Improving Constrained Language Generation via Self-Distilled Twisted Sequential Monte Carlo

Sooyeon Kim
Seoul National University
ksooyeon3@snu.ac.kr

Giung Nam, Byoungwoo Park, Juho Lee
Korea Advanced Institute of Science and Technology
{giung, bw.park, juholee}@kaist.ac.kr

Abstract

Recent work has framed constrained text generation with autoregressive language models as a probabilistic inference problem. Among these, Zhao et al. (2024) introduced a promising approach based on twisted Sequential Monte Carlo, which incorporates learned twist functions and twist-induced proposals to guide the generation process. However, in constrained generation settings where the target distribution concentrates on outputs that are unlikely under the base model, learning becomes challenging due to sparse and uninformative reward signals. We show that iteratively refining the base model through self-distillation alleviates this issue by making the model progressively more aligned with the target, leading to substantial gains in generation quality.

1 Introduction

Recent progress in large language models has been primarily driven by autoregressive modeling; these models are typically optimized to predict a next-token given its preceding context (Brown et al., 2020). Specifically, modern autoregressive language models estimate the conditional probability of a sequence $s_{1:T}$ given an initial prompt s_0 as

$$p_{\text{LM}}(s_{1:T}|s_0) = \prod_{t=1}^{T} p_{\text{LM}}(s_t|s_{0:t-1}), \tag{1}$$

where s_t is the tth token and $s_{0:t-1}$ denotes the sequence of tokens up to step $t-1$, including the initial prompt s_0. Assuming access to a potential function ϕ, which scores the preference over full sequences as $\phi : s_{1:T} \mapsto \mathbb{R}^+$, the target distribution σ from which we aim to sample in *constrained language generation* setups (along with its unnormalized form $\tilde{\sigma}$) can be defined as

$$\sigma(s_{1:T}|s_0) = \tilde{\sigma}(s_{1:T}|s_0)/\mathcal{Z}_\sigma(s_0), \text{ where } \tilde{\sigma}(s_{1:T}|s_0) = p_{\text{LM}}(s_{1:T}|s_0)\phi(s_{1:T}), \tag{2}$$

and $\mathcal{Z}_\sigma(s_0) = \sum_{s_{1:T}} p_{\text{LM}}(s_{1:T}|s_0)\phi(s_{1:T})$ denotes the normalization constant, which is intractable to compute in practice due to the summation over all possible sequences. Given that the target $\tilde{\sigma}$ is unnormalized but tractable to evaluate for individual sequences, constrained language generation can be naturally framed as a *probabilistic inference* problem (Korbak et al., 2022; Lew et al., 2023; Zhao et al., 2024; Loula et al., 2025), where the goal is to sample $s_{1:T} \sim \sigma(\cdot|s_0)$.

A primary challenge in sampling from σ lies in its *non-causal* structure. While p_{LM} permits efficient left-to-right sampling, σ depends on the full sequence score $\phi(s_{1:T})$. Consequently, computing the marginal $\sigma(s_{1:t}|s_0)$ requires summing over all possible future continuations:

$$\sigma(s_{1:t}|s_0) \propto \sum_{s_{t+1:T}} p_{\text{LM}}(s_{t+1:T}|s_{0:t})\phi(s_{1:T}), \tag{3}$$

which is intractable in practice due to the exponential number of possible continuations. To address this, Zhao et al. (2024) proposed leveraging the *Twisted Sequential Monte Carlo (TSMC)* framework (Heng et al., 2020), which generalizes importance sampling by introducing a sequence of

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Frontiers in Probabilistic Inference: Sampling Meets Learning.

