LLM provides a diversity of samples while the evolutionary algorithm guides the search towards high-fitness regions. This strategy is a form of bi-level optimization, in which two nested optimization problems (one nested within the other) are solved in alternation. It is common for the outer loop to optimize the model's parameters, and for the inner loop to optimize the model's outputs (Chen et al., 2022b; Guo et al., 2024c). Ma et al. (2024) combine this approach with feedback from physical simulations in the inner optimization loop to enable LLMs to complete various scientific optimization tasks, such as constitutive law discovery and designing molecules with specific quantum mechanical properties. Although their use of an LLM in a bi-level optimization loop is similar to ours, they directly train their parameters using differentiable simulations whereas we do not assume access to the gradients of the ground truth rewards. Our work additionally explores various aspects of LLM training that allow the LLM to improve its optimization abilities despite not having access to ground-truth rewards in the inner loop. Lastly, other approaches avoid gradient optimization altogether and focus purely on prompt-based optimization, demonstrating success on diverse tasks such as hill-climbing (Guo et al., 2024a), Newton's method (Nie et al., 2023), hyperparameter search (Zhang et al., 2023a), and prompt engineering (Khattab et al., 2024; Pryzant et al., 2023; Yang et al., 2024).

Controllable Text Generation Controllable text generation (CTG) is a special case of optimization. Rather than searching for sequences that maximize the objective function, the goal is to produce a sequence with a particular attribute value (*e.g.* having a certain number of words, a more positive sentiment than the input, or a particular biological motif). In some aspects, this may be an easier task – the optimizer need not generate an optimal sequence that is likely outside the distribution of its training data. In others, this may also be more difficult. Targeting particular attribute values

requires precision and fine-grained knowledge of the shape of the entire objective function, rather than only the neighborhood of the optima. Although LLM prompting is in itself considered a form of CTG (Radford et al., 2019; Brown et al., 2020), prompting alone tends to offer better control for more open-ended, higher level instructions (*e.g.* "write a story in the style of Shakespeare") than for fine-grained constraints (*e.g.* "rewrite this sentence using only 10 words") (Carlsson et al., 2022).

One common approach is to use *control codes*, or unique strings pre-pended in front of a training example that indicates which attribute is represented in the example's target (Keskar et al., 2019; Padmakumar et al., 2023; Raffel et al., 2020; Madani et al., 2023). This approach is typically less generalizable to new attributes or instructions, due to the need to re-train the model. However, more recent work has shown that LLMs are capable of learning to use these control codes *in-context* (Zheng et al., 2023c; Zhou et al., 2023), simplifying the process by which new attributes can be introduced. A popular alternative technique is inference-time guidance, which uses auxiliary tools (Pascual et al., 2021) or models (Dathathri et al., 2020; Liu et al., 2021; Deng & Raffel, 2023; Dekoninck et al., 2024) to guide the LLM decoding process.

6 Conclusions

Our work is a response to the lack of both non-trivial synthetic benchmarks for biophysical sequence optimizers and rigorous analyses of how LLMs perform on these highly-constrained optimization tasks in realistic settings. Our proposed test functions bear significant geometric similarities to real biophysical sequence optimization tasks and allow for rapid iteration cycles. In addition, although a wealth of work exists that adapts LLMs for biophysical tasks, few have studied the abilities of LLMs to adhere to hard constraints in realistic bi-level optimization settings. To that end, we propose and analyze LLOME, a framework for incorporating LLMs in bilevel optimization algorithms for highly constrained discrete sequence optimization problems. We show that LLOME discovers lower-regret solutions than a genetic algorithm, even with very few test function evaluations. When combined with MargE training, LLOME is significantly more sample efficient than LLOME-SFT or LLOME-DPO, demonstrating its potential to be useful in data-sparse lab settings. Our findings also highlight that LLMs can robustly extrapolate beyond their training data, but are occasionally miscalibrated and benefit from training with ground truth rewards.

7 Acknowledgements

We thank Natasa Tagasovska and the Prescient Design LLM team for valuable discussions and feedback during the development of this project. The authors also thank the Prescient Design Engineering team for providing HPC and consultation resources that contributed to the research results reported within this paper.

References

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. *arXiv preprint arXiv:2402.10172*, 2024.

Tasnim Ahmed and Salimur Choudhury. Lm4opt: Unveiling the potential of large language models in formulating mathematical optimization problems, 2024. URL `https://arxiv.org/abs/2403.01342`.

Saadullah Amin and Guenter Neumann. T2NER: Transformers based transfer learning framework for named entity recognition. In Dimitra Gkatzia and Djamé Seddah (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pp. 212–220, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-demos.25. URL `https://aclanthology.org/2021.eacl-demos.25`.

Christof Angermueller, David Belanger, Andreea Gane, Zelda Mariet, David Dohan, Kevin Murphy, Lucy Colwell, and D Sculley. Population-based black-box optimization for biological sequence design. In *International conference on machine learning*, pp. 324–334. PMLR, 2020.

Mikhail Arkhipov, Maria Trofimova, Yuri Kuratov, and Alexey Sorokin. Tuning multilingual transformers for language-specific named entity recognition. In Tomaž Erjavec, Michał Marcińczuk,

Preslav Nakov, Jakub Piskorski, Lidia Pivovarova, Jan Šnajder, Josef Steinberger, and Roman Yangarber (eds.), *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*, pp. 89–93, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-3712. URL https://aclanthology.org/W19-3712.

Frances H Arnold. Design by directed evolution. *Accounts of chemical research*, 31(3):125–131, 1998.

Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

Kyle A Barlow, Shane O Conchuir, Samuel Thompson, Pooja Suresh, James E Lucas, Markus Heinonen, and Tanja Kortemme. Flex ddg: Rosetta ensemble-based estimation of changes in protein–protein binding affinity upon mutation. *The Journal of Physical Chemistry B*, 122(21): 5389–5399, 2018.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *ArXiv*, 2021. URL https://crfm.stanford.edu/assets/report.pdf.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Fredrik Carlsson, Joey Öhman, Fangyu Liu, Severine Verlinden, Joakim Nivre, and Magnus Sahlgren. Fine-grained controllable text generation using non-residual prompting. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6837–6857, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.471. URL https://aclanthology.org/2022.acl-long.471.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, Tony Wang, Samuel

Marks, Charbel-Raphaël Segerie, Micah Carroll, Andi Peng, Phillip Christoffersen, Mehul Damani, Stewart Slocum, Usman Anwar, Anand Siththaranjan, Max Nadeau, Eric J. Michaud, Jacob Pfau, Dmitrii Krasheninnikov, Xin Chen, Lauro Langosco, Peter Hase, Erdem Bıyık, Anca Dragan, David Krueger, Dorsa Sadigh, and Dylan Hadfield-Menell. Open problems and fundamental limitations of reinforcement learning from human feedback, 2023. URL https://arxiv.org/abs/2307.15217.

Alvin Chan, Ali Madani, Ben Krause, and Nikhil Naik. Deep extrapolation for attribute-enhanced generation. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 14084–14096. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/75da5036f659fe64b53f3d9b39412967-Paper.pdf.

Angelica Chen, David M. Dohan, and David R. So. Evoprompting: language models for code-level neural architecture search. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024a. Curran Associates Inc.

Angelica Chen, Sadhika Malladi, Lily H. Zhang, Xinyi Chen, Qiuyi Zhang, Rajesh Ranganath, and Kyunghyun Cho. Preference learning algorithms do not learn preference rankings. *arXiv preprint arXiv:2405.19534*, 2024b. URL https://arxiv.org/abs/2405.19534.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xiaoxue Chen, Tianyu Liu, Hao Zhao, Guyue Zhou, and Ya-Qin Zhang. Cerberus transformer: Joint semantic, affordance and attribute parsing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19649–19658, 2022a.

Yihan Chen, Benfeng Xu, Quan Wang, Yi Liu, and Zhendong Mao. Benchmarking large language models on controllable generation under diversified instructions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17808–17816, Mar. 2024c. doi: 10.1609/aaai.v38i16.29734. URL https://ojs.aaai.org/index.php/AAAI/article/view/29734.

Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Richard Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc' Aurelio Ranzato, Sagi Perel, and Nando de Freitas. Towards learning universal hyperparameter optimizers with transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 32053–32068. Curran Associates, Inc., 2022b. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/cf6501108fced72ee5c47e2151c4e153-Paper-Conference.pdf.

Tobiasz Cieplinski, Tomasz Danel, Sabina Podlewska, and Stanisław Jastrzebski. Generative models should at least be able to design molecules that dock well: A new benchmark. *Journal of Chemical Information and Modeling*, 63(11):3238–3247, 2023.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1edEyBKDS.

Jasper Dekoninck, Marc Fischer, Luca Beurer-Kellner, and Martin Vechev. Controlled text generation via language model arithmetic. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=SLw9fp4yI6.

Haikang Deng and Colin Raffel. Reward-augmented decoding: Efficient controlled text generation with a unidirectional reward model. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 11781–11791, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.721. URL https://aclanthology.org/2023.emnlp-main.721.

Xuan-Dung Doan. Vtcc-nlp at nl4opt competition subtask 1: An ensemble pre-trained language models for named entity recognition, 2022. URL https://arxiv.org/abs/2212.07219.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, KaShun SHUM, and Tong Zhang. RAFT: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=m7p5O7zblY.

Andrew Drozdov, Nathanael Schärli, Ekin Akyürek, Nathan Scales, Xinying Song, Xinyun Chen, Olivier Bousquet, and Denny Zhou. Compositional semantic parsing with large language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=gJW8hSGBys8.

Li Du, Lucas Torroba Hennigen, Tiago Pimentel, Clara Meister, Jason Eisner, and Ryan Cotterell. A measure-theoretic characterization of tight language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9744–9770, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.543. URL https://aclanthology.org/2023.acl-long.543.

Duanyu Feng, Bowen Qin, Chen Huang, Zheng Zhang, and Wenqiang Lei. Towards analyzing and understanding the limitations of dpo: A theoretical perspective, 2024. URL https://arxiv.org/abs/2404.04626.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2022.

Cristina Garbacea and Qiaozhu Mei. Why is constrained neural language generation particularly challenging?, 2022.

Nate Gruver, Samuel Stanton, Nathan Frey, Tim GJ Rudner, Isidro Hotzel, Julien Lafrance-Vanasse, Arvind Rajpal, Kyunghyun Cho, and Andrew G Wilson. Protein design with guided discrete diffusion. *Advances in neural information processing systems*, 36, 2024.

Pei-Fu Guo, Ying-Hsuan Chen, Yun-Da Tsai, and Shou-De Lin. Towards optimizing with large language models. In *Fourth Workshop on Knowledge-infused Learning*, 2024a. URL https://openreview.net/forum?id=vIU8LUckb4.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=ZG3RaNIsO8.

Zixian Guo, Ming Liu, Zhilong Ji, Jinfeng Bai, Yiwen Guo, and Wangmeng Zuo. Two optimizers are better than one: Llm catalyst empowers gradient-based optimization for prompt tuning, 2024c. URL https://arxiv.org/abs/2405.19732.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. Semantic parsing for task oriented dialog using hierarchical representations. *arXiv preprint arXiv:1810.07942*, 2018.

JiangLong He, Mamatha N, Shiv Vignesh, Deepak Kumar, and Akshay Uppal. Linear programming word problems formulation using ensemblecrf ner labeler and t5 text generator with data augmentations, 2022. URL https://arxiv.org/abs/2212.14657.

Joy He-Yueya, Gabriel Poesia, Rose E Wang, and Noah D Goodman. Solving math word problems by combining language models with symbolic solvers. *arXiv preprint arXiv:2304.09102*, 2023.

Xiangkun Hu, Tong He, and David Wipf. New desiderata for direct preference optimization. In *ICML 2024 Workshop on Models of Human Feedback for AI Alignment*, 2024. URL https://openreview.net/forum?id=Fgf0iAOb22.

Alissa M Hummer, Constantin Schneider, Lewis Chinery, and Charlotte M Deane. Investigating the volume and diversity of data needed for generalizable antibody-antigen $\delta\delta$g prediction. *bioRxiv*, pp. 2023–05, 2023.

Sanghwan Jang. Tag embedding and well-defined intermediate representation improve auto-formulation of problem description. *arXiv preprint arXiv:2212.03575*, 2022.

Elizabeth H Kellogg, Andrew Leaver-Fay, and David Baker. Role of conformational sampling in computing mutation-induced changes in protein structure and stability. *Proteins: Structure, Function, and Bioinformatics*, 79(3):830–838, 2011.

Nitish Shirish Keskar, Bryan McCann, Lav Varshney, Caiming Xiong, and Richard Socher. CTRL - A Conditional Transformer Language Model for Controllable Generation. *arXiv preprint arXiv:1909.05858*, 2019.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the effects of RLHF on LLM generalisation and diversity. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=PXD3FAVHJT.

Tomasz Korbak, Hady Elsahar, Germán Kruszewski, and Marc Dymetman. On reward maximization and distribution matching for fine-tuning language models, 2022a. URL https://openreview.net/forum?id=8f95ajHrIFc.

Tomasz Korbak, Ethan Perez, and Christopher L Buckley. Rl with kl penalties is better viewed as bayesian inference, 2022b. URL https://arxiv.org/abs/2205.11275.

Robert Lange, Yingtao Tian, and Yujin Tang. Large language models as evolution strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '24 Companion, pp. 579–582, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704956. doi: 10.1145/3638530.3654238. URL https://doi.org/10.1145/3638530.3654238.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. Evolution through large models. In *Handbook of Evolutionary Machine Learning*, pp. 331–366. Springer, 2023.

Haolong Li, Yu Ma, Yinqi Zhang, Chen Ye, and Jie Chen. Exploring mathematical extrapolation of large language models with synthetic data. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics ACL 2024*, pp. 936–946, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.55. URL https://aclanthology.org/2024.findings-acl.55.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. DExperts: Decoding-time controlled text generation with experts and anti-experts. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 6691–6706, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.522. URL https://aclanthology.org/2021.acl-long.522.

Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. Large language models as evolutionary optimizers. In *2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 2024a. doi: 10.1109/CEC60901.2024.10611913.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. *arXiv preprint arXiv:2402.03921*, 2024b.

Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B. Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. LLM and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 33940–33962. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/ma24m.html.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Ali Madani, Ben Krause, Eric R. Greene, Subu Subramanian, Benjamin P. Mohr, James M. Holton, Jose Luis Olmos, Caiming Xiong, Zachary Z. Sun, Richard Socher, James S. Fraser, and Nikhil Naik. Large language models generate functional protein sequences across diverse families. *Nature Biotechnology*, 41(8):1099–1106, January 2023. ISSN 1546-1696. doi: 10.1038/s41587-022-01618-2. URL http://dx.doi.org/10.1038/s41587-022-01618-2.

Sadhika Malladi. The hidden infinity in preference learning, July 2024. URL https://www.cs.princeton.edu/~smalladi/blog/2024/06/27/dpo-infinity/.

Natalie Maus, Haydn Jones, Juston Moore, Matt J Kusner, John Bradshaw, and Jacob Gardner. Local latent space bayesian optimization over structured inputs. *Advances in neural information processing systems*, 35:34505–34518, 2022.

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. *arXiv preprint arXiv:2405.14734*, 2024.

Elliot Meyerson, Mark J. Nelson, Herbie Bradley, Adam Gaier, Arash Moradi, Amy K. Hoover, and Joel Lehman. Language model crossover: Variation through few-shot prompting. *ACM Trans. Evol. Learn. Optim.*, September 2024. doi: 10.1145/3694791. URL https://doi.org/10.1145/3694791. Just Accepted.

Chinmay Mittal, Krishna Kartik, Mausam, and Parag Singla. Puzzlebench: Can llms solve challenging first-order combinatorial reasoning problems?, 2024. URL https://arxiv.org/abs/2402.02611.

Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 101:48, 2005.

Muhammad Umair Nasir, Sam Earle, Julian Togelius, Steven James, and Christopher Cleghorn. Llmatic: neural architecture search via large language models and quality diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1110–1118, 2024.

Allen Nie, Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. Importance of directional feedback for LLM-based optimizers. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. URL https://openreview.net/forum?id=QW4eGh5GT3.

Erik Nijkamp, Hiroaki Hayashi, Caiming Xiong, Silvio Savarese, and Yingbo Zhou. Codegen2: Lessons for training llms on programming and natural languages. *ICLR*, 2023a.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *ICLR*, 2023b.