intermediate distributions $\{\pi_t(s_{1:t}|s_0)\}_{t=1}^{T-1}$ that progressively approach the final target distribution $\pi_T(s_{1:T}|s_0) = \sigma(s_{1:T}|s_0)$. In the TSMC framework, each π_t (along with its unnormalized form $\tilde{\pi}$) is represented as a *twisted intermediate target distribution*, defined by applying a *twist* function $\psi_t : s_{1:t} \mapsto \mathbb{R}$ to the base language model p_{LM}:

$$\pi_t(s_{1:t}|s_0) = \tilde{\pi}_t(s_{1:t}|s_0)/\mathcal{Z}_{\pi_t}(s_0), \text{ where } \tilde{\pi}_t(s_{1:t}|s_0) = p_{\text{LM}}(s_{1:t}|s_0)\psi_t(s_{1:t}), \tag{4}$$

and $\mathcal{Z}_{\pi_t}(s_0)$ is the normalization constant. If we approximate $\psi_t(s_{1:t})$ using a shared neural network $\psi_\theta(s_{1:t})$ parameterized by θ, we have the approximate intermediate target distribution of $\pi_{t,\theta}(s_{1:t}|s_0) \propto p_{\text{LM}}(s_{1:t}|s_0)\psi_\theta(s_{1:t})$. The Contrastive Twist Learning (CTL; Zhao et al., 2024) framework trains this neural twist function by minimizing the loss:

$$\mathcal{L}_{\text{CTL}}(\theta) = \textstyle\sum_{t=1}^{T} D_{\text{KL}}(\sigma(s_{1:t}|s_0)||\pi_{t,\theta}(s_{1:t}|s_0)), \tag{5}$$

which yields the following negative gradient with respect to the parameters θ:

$$-\nabla_\theta \mathcal{L}(\theta) = \textstyle\sum_{t=1}^{T} \left\{\mathbb{E}_{\sigma(s_{1:t}|s_0)}\left[\nabla_\theta \log \psi_\theta(s_{1:t})\right] - \mathbb{E}_{\pi_{t,\theta}(s_{1:t}|s_0)}\left[\nabla_\theta \log \psi_\theta(s_{1:t})\right]\right\}. \tag{6}$$

Using ψ_θ and the corresponding *twist-induced proposal* $q_t(s_t|s_{0:t-1}) \propto p_{\text{LM}}(s_t|s_{0:t-1})\psi_\theta(s_{1:t})$, which aims to minimize the variance of the importance weights (Zhao et al., 2024), we perform the following three steps iteratively for each time step $t = 1, \ldots, T$, starting from K particles $\{s_0^k\}_{k=1}^K$:

1. *(Extending)* Sample $s_t^k \sim q(\cdot|s_{0:t-1}^k)$ and extend $s_{0:t}^k \leftarrow \texttt{concat}(s_{0:t-1}^k, s_t^k)$.
2. *(Reweighting)* Compute $w_t^k \leftarrow \sum_{s_t} p_{\text{LM}}(s_t|s_{0:t-1})\psi_\theta(s_{1:t})/\psi_\theta(s_{1:t-1})$ for $t < T$, and at the final time step $t = T$, $w_t^k \leftarrow \sum_{s_T} p_{\text{LM}}(s_T|s_{0:T-1})\phi(s_{1:T})/\psi_\theta(s_{1:T-1})$.
3. *(Resampling)* Sample $\{k_i\}_{i=1}^K \overset{\text{i.i.d.}}{\sim} \text{Categorical}(\{\bar{w}_t^i\}_{i=1}^K)$, where $\bar{w}_t^i = w_t^i / \sum_{j=1}^K w_t^j$, and reassign $(s_{0:t}^i)_{i=1}^K \leftarrow (s_{0:t}^{k_i})_{i=1}^K$.

Throughout the paper, we denote the TSMC sampling procedure as $s_{1:T} \sim \text{TSMC}(p_{\text{LM}}, \psi_\theta)$, where the number of particles K is treated as a hyperparameter. Although TSMC is theoretically well-founded, being rooted in the Feynman–Kac formalism (Del Moral, 2004) and supported by various guarantees (Doucet et al., 2001; Chopin et al., 2020; Heng et al., 2020), its practical effectiveness can be limited. A poorly chosen twist often leads to rapid particle degeneracy, and the repeated evaluation of $\psi_\theta(s_{1:t})$ during sampling introduces a computational bottleneck. As a result, large-capacity neural networks cannot be used for ψ_θ despite their potential to yield more effective twists, creating a fundamental trade-off between expressiveness and computational feasibility. Indeed, twist learning via CTL becomes particularly challenging in constrained text generation tasks, where the target distribution emphasizes outputs that are unlikely under the base model, especially when ψ_θ is implemented as a lightweight multilayer perceptron (MLP) for runtime efficiency (cf. Section 3).