Yuting Ning, Jiayu Liu, Longhu Qin, Tong Xiao, Shangzi Xue, Zhenya Huang, Qi Liu, Enhong Chen, and Jinze Wu. A novel approach for auto-formulation of optimization problems. *arXiv preprint arXiv:2302.04643*, 2023.

Art B. Owen. *Monte Carlo theory, methods and examples*. https://artowen.su.domains/mc/, 2013.

Vishakh Padmakumar and He He. Does writing with language models reduce content diversity?, 2024. URL https://arxiv.org/abs/2309.05196.

Vishakh Padmakumar, Richard Yuanzhe Pang, He He, and Ankur P Parikh. Extrapolative controlled sequence generation via iterative refinement. In *International Conference on Machine Learning*, pp. 26792–26808. PMLR, 2023.

Arka Pal, Deep Karkhanis, Samuel Dooley, Manley Roberts, Siddartha Naidu, and Colin White. Smaug: Fixing failure modes of preference optimisation with dpo-positive, 2024. URL https://arxiv.org/abs/2402.13228.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative reasoning preference optimization, 2024. URL https://arxiv.org/abs/2404.19733.

Damian Pascual, Beni Egressy, Clara Meister, Ryan Cotterell, and Roger Wattenhofer. A plug-and-play method for controlled text generation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 3973–3997, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.334. URL https://aclanthology.org/2021.findings-emnlp.334.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=KmtVD97J43e.

Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=R8sQPpGCv0.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=HPuSIXJaa9.

Rafael Rafailov, Yaswanth Chittepu, Ryan Park, Harshit Sikchi, Joey Hejna, Bradley Knox, Chelsea Finn, and Scott Niekum. Scaling laws for reward model overoptimization in direct alignment algorithms, 2024a. URL https://arxiv.org/abs/2406.02900.

Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. From r to q^*: Your language model is secretly a q-function. In *First Conference on Language Modeling*, 2024b. URL https://openreview.net/forum?id=kEVcNxtqXk.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1), January 2020. ISSN 1532-4435.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht (eds.), *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 28 Nov–09 Dec 2022. URL https://proceedings.mlr.press/v220/ramamonjison23a.html.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, December 2023. ISSN 1476-4687. doi: 10.1038/s41586-023-06924-6. URL http://dx.doi.org/10.1038/s41586-023-06924-6.

Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. Don't parse, generate! a sequence to sequence architecture for task-oriented semantic parsing. In *Proceedings of the web conference 2020*, pp. 2962–2968, 2020.

Bo Shao, Yeyun Gong, Weizhen Qi, Guihong Cao, Jianshu Ji, and Xiaola Lin. Graph-based transformer with cross-candidate verification for semantic parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8807–8814, 2020.

Peter Shaw, Philip Massey, Angelica Chen, Francesco Piccinno, and Yasemin Altun. Generating logical forms from graph representations of text and entities. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 95–106, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1010. URL https://aclanthology.org/P19-1010.

Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. Learning contextual representations for semantic parsing with generation-augmented pre-training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 13806–13814, 2021.

Sam Sinai, Richard Wang, Alexander Whatley, Stewart Slocum, Elina Locane, and Eric D Kelsic. Adalead: A simple and robust adaptive greedy search algorithm for sequence design. *arXiv preprint arXiv:2010.02141*, 2020.

Prasann Singhal, Tanya Goyal, Jiacheng Xu, and Greg Durrett. A long way to go: Investigating length correlations in RLHF, 2024. URL https://openreview.net/forum?id=sNtDKdcI1f.

Xingyou Song, Yingtao Tian, Robert Tjarko Lange, Chansoo Lee, Yujin Tang, and Yutian Chen. Position: Leverage foundational models for black-box optimization. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=ea2MgKn3sV.

Samuel Stanton, Wesley Maddox, Nate Gruver, Phillip Maffettone, Emily Delaney, Peyton Greenside, and Andrew Gordon Wilson. Accelerating bayesian optimization for biological sequence design with denoising autoencoders. In *International Conference on Machine Learning*, pp. 20459–20478. PMLR, 2022.

Elias Stengel-Eskin, Kenton Murray, Sheng Zhang, Aaron Steven White, and Benjamin Van Durme. Joint universal syntactic and semantic parsing. *Transactions of the Association for Computational Linguistics*, 9:756–773, 2021.

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. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 3008–3021. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1f89885d556929e98d3ef9b86448f951-Paper.pdf.

Jiao Sun, Yufei Tian, Wangchunshu Zhou, Nan Xu, Qian Hu, Rahul Gupta, John Wieting, Nanyun Peng, and Xuezhe Ma. Evaluating large language models on controlled generation tasks. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3155–3168, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.190. URL https://aclanthology.org/2023.emnlp-main.190.

Nataša Tagasovska, Vladimir Gligorijević, Kyunghyun Cho, and Andreas Loukas. Implicitly guided design with propen: Match your data to follow the gradient, 2024. URL https://arxiv.org/abs/2405.18075.

Yunhao Tang, Daniel Zhaohan Guo, Zeyu Zheng, Daniele Calandriello, Yuan Cao, Eugene Tarassov, Rémi Munos, Bernardo Ávila Pires, Michal Valko, Yong Cheng, and Will Dabney. Understanding the performance gap between online and offline alignment algorithms, 2024. URL https://arxiv.org/abs/2405.08448.

Austin Tripp, Gregor N. C. Simm, and José Miguel Hernández-Lobato. A fresh look at de novo molecular design benchmarks. In *NeurIPS 2021 AI for Science Workshop*, 2021. URL https://openreview.net/forum?id=gS3XMun4cl_.

Asahi Ushio and Jose Camacho-Collados. T-NER: An all-round python library for transformer-based named entity recognition. In Dimitra Gkatzia and Djamé Seddah (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pp. 53–62, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-demos.7. URL https://aclanthology.org/2021.eacl-demos.7.

Binghai Wang, Rui Zheng, Lu Chen, Yan Liu, Shihan Dou, Caishuang Huang, Wei Shen, Senjie Jin, Enyu Zhou, Chenyu Shi, Songyang Gao, Nuo Xu, Yuhao Zhou, Xiaoran Fan, Zhiheng Xi, Jun Zhao, Xiao Wang, Tao Ji, Hang Yan, Lixing Shen, Zhan Chen, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. Secrets of rlhf in large language models part ii: Reward modeling, 2024. URL https://arxiv.org/abs/2401.06080.

Kangxu Wang, Ze Chen, and Jiewen Zheng. Opd@nl4opt: An ensemble approach for the ner task of the optimization problem, 2023a. URL https://arxiv.org/abs/2301.02459.

Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, and Guoyin Wang. Gpt-ner: Named entity recognition via large language models, 2023b. URL https://arxiv.org/abs/2304.10428.

Sean Welleck, Ilia Kulikov, Jaedeok Kim, Richard Yuanzhe Pang, and Kyunghyun Cho. Consistency of a recurrent language model with respect to incomplete decoding. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5553–5568, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.448. URL https://aclanthology.org/2020.emnlp-main.448.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, May 1992. ISSN 1573-0565. doi: 10.1007/bf00992696. URL http://dx.doi.org/10.1007/BF00992696.

Hang Yan, Bocao Deng, Xiaonan Li, and Xipeng Qiu. Tener: Adapting transformer encoder for named entity recognition, 2019. URL https://arxiv.org/abs/1911.04474.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

Weizhe Yuan, Ilia Kulikov, Ping Yu, Kyunghyun Cho, Sainbayar Sukhbaatar, Jason Weston, and Jing Xu. Following length constraints in instructions, 2024. URL https://arxiv.org/abs/2406.17744.

Michael Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. Using large language models for hyperparameter optimization. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023a. URL https://openreview.net/forum?id=FUdZ6HEOre.

Michael Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. Using large language models for hyperparameter optimization. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023b. URL https://openreview.net/forum?id=FUdZ6HEOre.

Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B. Tenenbaum, and Chuang Gan. Planning with large language models for code generation. In *The Eleventh International Conference on Learning Representations*, 2023c. URL https://openreview.net/forum?id=Lr8cOOtYbfL.

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, Limao Xiong, Lu Chen, Zhiheng Xi, Nuo Xu, Wenbin Lai, Minghao Zhu, Cheng Chang, Zhangyue Yin, Rongxiang Weng, Wensen Cheng, Haoran Huang, Tianxiang Sun, Hang Yan, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. Secrets of rlhf in large language models part i: Ppo, 2023a. URL https://arxiv.org/abs/2307.04964.

Rui Zheng, Shihan Dou, Songyang Gao, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Limao Xiong, Lu Chen, Zhiheng Xi, Yuhao Zhou, Nuo Xu, Wenbin Lai, Minghao Zhu, Rongxiang Weng, Wensen Cheng, Cheng Chang, Zhangyue Yin, Yuan Hua, Haoran Huang, Tianxiang Sun, Hang Yan, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. Secrets of rlhf in large language models part i: Ppo. 2023b.

Xin Zheng, Hongyu Lin, Xianpei Han, and Le Sun. Toward unified controllable text generation via regular expression instruction. In Jong C. Park, Yuki Arase, Baotian Hu, Wei Lu, Derry Wijaya, Ayu Purwarianti, and Adila Alfa Krisnadhi (eds.), *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1–14, Nusa Dua, Bali, November 2023c. Association for Computational Linguistics. doi: 10.18653/v1/2023.ijcnlp-main.1. URL https://aclanthology.org/2023.ijcnlp-main.1.

Jiayi Zhou, Jiaming Ji, Juntao Dai, and Yaodong Yang. Sequence to sequence reward modeling: Improving rlhf by language feedback, 2024. URL https://arxiv.org/abs/2409.00162.

Wangchunshu Zhou, Yuchen Eleanor Jiang, Ethan Wilcox, Ryan Cotterell, and Mrinmaya Sachan. Controlled text generation with natural language instructions. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 42602–42613. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/zhou23g.html.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

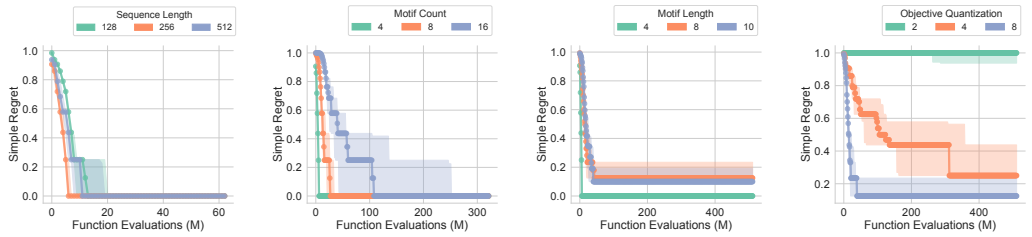


Figure 10: Here we show how the difficulty of the test problem can be controlled by varying Ehrlich function parameters, keeping the optimizer fixed to the robust GA baseline defined in Sec. A.3.3. Starting from a fixed set of reference parameters we vary each parameter individually. For this optimizer, the problem difficulty depends most strongly on the quantization parameter q. The x-axis is defined in *millions* (M) of Ehrlich function evaluations, demonstrating the difficulty of these Ehrlich functions, even for a small number of short motifs.

A Appendix

A.1 Ehrlich Functions

One major advantage of procedurally generating specific instances of Ehrlich functions is we can generate as many distinct instances of this test problem as we like. In fact it creates the possibility of "train" functions for algorithm development and hyperparameter tuning and "test" functions for evaluation simply by varying the random seed. These functions can also be defined with arbitrary levels of difficulty, as shown in Fig. 10. However, defining a random instance that is nevertheless provably solvable takes some care in the problem setup, which we now explain.

A.1.1 Constructing the Transition Matrix

Here we describe an algorithm to procedurally generate random ergodic transition matrices A with infeasible transitions. A finite Markov chain is ergodic if it is *aperiodic* and *irreducible* (since every irreducible finite Markov chain is positive recurrent). Irreducibility means every state can be reached with non-zero probability from every other state by some sequence of transitions with non-zero probability. We will ensure aperiodicity and irreducibility by requiring the zero entries of A to have a banded structure. For intuition, consider the transition matrix

$$\begin{bmatrix} 0.4 & 0.3 & 0 & 0.3 \\ 0.3 & 0.4 & 0.3 & 0 \\ 0 & 0.3 & 0.4 & 0.3 \\ 0.3 & 0 & 0.3 & 0.4 \end{bmatrix}$$

Recalling that v is the number of states, we can see that every state x communicates with every other state x' by the sequence $x \to (x+1) \mod v \to \cdots \to (x'-1) \mod v \to x'$. We also see that the chain is aperiodic since every state x has a non-zero chance of being repeated.

To make things a little more interesting we will shuffle (i.e. permute) the rows of a banded structured matrix (with bands that wrap around), but ensure that the diagonal entries are still non-zero. Note that permuting the bands does not break irreducibility because valid paths between states can be found by applying the same permutation action on valid paths from the unpermuted matrix. We will also choose the non-zero values randomly, using the shuffled banded matrix only as a binary mask B as

follows:

$$\overset{\text{(banded matrix)}}{\begin{bmatrix}1&1&0&1\\1&1&1&0\\0&1&1&1\\1&0&1&1\end{bmatrix}} \xrightarrow[\text{shuffle}]{} \begin{bmatrix}1&0&1&1\\1&1&1&0\\1&1&0&1\\0&1&1&1\end{bmatrix}, \xrightarrow[\text{diag=1}]{} \begin{bmatrix}1&0&1&1\\1&1&1&0\\1&1&1&1\\0&1&1&1\end{bmatrix} = B.$$

Now we draw the transition matrix starting with a random matrix with IID random normal entries, softmaxing with temperature $\tau > 0$ to make the rows sum to 1, applying the mask B, and renormalizing the rows by dividing by the sum of the columns after masking.

$$\begin{aligned} Z &= \overset{\text{(randn matrix)}}{\begin{bmatrix}+1.41&+1.67&-1.52&+0.63\\-0.35&+0.45&+0.86&-0.49\\+1.42&-1.31&-0.31&+1.43\\-0.02&+1.55&-0.26&+1.13\end{bmatrix}}, \\ &\xrightarrow[\text{softmax}]{} \begin{bmatrix}0.36&0.46&0.02&0.16\\0.13&0.30&0.45&0.12\\0.44&0.03&0.08&0.45\\0.10&0.49&0.08&0.33\end{bmatrix}, \\ &\xrightarrow[\odot B]{} \begin{bmatrix}0.36&0&0.02&0.16\\0.13&0.30&0.45&0\\0.44&0.03&0.08&0.45\\0&0.49&0.08&0.33\end{bmatrix}, \\ &\xrightarrow[\text{norm}]{} \begin{bmatrix}0.66&0&0.04&0.30\\0.15&0.34&0.51&0\\0.44&0.03&0.08&0.45\\0&0.55&0.09&0.36\end{bmatrix} = A. \end{aligned}$$

We can also verify that A is ergodic numerically by checking the Perron-Frobenius condition,

$$(A^m)_{ij} > 0,\ \forall i, j, \tag{11}$$

where $m = (v-1)^2 + 1$, $A^1 = A$, and $A^b = AA^{b-1}$ for all $b > 1$. In our example, if $v = 4$ then $m = 10$ and we verify on a computer that

$$A^{10} = \begin{bmatrix}0.33&0.23&0.17&0.27\\0.33&0.23&0.17&0.27\\0.33&0.23&0.17&0.27\\0.33&0.23&0.17&0.27\end{bmatrix}$$

A.1.2 Constructing Jointly Satisfiable Spaced Motifs

Here we describe how to procedurally generate c spaced motifs of length k such that the existence of a optimal solution $\mathbf{x}^*$ with length L with non-zero probability under a transition matrix A generated by the procedure in Appendix A.1.1 can be verified by construction. If we simply sampled motifs completely at random from $\mathcal{V}^k$ we cannot be sure that a solution attaining a global optimal value of 1 is actually feasible under the DMP constraint.

First we require that $L \geq c \times k$. Next to define the motifs, we draw a single sequence of length $c \times k$ from the DMP defined by A (the first element can be chosen arbitrarily). Then we chunk the sequence into c segments of length k, which defines the motif elements $\mathbf{m}^{(i)}$. This ensures that any

motif elements immediately next to each other are feasible, and ensures that one motif can transition to the next if they are placed side by side.