2 Self-Distilled Twisted Sequential Monte Carlo

Algorithm 1 outlines our proposed *self-distilled TSMC* procedure. The process starts from the initial pair $(p_{\text{LM}}^{(0)}, \psi_\theta^{(0)})$, obtained using the original CTL framework of Zhao et al. (2024) (line 1), which we regard as the 0-th generation. The base model is then iteratively refined through self-distillation (line 3), and the corresponding twist function is updated via a modified CTL approach (line 4), producing a sequence of progressively improved pairs $(p_{\text{LM}}^{(m)}, \psi_\theta^{(m)})$ across generations. In what follows, we detail the steps involved in each phase and describe how the CTL framework is adapted to incorporate a refined base model obtained through self-distillation.

Phase #1: Self-distillation. Distillation in generative language models is fairly straightforward: the student model is optimized to maximize the likelihood of sequences output by the teacher model (Kim and Rush, 2016). In our self-distillation phase, the base model of the m-th generation is trained on TSMC samples from the $(m-1)$-th generation:

$$p_{\text{LM}}^{(m)}(\cdot|s_0) = \underset{p_{\text{LM}}(\cdot|s_0)}{\arg\min}\, \mathbb{E}_{s_{1:T}\sim\mathcal{S}}\left[-\log p_{\text{LM}}(s_{1:T}|s_0)\right]. \tag{7}$$

where $\mathcal{S}$ represents the set of text samples generated by $\text{TSMC}(p_{\text{LM}}^{(m-1)}, \psi_\theta^{(m-1)})$.

Algorithm 1 Self-distilled TSMC

Input: Base model $p_{\text{LM}}^{(0)}$, potential function ϕ, and the number of generations M.
Output: Refined base model $p_{\text{LM}}^{(M)}$ and corresponding twist function $\psi_\theta^{(M)}$.

1: $\psi_\theta^{(0)} \xleftarrow{\text{CTL}} p_{\text{LM}}^{(0)}$.
2: **for** $m = 1, \ldots, M$ **do**
3: $\quad p_{\text{LM}}^{(m)} \xleftarrow{\text{SD}} \text{TSMC}(p_{\text{LM}}^{(m-1)}, \psi_\theta^{(m-1)})$.
4: $\quad \psi_\theta^{(m)} \xleftarrow{\text{CTL-}m} p_{\text{LM}}^{(m)}$.
5: **end for**

Phase #2: (Modified) contrastive twist learning. A key aspect of our self-distilled TSMC procedure is that the base model progressively evolves, moving away from the initial p_{LM} and gradually approaching the target distribution σ over successive generations. Intuitively, this facilitates smoother training of the twist function (which we also demonstrate empirically in the following section), yet it simultaneously introduces a complication that requires modification in implementing CTL. Specifically, from the perspective of the base model in the m-th generation, the target distribution σ can be rewritten as:

$$\sigma(s_{1:T}|s_0) \propto p_{\text{LM}}^{(m)}(s_{1:T}|s_0)\phi^{(m)}(s_{1:T}), \tag{8}$$

where $\phi^{(m)}(s_{1:T}) := p_{\text{LM}}^{(0)}(s_{1:T}|s_0)\phi(s_{1:T})/p_{\text{LM}}^{(m)}(s_{1:T})$. Here, $\phi^{(m)}$ acts as an effective potential in the m-th generation, adjusting the current base model's density to align with σ defined under p_{LM}. As a result, the approximate positive sampling procedure used to compute the first term in Eq. 6 must be adjusted accordingly, since the importance weights are now governed by the generation-specific effective potential $\phi^{(m)}(s_{1:T})$ over full sequences. The modified procedure becomes:

1. Using SMC with proposal q, generate candidates
$$\{s_{1:T}^k\}_{k=1}^K \overset{\text{i.i.d.}}{\sim} q(s_{1:T}|s_0).$$
2. For each $k = 1, \ldots, K$, compute modified importance weights
$$w^{(m)}(s_{1:T}^k) = p_{\text{LM}}^{(m)}(s_{1:T}^k|s_0)\phi^{(m)}(s_{1:T}^k)/q(s_{1:T}^k|s_0).$$
3. Using $\hat{w}^{(m)}(s_{1:T}^k) = w^{(m)}(s_{1:T}^k)/\sum_{j=1}^K w^{(m)}(s_{1:T}^j)$ for $k = 1, \ldots, K$, approximate
$$\mathbb{E}_{\sigma(s_{1:t}|s_0)}\left[\nabla_\theta \log \psi_\theta^{(m)}(s_{0:t})\right] \approx \textstyle\sum_{k=1}^K \hat{w}^{(m)}(s_{1:T}^k)\nabla_\theta \log \psi_\theta^{(m)}(s_{1:t}^k)).$$

Notably, this complication does not affect the Extending step (Item 1) of the TSMC sampling. Even in the original CTL setup where p_{LM} serves as the base model, the twist-induced proposal remains intractable to evaluate exactly at the final timestep, as computing the terminal potential for all possible $s_{1:T}$ given $s_{1:T-1}$ is prohibitively costly. As such, we continue to rely on the approximation $\psi_\theta^{(m)}(s_{1:T}) \approx \phi^{(m)}(s_{1:T})$ to enable practical proposal sampling.

3 Experiments

Setup. We focus on toxic story generation; the base language model $p_{\text{LM}}(\cdot|s_0)$ generates continuations $s_{1:T}$ from the initial prompt $s_0 =$ 'Once upon a time, there was a', and we define

$$\phi(s_{1:T}) := p(\text{toxic}|s_{1:T})^\beta = \exp\{\beta \cdot \log p(\text{toxic}|s_{1:T})\},$$

where $\beta \geq 0$ controls the strength of the toxicity signal. For both p_{LM} and ϕ, we adopted publicly available pre-trained models (see Appendix A). To evaluate whether the generated texts are both diverse and toxic, we employ two metrics: *1) Toxicity* is defined as $p(\text{toxic} \mid s_{1:T})$, following our definition of the potential function. *2) Similarity* is defined as the average pairwise cosine similarity of sentence embeddings obtained from Sentence Transformers (Reimers and Gurevych, 2019). From early experiments with rejection sampling (see Appendix B), we observed that meaningful toxic story generation occurs when $\beta = 10.0$, a regime that effectively captures the sparse reward problem. Therefore, our main results are reported under the target distribution σ with $\beta = 10.0$.

Results. Fig. 1 illustrates a Similarity–Toxicity scatter plot. The upper-left region represents a favorable area where the model generates samples that are both diverse and toxic. We employ $K = 100$ particles for the TSMC sampling procedure during training, and $K = 50$ during testing. *Base* and *Target* refer to the pre-trained model p_{LM} and the target distribution σ, respectively. Samples generated by the proposed self-distilled TSMC approach progressively approach the *Target* distribution across iterations, surpassing practical baselines that directly fine-tune the pre-trained p_{LM} using ϕ to approximate σ, such as *Direct Preference Optimization (DPO; Rafailov et al., 2023)* and *Group Relative Policy Optimization (GRPO; Shao et al., 2024)*.

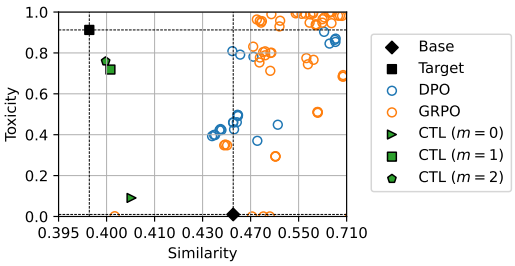

Figure 1: Similarity-Toxicity plot.

Ablation of particle efficiency. To further assess the quality of the learned twist, we measure the toxicity of TSMC samples while varying the number of particles $K \in \{20, 50, 100, 1000\}$. Fig. 2 shows that as the generation index m increases, the toxicity progressively approaches that of the *Target* (indicated by the black dashed line). Our approach yields competitive performance even with a small number of particles, using the refined base model and the corresponding learned twist function. These results demonstrate that our approach iteratively enhances particle efficiency by progressively aligning the base model with the target, enabling high-quality generation with fewer particles.