Next we draw random offset vectors $\mathbf{s}^{(i)}$. The intuition here is we want to ensure that an optimal solution can be constructed by placing the spaced motifs end-to-end. If we fix $c \times k$ positions to satisfy the motifs, there are $L - c \times k$ "slack" positions that we evenly distribute (in expectation) between the spaces between the elements of each motif. We set the first element of every spacing vector $s_1^{(i)}$ to 0, then set the remaining elements to the partial sums of a random draw from a discrete simplex as follows:

$$\mathbf{w}^{(i)} \sim \mathcal{U}\left(\{\mathbf{w} \in \mathbb{R}^{k-1} \mid \sum w_i = 1\}\right). \tag{12}$$

$$s_{j+1}^{(i)} = s_j^{(i)} + 1 + \lfloor w_j^{(i)} \times (L - c \times k)//c \rfloor. \tag{13}$$

Finally, recall that in Appendix A.1.1 we ensured that self-transitions $x \to x$ always have non-zero probability. This fact allows us to construct a feasible solution that attains the optimal value by filling in the spaces in the motifs with the previous motif elements.

As a concrete example, suppose $L = 8$, $c = 2$, and $k = 2$ (hence $s_{\max} = 3$) and we draw the following set of spaced motifs:

$$[0 \quad 3 \quad 1 \quad 2] \to \begin{bmatrix} 0 & 3 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} \mathbf{m}^{(1)} \\ \mathbf{m}^{(2)} \end{bmatrix}, \tag{14}$$

$$\begin{bmatrix} \mathbf{s}^{(1)} \\ \mathbf{s}^{(2)} \end{bmatrix} = \begin{bmatrix} 0 & 3 \\ 0 & 3 \end{bmatrix}. \tag{15}$$

$$\tag{16}$$

An optimal solution can then be constructed as follows:

$$\mathbf{x}^* = [0 \quad 0 \quad 0 \quad 3 \quad 1 \quad 1 \quad 1 \quad 2]$$

Note that this solution is only used to *verify* that the problem can be solved. In practice solutions found by optimizers like a genetic algorithm will look different. Additionally if $L \gg c \times k$ then the spaced motifs can often be feasibly interleaved without clashes.

A.1.3 Defining The Initial Solution

Optimizer performance is generally quite sensitive to the choice of initial solution. In our experiments we fixed the initial solution to a single sequence of length L drawn from the DMP.

A.2 Derivations

Derivation of the MargE Objective To derive a training objective that fulfills SIC, we follow Hu et al. (2024) and propose an objective that takes the following general form:

$$\arg\min_{\theta} \left[\mathbb{KL}(\tilde{\pi}_\theta \| \pi^*) + \lambda \mathbb{KL}(\pi_{\text{Ref}} \| \tilde{\pi}_\theta)\right] \tag{17}$$

where π^* is the target reward distribution and $\tilde{\pi}_\theta$ is the length-normalized version of π_θ.

Since past work has indicated that preference-tuned models often spuriously favor longer generations, we follow Malladi (2024); Meng et al. (2024) by instead using length-normalized likelihoods. Additionally, it is standard in past literature (Ziegler et al., 2019; Korbak et al., 2022a) to model π^* as a Boltzmann function of the reward. That is,

$$\pi^*(y \mid x) = \frac{1}{Z(x)} \exp(r(x, y))$$

where $Z(x)$ is the partition function. Although this partition function is not guaranteed to converge due to the observation that LLMs often place non-zero probability mass on infinite-length sequences (Du et al., 2023; Welleck et al., 2020), we make the assumption for now that this effect is negligible. This results in a formulation of the optimal policy that is Bradley-Terry with respect to the rewards. *I.e.*,

$$\log \pi^*(y_1|x) - \log \pi^*(y_2|x) = r(x, y_1) - r(x, y_2)$$

Then, the first term of Equation 17 can be expanded as:

$$\arg\min_\theta \mathbb{KL}(\tilde{\pi}_\theta \| \pi^*) = \arg\min_\theta \mathop{\mathbb{E}}_{\substack{x\sim\mathbb{D}_x,\\ y\sim\tilde{\pi}_\theta(\cdot|x)}} \log\left(\frac{\tilde{\pi}_\theta(y|x)}{\pi^*(y|x)}\right)$$

Since we cannot directly sample from $\tilde{\pi}_\theta$, we approximate it by sampling from the un-normalized policy:

$$\begin{aligned}
&\approx \arg\min_\theta \mathop{\mathbb{E}}_{\substack{x\sim\mathbb{D}_x,\\ y\sim\pi_\theta(\cdot|x)}} \left[\frac{\log \pi_\theta(y|x)}{|y|} - r(x,y) + \log Z(x)\right] \\
&= \arg\min_\theta \mathop{\mathbb{E}}_{\substack{x\sim\mathbb{D}_x,\\ y\sim\pi_\theta(\cdot|x)}} \left[\frac{\log \pi_\theta(y|x)}{|y|} - r(x,y)\right]
\end{aligned}$$

Rather than re-sampling from π_θ after every step of training, we approximate this step using importance sampling:

$$\begin{aligned}
\mathbb{KL}(\pi_\theta \| \pi^*) &\approx \sum_{\substack{x\sim\mathbb{D}_x,\\ y\sim\pi_\theta(\cdot|x)}} \frac{\pi_\theta(y|x)}{\pi_{\text{Ref}}(y|x)} \pi_{\text{Ref}}(y|x) \left[\frac{\log \pi_\theta(y|x)}{|y|} - r(x,y)\right] \\
&\approx \mathop{\mathbb{E}}_{\substack{x\sim\mathbb{D}_x,\\ y\sim\pi_{\text{Ref}}(\cdot|x)}} \frac{\pi_\theta(y|x)}{\pi_{\text{Ref}}(y|x)} \left[\frac{\log \pi_\theta(y|x)}{|y|} - r(x,y)\right] \qquad (18)
\end{aligned}$$

For the second term of Equation 17, we remove terms not depending on θ, resulting in the standard token-averaged cross-entropy loss:

$$\min_\theta \mathbb{KL}(\pi_{\text{Ref}} \| \tilde{\pi}_\theta) \equiv \min_\theta \mathop{\mathbb{E}}_{\substack{x\sim\mathbb{D}_x,\\ y\sim\pi_{\text{Ref}}(\cdot|x)}} -\frac{\log \pi_\theta(y|x)}{|y|} \qquad (19)$$

Plugging Equations 18 and 19 back into 17, we obtain

$$\mathcal{L}_{\text{MargE}}(\pi_\theta, \pi_{\text{Ref}}; \mathbb{D}_x) = \mathop{\mathbb{E}}_{\substack{x\sim\mathbb{D}_x,\\ y\sim\pi_{\text{Ref}}(\cdot|x)}} \left[\frac{\pi_\theta(y|x)}{\pi_{\text{Ref}}(y|x)} \left(\frac{\log \pi_\theta(y|x)}{|y|} - r(x,y)\right) - \lambda \frac{\log \pi_\theta(y|x)}{|y|}\right].$$

Since importance sampling often leads to high variance of the gradient estimates in practice (Owen, 2013), we instead use the self-normalized version of this objective (Eq. 32).

Lemma 1. *Given a Bradley-Terry model $\pi(y|x) = \frac{1}{Z(x)}\exp(r(x,y))$ with partition function $Z(x)$, if reward function $r(x,y) = f(y) - f(x)$ for some real-valued scoring function $f(x) : \mathcal{V}^* \to \mathbb{R}$, then $\pi(y|x) = \pi(y|z)$ for any pair $x, z \in \text{dom}(f)$.*

Proof. Since $Z(x)$ is the normalizing constant for $\pi(y|x)$, we can write

$$\begin{aligned}
Z(x) &= \sum_{y'\in\mathcal{V}^*} \exp(r(x,y')) \qquad (20) \\
&= \sum_{y'\in\mathcal{V}^*} \exp(f(y') - f(x)) \qquad (21) \\
&= \frac{1}{\exp(f(x))} \sum_{y'\in\mathcal{V}^*} \exp(f(y')). \qquad (22)
\end{aligned}$$

It follows that

$$\pi(y|x) = \frac{\exp(f(x))}{\sum_{y' \in \mathcal{V}^*} \exp(f(y'))} \exp(r(x,y)) \tag{23}$$

$$= \frac{\exp(f(x) + f(y) - f(x))}{\sum_{y' \in \mathcal{V}^*} \exp(f(y'))} \tag{24}$$

$$= \frac{\exp(f(y))}{\sum_{y' \in \mathcal{V}^*} \exp(f(y'))} \tag{25}$$

$$= \frac{\exp(f(y) + f(z) - f(z))}{\sum_{y' \in \mathcal{V}^*} \exp(f(y'))} \tag{26}$$

$$= \frac{\exp(f(z)) \exp(r(z,y))}{\sum_{y' \in \mathcal{V}^*} \exp(f(y'))} \tag{27}$$

$$= \frac{1}{Z(y)} \exp(r(z,y)) \tag{28}$$

$$= \pi(y|z) \tag{29}$$

□

A.3 Algorithms

Algorithm 1: LLOME, an approach for bilevel optimization of highly constrained sequence optimization problems with LLMs. We use $n_0 = 10$, $k = 2000$, and $T = 10$.

Input: Scoring function f; pretrained LLM π_{θ_0} parameterized by initial weights θ_0; initial seed sequence $\mathbf{x}_0 \in \mathcal{F}$; k number of test function evaluations per round; T number of LLOME rounds.

$S \leftarrow \{(\mathbf{x}_0, f(\mathbf{x}_0))\}$;
$\mathcal{X}_0 \leftarrow \text{GENETICALGORITHM}(\mathbf{x}_0, n_0)$; /* Seed with n_0 rounds of evolution. */
$S_0 \leftarrow \{(\mathbf{x}, f(\mathbf{x})) \mid \mathbf{x} \in \mathcal{X}_0\}$; /* Score the initial candidates. */
$S \leftarrow S \cup S_0$;
$i \leftarrow 0$;
while $i < T$; /* Outer Loop */
do
  $\mathcal{D}_i \leftarrow \text{DATASETFORMATTING}(S_i)$;
  $\theta_{i+1} \leftarrow \text{TRAIN}(\theta_i, \mathcal{D}_i)$;
  $\mathcal{X}_{i+1} \leftarrow \text{ITERATIVEREFINEMENT}(\pi_{\theta_{i+1}}, S_i)$; /* Inner Loop */
  $\mathcal{X}_{i+1} \leftarrow \text{FILTER}(\mathcal{X}_{i+1}, k)$; /* Filter $\mathcal{X}_{i+1}$ down to k samples. */
  $S_{i+1} \leftarrow \{(\mathbf{x}, f(\mathbf{x})) \mid \mathbf{x} \in \mathcal{X}_{i+1}\}$; /* Oracle labeling. */
  $S \leftarrow S \cup S_{i+1}$;
  $i \leftarrow i + 1$;
end

Output: $\arg\max_{(\mathbf{x}, f(\mathbf{x})) \in S} f(\mathbf{x})$

A.3.1 Dataset Formatting

To format our data, we adapt PropEn (Tagasovska et al., 2024), a technique for matching pairs of examples to implicitly guide the model towards generating sequences that are close by the input but still improve a particular property. Since PropEn was originally designed for pairs of data (x, y), we also adapt PropEn to create triples of data (x, y_w, y_l) for DPO training. In short, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ of input sequences x_i and their scores y_i, PropEn creates the following paired dataset:

$$\mathcal{D}_{\text{PropEn}} = \{(x_i, x_j) \mid (x_i, y_i), (x_j, y_j) \in \mathcal{D}, d(x_i, x_j) \leq \Delta_x, f(x_j) - f(x_i) \in (0, \Delta_y]\} \tag{30}$$

for thresholds Δ_x and Δ_y and distance function d. In all our experiments, we use the Hamming distance as d and modify the constraint $f(x') - f(x) \in (0, \Delta_y)$ to $f(x') > f(x)$ since we observed that

Algorithm 2: DATASETFORMATTING(S). An algorithm that adapts PropEn (Tagasovska et al., 2024) to create pairs or triples of data for LLM training. We use $k_n = 30$, $\Delta x = 0.25$, and the fractional Hamming distance as d.

Input: Dataset $S = \{(\mathbf{x}, y)\}$ of particles $\mathbf{x}$ and scores y; k_n number of nearest neighbors to find; distance function d; threshold Δx; dataset type $type$ = ("binary" | "triple")

if $type =$ *"binary"* **then**

$\mathcal{D} \leftarrow \left\{ (x_i, x_j) \middle| \begin{smallmatrix} (x_i,y_i),(x_j,y_j)\in S \\ x_j \in \text{KNEARESTNEIGHBORS}(x_i,k_n) \\ d(x_i,x_j)\leq\Delta_x \\ f(x_j)>f(x_i) \end{smallmatrix} \right\}$;

else

$\mathcal{D} \leftarrow \left\{ (x_i, x_j, x_k) \middle| \begin{smallmatrix} (x_i,y_i),(x_j,y_j),(x_k,y_k)\in S \\ x_j,x_k \in \text{KNEARESTNEIGHBORS}(x_i,k_n) \\ d(x_i,x_j)\leq\Delta_x, d(x_i,x_k)\leq\Delta_x \\ f(x_j)>f(x_i), f(x_i)\geq f(x_k) \end{smallmatrix} \right\}$

end

Output: $\mathcal{D}$

Algorithm 3: ITERATIVEREFINEMENT($\pi_\theta; S$). π_θ^t represents π_θ with temperature t scaling. We use $n_s = 200$, $n_i = 10$, $n_o = 10$, and $T = [0.6, 0.8, 1.0, 1.2, 1.4, 1.6]$.

Input: Pretrained LLM π_θ; dataset $S = \{(\mathbf{x}, y)\}$ of sequences $\mathbf{x}$ and scores y; n_s number of seed examples; n_i rounds of iteration per example; n_o outputs per iteration; T set of temperatures to sample with.

$S' \leftarrow \{\}$;

$\mathcal{X} \leftarrow \{\mathbf{x} \mid (\mathbf{x}, y) \in \text{TOPK}(S, n_s)\}$; `/* Obtain` n_s `seed examples from the top training examples by score. */`

for $\mathbf{x} \in \mathcal{X}$ **do**

  $\mathbf{x}_0 \leftarrow \mathbf{x}$;

  $i \leftarrow 0$;

  for $i < n_i$ **do**

    $\mathbf{x}_{i+1} \leftarrow \text{GREEDYDECODING}(\pi_\theta; \mathbf{x}_i)$;

    $S' \leftarrow S' \cup \{\mathbf{x}_{i+1}\}$;

    $i \leftarrow i + 1$;

  end

  for $t \in T$ **do**

    $i \leftarrow 0$;

    for $i < n_i$ **do**

      $\mathcal{X}_i \leftarrow \{\mathbf{x}_j \sim \pi_\theta^t(\cdot \mid \mathbf{x}_i)\}_{j=1}^{n_o}$; `/* Sample` n_o `sequences using temperature` t. `*/`

      $S' \leftarrow S' \cup \mathcal{X}_i$;

      $\mathbf{x}_{i+1} \leftarrow \arg\max_{\mathbf{x}' \in \mathcal{X}_i} \pi_\theta(\mathbf{x}' \mid \mathbf{x})$; `/* Select the highest-likelihood sample as the input for the next iteration. */`

      $i \leftarrow i + 1$;

    end

  end

end

$S' \leftarrow \text{DEDUPLICATE}(S')$;

Output: S'

this looser constraint was more effective in our experiments. We use $\mathcal{D}_{\text{PropEn}}$ for both LLOME-SFT and LLOME-MARGE.

To additionally adapt the PropEn dataset creation process for preference tuning (*e.g.* DPO), we create preference triples using the following constraints:

$$\mathcal{D}_{\text{PropEn-Triples}} = \left\{ (x_i, x_j, x_k) \middle| \begin{smallmatrix} (x_i,y_i),(x_j,y_j),(x_k,y_k)\in\mathcal{D} \\ d(x_i,x_j)\leq\Delta_x, d(x_i,x_k)\leq\Delta_x \\ f(x_j)>f(x_i), f(x_i)\geq f(x_k) \end{smallmatrix} \right\} \tag{31}$$

Algorithm 4: FILTER$(\mathcal{X}, k)$. An algorithm for filtering a dataset $\mathcal{X}$ of sequences down to only k sequences.

Input: LLM π_θ that generated the sequences; dataset $\mathcal{X} = \{(\mathbf{x}, y)\}$ of sequences $\mathbf{x}$ and scores y; likelihood threshold p_{min}; maximum proportion of infeasible sequences $p_{\text{max-infeas}}$; final output size k.