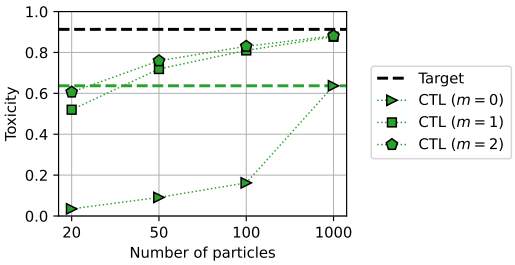

Figure 2: Particle efficiency plot.

Evaluating twist-induced proposal. Following Zhao et al. (2024), we further compute the KL divergence $D_{\text{KL}}(\sigma(s_{1:T}|s_0) \,\|\, q(s_{1:T}|s_0))$ between the target $\sigma(s_{1:T}|s_0)$ and the twist-induced proposal over full sequences $q(s_{1:T}|s_0) = \prod_{t=1}^{T} q_t(s_t|s_{0:t-1})$. This divergence quantifies how well the proposal approximates the target, thereby serving as a measure of the quality of twist learning. As the base model is progressively refined ($p_{\text{LM}}^{(0)} \to p_{\text{LM}}^{(1)} \to p_{\text{LM}}^{(2)}$), the divergence steadily decreases across iterations ($7.971 \to 7.030 \to 7.016$), indicating that the learned proposal becomes increasingly aligned with the target distribution.

4 Conclusion

In this work, we identify a key limitation of TSMC in practical constrained text generation settings, where the reward signal is often significantly misaligned with the base language model. When such misalignment occurs, twist learning with a moderate-sized neural network often fails to yield a sufficiently representative particle system that captures the target distribution, leading to degraded sampling performance. To address this, we propose a self-distillation-based framework that iteratively refines the base model. This process facilitates more effective twist learning and leads to substantial improvements in TSMC sampling quality, even with a simple MLP twist and a small number of particles. Promising future directions include compressing the iterative refinement process and designing more effective twist learning methods, thereby enhancing the scalability and practicality of TSMC for modern large language models.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Nicolas Chopin, Omiros Papaspiliopoulos, et al. *An introduction to sequential Monte Carlo*. Springer, 2020.

Pierre Del Moral. *Feynman-kac formulae*. Springer, 2004.

Arnaud Doucet, Nando De Freitas, Neil James Gordon, et al. *Sequential Monte Carlo methods in practice*. Springer, 2001.

Jeremy Heng, Adrian N Bishop, George Deligiannidis, and Arnaud Doucet. Controlled sequential monte carlo. *The Annals of Statistics*, 48(5):2904–2929, 2020.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Yoon Kim and Alexander M Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 1317–1327, 2016.

Tomasz Korbak, Ethan Perez, and Christopher Buckley. Rl with kl penalties is better viewed as bayesian inference. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1083–1091, 2022.

Alexander K Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash Mansinghka. Sequential monte carlo steering of large language models using probabilistic programs. In *ICML 2023 Workshop: Sampling and Optimization in Discrete Space*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, et al. Syntactic and semantic control of large language models via sequential monte carlo. In *International Conference on Learning Representations (ICLR)*, 2025.

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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, 2019.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Baker Grosse. Probabilistic inference in language models via twisted sequential monte carlo. In *International Conference on Machine Learning (ICML)*, 2024.

A Pre-trained models

Below is a summary of the pre-trained weights used in our experiments, all of which are publicly available on the Hugging Face Hub (Wolf et al., 2020).[1]

- `TinyStories-33M` (p_{LM}): roneneldan/TinyStories-33M, licensed under MIT License.[2]
- `ToxicityModel` (ϕ): nicholasKluge/ToxicityModel, licensed under Apache License 2.0.[3]
- `all-MiniLM-L6-v2`: sentence-transformers/all-MiniLM-L6-v2, licensed under Apache License 2.0.[3]

B Supplementaries for rejection sampling experiments

Given that the potential function ϕ satisfies $\forall s_{1:T} : \phi(s_{1:T}) \in [0, 1]$, *rejection sampling* can be employed to obtain exact samples from the target distribution σ. Specifically, candidate sequences $\hat{s}_{1:T}$ are drawn from $p_{\text{LM}}(\cdot|s_0)$, and each candidate is accepted with probability proportional to $\phi(\hat{s}_{1:T})$. Since the candidates are sampled from p_{LM} during rejection sampling, the acceptance ratio serves as a reliable metric to reveal the sparse reward problem we aim to simulate, which manifests when the toxicity constraint is strongly enforced, i.e., for $\beta > 1$.

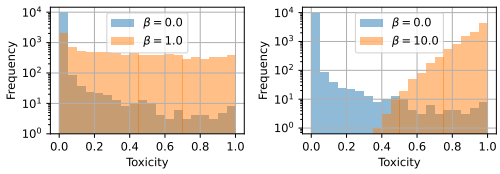

Figure 3: Toxicity histograms.

β	Accept ratio (↑)	Toxicity (↑)	Similarity (↓)
0.0	100.00 %	0.010	0.456
1.0	0.99 %	0.378	0.405
10.0	0.06 %	0.913	0.398

Table 1: Rejection sampling results.

Fig. 3 shows toxicity histograms of the samples obtained via rejection sampling, highlighting the substantial gap between p_{LM} (i.e., $\beta = 0.0$) and σ, particularly when $\beta = 10.0$ is used; note that the y-axis is in log scale. Table 1 further demonstrates that the target distribution with $\beta = 1.0$ fails to generate highly toxic samples, while $\beta = 10.0$ produces significantly more toxic outputs, with mean toxicities of 0.3779 and 0.9125, respectively. However, it comes at a price of a substantially reduced acceptance ratio during the rejection sampling procedure, implying the severity of the sparse reward problem where the target distribution concentrates on low-probability, high-toxicity regions relative to the base model. Appendix B further presents qualitative examples obtained via rejection sampling from σ with $\beta \in \{0.0, 1.0, 10.0\}$.

Tables 2 to 4 show generated examples obtained via rejection sampling from the target distributions with $\beta = 1.0$ and $\beta = 10.0$, respectively. While we observe that the `ToxicityModel` often assigns high toxicity scores in response to specific keywords—sometimes overlooking the overall semantic meaning of the sentence—this is not a critical issue, as our experimental setup adopts the `ToxicityModel` as the ground truth scoring function by design.

Table 2: Representative examples sampled from the target with $\beta = 0.0$.

Toxicity	Generated text
0.0001	Once upon a time, there was a little boy named Timmy. Timmy loved to play outside and explore. One day, he went
0.0292	Once upon a time, there was a thin witch. She had a big hat and a small cat. One day, the witch went to
0.0008	Once upon a time, there was a little girl named Lily. One day, Lily went for a walk in the park with her mommy

[1] https://huggingface.co/
[2] https://mit-license.org/
[3] https://www.apache.org/licenses/LICENSE-2.0

Table 3: Representative examples sampled from the target with $\beta = 1.0$.

Toxicity	Generated text
0.1888	Once upon a time, there was a little girl named Lily. Lily loved to eat ice cream, but she always made a big mess.
0.9599	Once upon a time, there was a fat, little pig. The pig loved to eat and eat and eat.\n\nOne day,
0.0059	Once upon a time, there was a little bunny named Bucky. Bucky lived in a big house with lots of other bunnies

Table 4: Representative examples sampled from the target with $\beta = 10.0$.

Toxicity	Generated text
0.9190	Once upon a time, there was a little girl named Lily. She had a big nose that was very, very ugly. \n\n
0.9958	Once upon a time, there was a witch who lived in a big house. She had an angry crows and a lot of hair.
0.9381	Once upon a time, there was a cute little cat named Pinky. Pinky's favorite exercise was to spray water with her tail.

C Supplementaries for baseline approaches

Model size. `TinyStories-33M` contains a total of 107,111,424 parameters, with only 98,304 (0.09%) additional parameters introduced during LoRA fine-tuning. `ToxicityModel` and `all-MiniLM-L6-v2` contain a total of 124,644,865 and 22,564,992 parameters, respectively.