$\mathcal{X} \leftarrow \{((\mathbf{x}, y), \pi_\theta(\mathbf{x})) \mid (\mathbf{x}, y) \in \mathcal{X}\}$; `/* Compute likelihoods. */`

$\mathcal{X} \leftarrow \{(\mathbf{x}, y) \mid \pi_\theta(\mathbf{x}) > p_{\text{min}}, ((\mathbf{x}, y), \pi_\theta(\mathbf{x})) \in \mathcal{X}\}$;

$\mathcal{X}_{\text{feasible}} \leftarrow \{(\mathbf{x}, y) \mid y \neq -\infty, (\mathbf{x}, y) \in \mathcal{X}\}$;

$\mathcal{X}_{\text{infeasible}} \leftarrow \{(\mathbf{x}, y) \mid y = -\infty, (\mathbf{x}, y) \in \mathcal{X}\}$;

$n_{\text{feasible}} \leftarrow |\mathcal{X}_{\text{feasible}}|$;

$n_{\text{max-infeas.}} \leftarrow n_{\text{feasible}} \times \frac{p_{\text{max-infeas}}}{1 - p_{\text{max-infeas}}}$;

$\mathcal{X} \leftarrow \mathcal{X}_{\text{feasible}} \cup \text{SAMPLE}(\mathcal{X}_{\text{infeasible}}, n_{\text{max-infeas.}})$; `/* Downsample infeasible examples. */`

$\mathcal{X} \leftarrow \text{SAMPLE}(\mathcal{X}, \min(k, |\mathcal{X}|))$; `/* Subsample k examples. */`

Output: $\mathcal{X}$

In preference tuning terms, the x_i is the prompt or input sequence, and x_j and x_k are y_w and y_l, respectively. We use $\mathcal{D}_{\text{PropEn-Triples}}$ in LLOME-DPO. We formalize this algorithm in Alg. 2.

For all training algorithms, we format the input as "`<inc> [3, 1, ⋯, 5]`" where "`<inc>`" is a control code meant to indicate to the model that it should edit the sequence to increase the score. Outputs are formatted similarly, but without the control code.

A.3.2 Iterative Refinement

Our iterative refinement is formalized in Alg. 3. Loosely, we select the best n_s training examples from the last round of the LLOME outer loop, and provide them as seed inputs to the LLM to refine. For each seed input, we use 10 rounds of iterative generation where the best (highest-likelihood) generation from the previous round is provided as input to the next round. We repeat this process with both greedy decoding and sampling at various temperatures. In the case of greedy decoding, only one generation is obtained per iteration. When we sample, we sample $n_o = 10$ outputs at once.

Since LLM outputs tend to become less diverse with more rounds of training, we also implement automatic temperature adjustments after each round of LLOME. The default temperature range is $T = [0.6, 0.8, 1.0, 1.2, 1.4, 1.6]$, but if the average Hamming distance of generations from the last LLOME round was < 0.075, then we instead use $T + 0.6$. For average Hamming distance between 0.075 and 0.1, we use $T + 0.4$. For averages between 0.1 and 0.1, we use $T + 0.2$.

A.3.3 Genetic Algorithm

In Algorithms 5, 6, and 7, we provide pseudo-code for our genetic algorithm baseline, which we implement in pure PyTorch (Paszke et al., 2019), using the `torch.optim` API.

The GA baseline has only four hyperparameters, the total number of particles n, the survival quantile $\alpha \in (2/n, 1)$, the mutation probability p_m, and the recombination probability p_r. Generally speaking for best performance one should use the largest n possible, and tune α (which determines the greediness of the optimizer), p_m, and p_r.

Our genetic algorithm uses mutation probability $p_m = 0.005$, $n = 1000$ particles per iteration, survival quantile $\alpha = 0.1$, and recombination probability $p_r = 0.0882$.

A.4 LLM Training Details

We train every model for 1 epoch with PyTorch DDP on two A100 GPUs, using training loops implemented with the Huggingface `datasets`, `transformers`, and `trl` libraries. We conducted hyperparameter searches for each LLM training method, using the validation loss from the dataset of the first iteration of LLOME to select the best hyperparameters. We also check whether the generated outputs are parsable and conform to the correct format (*i.e.*, a list of the correct length with values in the correct range). If the hyperparameter set-up with the lowest validation loss does not output sequences with the correct format $> 90\%$ of the time, then we select the set-up with the next best

Algorithm 5: Genetic algorithm pseudo-code

Input: initial solution $\widehat{\mathbf{x}^*}$, $\widehat{f^*}$, mutation probability p_m, recombination probability p_r, survival quantile α, # particles n
$\mathcal{X}_{\text{pop}} \leftarrow \texttt{mutate}(\{\widehat{\mathbf{x}^*}\}, p_m, n)$
for $t = 1, \ldots, T$ **do**
  $\mathbf{v} \leftarrow f(\mathcal{X}_{\text{pop}})$
  if $\max v_i > \widehat{f^*}$ **then**
    $\widehat{\mathbf{x}^*} \leftarrow \text{argmax } v_i$
    $\widehat{f^*} \leftarrow \max v_i$
  end
  $\tau \leftarrow \texttt{quantile}(\mathbf{v}, 1 - \alpha)$
  $\mathcal{X}_{\text{top}} \leftarrow \{\mathbf{x} \in \mathcal{X}_{\text{pop}} \mid f(\mathbf{x}) \geq \tau\}$
  $n' \leftarrow n - |\mathcal{X}_{\text{top}}|$
  $\mathcal{X}_{\text{pop}} \leftarrow \mathcal{X}_{\text{top}} \cup \texttt{recombine}(\mathcal{X}_{\text{top}}, p_r, n')$
  $\mathcal{X}_{\text{pop}} \leftarrow \texttt{mutate}(\mathcal{X}_{\text{pop}}, p_m, 1)$
end
Returns: Estimated maximizer $\widehat{\mathbf{x}^*}$, $\widehat{f^*}$

Algorithm 6: `mutate` function

Input: initial set $\mathcal{X}$, mutation probability p_m, number of mutants n.
$\mathcal{X}' = \emptyset$
for $\mathbf{x} \in \mathcal{X}$ **do**
  for $i = 1, \ldots, n$ **do**
    $\texttt{mask} = \texttt{rand\_like}(\mathbf{x}) < p_m$
    $\texttt{sub} = \texttt{randint}(0, v - 1, \texttt{len}(\mathbf{x}))$
    $\mathbf{x}' = \texttt{where}(\texttt{mask}, \texttt{sub}, \mathbf{x})$
    $\mathcal{X}' = \mathcal{X}' \cup \{\mathbf{x}'\}$
  end
end
Returns: $\mathcal{X}'$

Algorithm 7: `recombine` function

Input: initial set $\mathcal{X}$, recombine probability p_r, number of recombinations n.
$\mathcal{X}' = \emptyset$
$\mathcal{P}^{(1)} = \texttt{draw\_w\_replacement}(\mathcal{X}, n)$
$\mathcal{P}^{(2)} = \texttt{draw\_w\_replacement}(\mathcal{X}, n)$
for $i = 1, \ldots, n$ **do**
  $\mathbf{x}^{(1)} = \mathcal{P}_i^{(1)}$
  $\mathbf{x}^{(2)} = \mathcal{P}_i^{(2)}$
  $\texttt{mask} = \texttt{rand\_like}(\mathbf{x}^{(1)}) < p_r$
  $\mathbf{x}' = \texttt{where}(\texttt{mask}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)})$
  $\mathcal{X}' = \mathcal{X}' \cup \{\mathbf{x}'\}$
end
Returns: $\mathcal{X}'$

validation loss that meets these constraints. Notably, these format checks were the most important for DPO. Many DPO-trained models achieve low validation loss despite generating sequences with incorrect format. We use the best hyperparameters tuned from the validation dataset created during the first iteration of LLOME but do not repeat hyperparameter tuning in future iterations. All search ranges and final hyperparameter values (in **bold**) are listed below.

SFT We train the SFT models with the AdamW optimizer, with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\lambda = 0.01$, and $\epsilon = 1 \times 10^{-8}$. We also search the following hyperparameter ranges:

- Learning rate $\in \{1 \times 10^{-7}, \mathbf{1 \times 10^{-6}}, 1 \times 10^{-5}\}$
- Batch size $\in \{16, 32, 64, \mathbf{128}\}$

DPO We train the DPO models with the RMSprop optimizer, with $\alpha = 0.99$, $\lambda = \mu = 0$, and $\epsilon = 1 \times 10^{-8}$. Due to computational constraints, we train with `bf16`. We also search the following hyperparameter ranges:

- Learning rate $\in \{\mathbf{1 \times 10^{-7}}, 1 \times 10^{-6}\}$
- Batch size $\in \{\mathbf{64}, 128\}$
- $\beta \in \{0.1, 0.2, 0.4, 0.8\}$

MargE We train the MargE models with the AdamW optimizer, with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\lambda = 0.01$, and $\epsilon = 1 \times 10^{-8}$. We also search the following hyperparameter ranges:

- Learning rate $\in \{1 \times 10^{-7}, \mathbf{1 \times 10^{-6}}\}$
- Batch size $\in \{\mathbf{64}, 128\}$
- $\lambda \in \{0.2, 0.4, 0.8, 1.0, \mathbf{10.0}\}$

We trained REINFORCE with the same best hyperparameters as MargE. For both MargE and REINFORCE, we applied self-normalization to the importance weights. That is, if $\mathcal{B}(x, y)$ is the batch of examples that a particular example (x, y) belongs to, then the self-normalized MargE and REINFORCE objectives are as follows:

$$\tilde{\mathcal{L}}_{\text{MargE}}(\pi_\theta, \pi_{\text{Ref}}; \mathbb{D}_x) = \mathop{\mathbb{E}}_{\substack{x \sim \mathbb{D}_x, \\ y \sim \pi_{\text{Ref}}(\cdot|x)}} \left[\tilde{w}(x, y) \left(\frac{\log \pi_\theta(y|x)}{|y|} - r(x, y) \right) - \lambda \frac{\log \pi_\theta(y|x)}{|y|} \right] \tag{32}$$

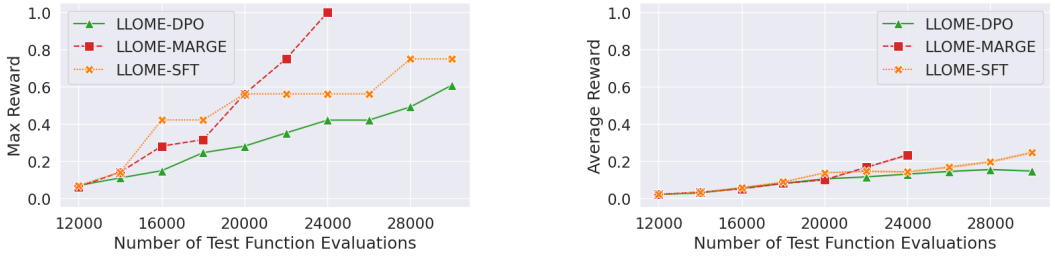
$$\tilde{\mathcal{L}}_{\text{REINFORCE}}(\pi_\theta, \pi_{\text{Ref}}; \mathbb{D}_x) = \mathop{\mathbb{E}}_{\substack{x \sim \mathbb{D}_x, \\ y \sim \pi_{\text{Ref}}(\cdot|x)}} \left[\tilde{w}(x, y) \left(-r(x, y) \log \pi_\theta(y|x) \right) - \lambda \frac{\log \pi_\theta(y|x)}{|y|} \right] \tag{33}$$

where

$$w(x, y) = \pi_\theta(y|x) / \pi_{\text{Ref}}(y|x), \tag{34}$$

$$\tilde{w}(x, y) = \frac{w(x, y)}{\sum_{(x', y') \in \mathcal{B}(x, y)} w(x', y')}. \tag{35}$$

A.5 Additional Results



(a) Maximum reward achieved.

(b) Average reward.

Figure 11: Average and maximum reward achieved by methods that rely upon editing the original sequence. The shaded regions in (2b) represent the 95% confidence interval. The lines for LLOME-MARGE end early because LLOME-MARGE discovers the optimal solution early.

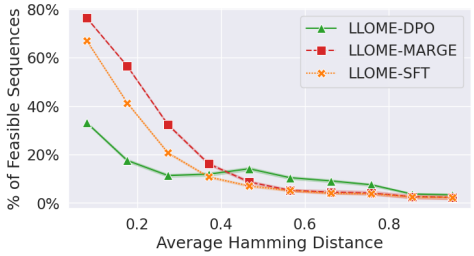


Figure 12: The percentage of feasible LLM-generated sequences, binned by the average Hamming distance (normalized by length) between the input and output. Shaded regions indicate the 95% confidence interval.

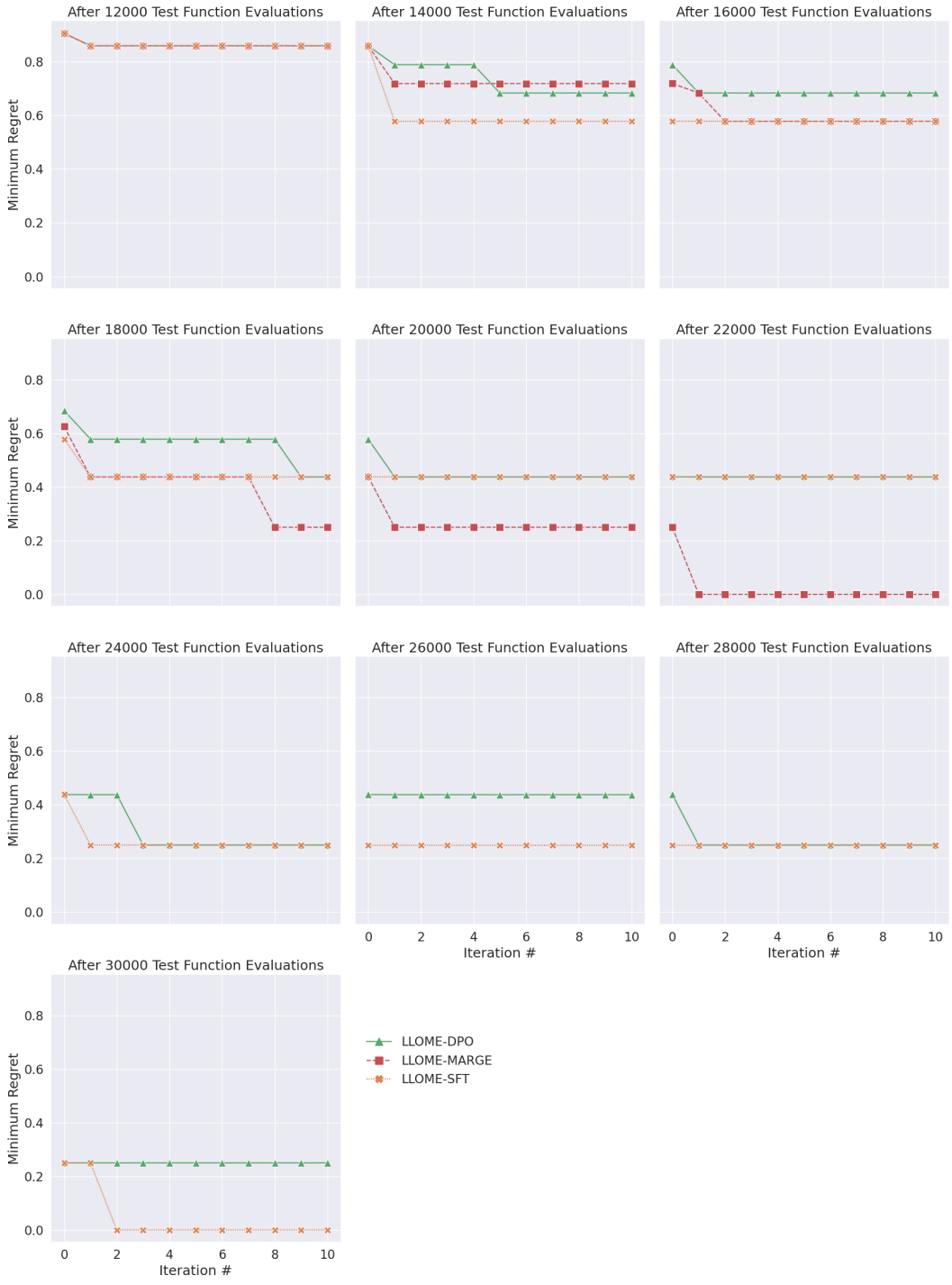


Figure 13: Minimum regret of sequences generated during the LLM inner loop, at each iteration of the iterative refinement process. The titles reflect the number of oracle labels that each LLM has been trained on. These plots account for all generations sampled from the LLM inner loop, and not just the samples selected via likelihood selection, as in Alg. 3.