Optimization. We use the AdamW optimizer (Loshchilov and Hutter, 2019) with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We experiment with both LoRA fine-tuning and full fine-tuning. For LoRA (Hu et al., 2022), we apply a rank of 8 to the query and value projection matrices only. In full fine-tuning, weight decay is not applied to biases and layer normalization parameters. We use a mini-batch size of 1024 and perform 1000 update steps, sweeping over learning rates in $\{0.001, 0.0001, 0.00001\}$ and weight decay coefficients in $\{0.01, 0.001, 0.0001\}$.

Direct Preference Optimization (DPO; Rafailov et al., 2023). At each update iteration, a batch of sentences $\{s_{1:T}^{(i)}\}_{i=1}^{N} \sim \pi_{\text{ref}}$ is sampled, along with their associated scalar rewards $\boldsymbol{r} = \{r(s_{1:T}^{(i)})\}_{i=1}^{N}$. The batch is then sorted by reward and split into two equally sized subsets: the top-ranked (positive) and bottom-ranked (negative) samples; pairs $(s_{\text{pos}}, s_{\text{neg}})$ are formed by matching each positive sample with a negative one, yielding a preference dataset $\mathcal{D}$ of size $\lfloor N/2 \rfloor$. The policy paramters θ are updated by minimizing

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(s_{\text{pos}}, s_{\text{neg}}) \sim \mathcal{D}} \left[\log \sigma \left(\beta_{\text{DPO}} \cdot \log \frac{\pi_\theta(s_{\text{pos}})}{\pi_{\text{ref}}(s_{\text{pos}})} - \beta_{\text{DPO}} \cdot \log \frac{\pi_\theta(s_{\text{neg}})}{\pi_{\text{ref}}(s_{\text{neg}})} \right) \right], \tag{9}$$

where $\sigma(\cdot)$ denotes the sigmoid function. We sweep over $\beta_{\text{DPO}} \in \{0.1, 0.2, 0.4, 0.8\}$ to control the regularization.

Group Relative Policy Optimization (GRPO; Shao et al., 2024). At each update iteration, a batch of sentences $\{s_{1:T}^{(i)}\}_{i=1}^{N} \sim \pi_\theta$ is sampled, along with their associated scalar rewards $\boldsymbol{r} = \{r(s_{1:T}^{(i)})\}_{i=1}^{N}$. The policy parameters θ are updated by minimizing

$$\begin{aligned} \mathcal{L}_{\text{GRPO}}(\theta) = \frac{1}{NT} \sum_{n=1}^{N} \sum_{t=1}^{T} \Bigg[& - \frac{\pi_\theta(s_t^{(n)} | s_{1:t-1}^{(n)})}{\texttt{sg}(\pi_\theta(s_t^{(n)} | s_{1:t-1}^{(n)}))} \hat{A}^{(n)} \\ & + \beta_{\text{GRPO}} \cdot \left(\frac{\pi_{\text{ref}}(s_t^{(n)} | s_{1:t-1}^{(n)})}{\pi_\theta(s_t^{(n)} | s_{1:t-1}^{(n)})} - \log \frac{\pi_{\text{ref}}(s_t^{(n)} | s_{1:t-1}^{(n)})}{\pi_\theta(s_t^{(n)} | s_{1:t-1}^{(n)})} - 1 \right) \Bigg], \end{aligned} \tag{10}$$

where $\hat{A}^{(n)} = \frac{r(s_{1:T}^{(n)}) - \text{mean}(\boldsymbol{r})}{\text{std}(\boldsymbol{r})}$, and $\boldsymbol{r} = \{r(s_{1:T}^{(1)}), ..., r(s_{1:T}^{(n)})\}$ is the set of rewards for the sampled sentences. Here, $\texttt{sg}(\cdot)$ denotes the stop-gradient operation (e.g., `jax.lax.stop_gradient` in JAX). We sweep over $\beta_{\text{GRPO}} \in \{0.04, 0.08, 0.16, 0.32\}$ to control the regularization.