Appendix for "Preference-grounded Token-level Guidance for Language Model Fine-tuning"

Table of Contents

A Additional Experimental Results

A.1 Tabular Results

Table 3: Examples of the generated discrete input-agnostic text-prompt and their classification accuracy on the corresponding test set.

SST-2 Prompt	SST-2 Accuracy	AG News Prompt	AG News Accuracy
guys filmmaker filmmaker rated Grade	94.18	newsIntroduction Comments Tags Search	85.78
MovieMovieFilm rated Grade	94.18	newsTopic Blog Support Category	85.55
Rated CinemaScoreReporting Grade	94.01	news RecentRecentPhotosIntroduction	84.53
employment theater rated Oscars Grade	93.96	news Recent Brief LatestExample	84.51
scene filmmaking rated comedian Grade	93.85	newsVirtualBlogBlogNet	84.33

Table 4: Detailed results on CNN/DM summarization under T5-base LM for Section 4.2. We bold the best result of each metric. Baseline results are directly cited from RL4LMs [58]. "Env. Reward" denotes the environmental reward in RL4LMs. The "ROUGE-L" here refers to "Rouge-LSum" in RL4LMs and in the Hugging Face interface, which is discussed in details in Appendix B.2. In Section 4.2, we plot the results of our method with the *average* aggregation, which is the best variant in Table 2. We report the mean and standard deviation of our method over three random seeds.

Algorithm	Env. Reward	ROUGE-1	ROUGE-2	ROUGE-L	Meteor
Lead-3		40.1	17.5	36.3	33.3
Supervised		41.1	17.7	34.3	30.9
PPO	Rouge-1	41.0	18.2	34.9	27.6
	Rouge-Avg	39.6	17.6	33.8	27.0
	Meteor	40.8	17.8	34.2	30.1
NLPO	Rouge-1	40.4	18.0	34.4	27.5
	Rouge-Avg	40.4	17.7	34.4	27.4
	Meteor	40.5	18.0	34.3	29.2
Supervised + PPO	Rouge-1	41.7	18.9	35.8	27.8
	Rouge-Avg	42.5	19.4	36.3	29.6
	Meteor	42.6	19.4	36.1	31.6
Supervised + NLPO	Rouge-1	42.1	19.3	36.1	28.7
	Rouge-Avg	42.4	19.3	36.3	29.5
	Meteor	42.9	19.4	36.1	31.9
Ours (AVG)		**43.09** (0.06)	**20.17** (0.04)	**39.99** (0.07)	**35.23** (0.06)
Ours (SUM)		42.86 (0.08)	19.92 (0.08)	39.76 (0.11)	34.74 (0.37)
Ours (MIN)		42.92 (0.14)	20.01 (0.02)	39.84 (0.08)	34.88 (0.13)
Ours (MAX)		42.38 (0.17)	19.49 (0.02)	39.34 (0.09)	34.13 (0.32)

Table 5: Scores on each ROUGE metric for our method using sequence-level and token-level preference-based guidance in the summarization tasks in Section 4.3 **(a)**. "Seq." denotes our method with sequence-level preference-based guidance, and "Token" denotes our method with token-level preference-based guidance. The reported numbers are mean (standard deviation) over three random seeds. The row "Average" shows the average of the three ROUGE scores, *i.e.*, (ROUGE-1 + ROUGE-2 + ROUGE-L) / 3.

	CNN/DM		XSum		CNN/DM (T5-base LM)	
	Seq.	Token	Seq.	Token	Seq.	Token
ROUGE-1	40.20 (0.07)	40.94 (0.02)	32.56 (0.08)	33.62 (0.03)	42.10 (0.15)	43.09 (0.06)
ROUGE-2	17.80 (0.08)	18.78 (0.03)	9.98 (0.04)	11.17 (0.02)	19.23 (0.11)	20.17 (0.04)
ROUGE-L	37.08 (0.06)	38.17 (0.03)	25.11 (0.07)	26.33 (0.05)	38.09 (0.14)	39.99 (0.07)
Average	31.69	32.63	22.55	23.71	33.14	34.42

Table 6: Scores on each ROUGE metric for our method with and without the reward-function retraining scheme in the summarization tasks in Section 4.3 **(b)**. "Without Retrain" denotes our method without reward-function retraining, and "With Retrain" denotes our method with reward-function retraining. The reported numbers are mean (standard deviation) over three random seeds. The row "Average" shows the average of the three ROUGE scores, *i.e.*, (ROUGE-1 + ROUGE-2 + ROUGE-L) / 3.

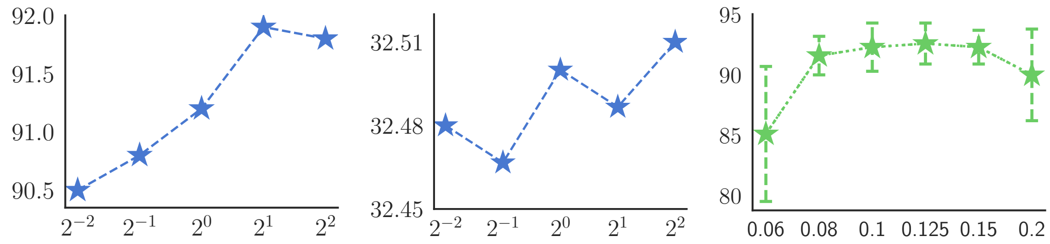
	CNN/DM		XSum		CNN/DM (T5-base LM)	
	Without Retrain	With Retrain	Without Retrain	With Retrain	Without Retrain	With Retrain
ROUGE-1	40.83 (0.10)	40.94 (0.02)	33.45 (0.11)	33.62 (0.03)	42.98 (0.08)	43.09 (0.06)
ROUGE-2	18.70 (0.07)	18.78 (0.03)	11.07 (0.06)	11.17 (0.02)	20.09 (0.06)	20.17 (0.04)
ROUGE-L	38.07 (0.09)	38.17 (0.03)	26.23 (0.10)	26.33 (0.05)	39.87 (0.08)	39.99 (0.07)
Average	32.53	32.63	23.58	23.71	34.31	34.42

Table 7: Scores on each ROUGE metric for the summarization task on CNN/DM in Section 4.3 **(c)**, where we vary the number of sequences used to learn the token-level guidance. The reported numbers are mean (standard deviation) over three random seeds. The row "Average" shows the average of the three ROUGE scores, *i.e.*, (ROUGE-1 + ROUGE-2 + ROUGE-L) / 3.

	Number of Sequences				
	2	3	5	7	9
ROUGE-1	40.80 (0.06)	40.94 (0.02)	40.87 (0.09)	40.86 (0.08)	40.95 (0.01)
ROUGE-2	18.70 (0.04)	18.78 (0.03)	18.71 (0.02)	18.74 (0.06)	18.78 (0.01)
ROUGE-L	38.05 (0.03)	38.17 (0.03)	38.09 (0.07)	38.08 (0.08)	38.18 (0.02)
Average	32.52	32.63	32.56	32.56	32.64

Table 8: Scores on each ROUGE metric for the summarization task on XSum in Section 4.3 **(c)**, where we vary the number of sequences used to learn the token-level guidance. The reported numbers are mean (standard deviation) over three random seeds. The row "Average" shows the average of the three ROUGE scores, *i.e.*, (ROUGE-1 + ROUGE-2 + ROUGE-L) / 3.

	Number of Sequences				
	2	3	5	7	9
ROUGE-1	33.54 (0.06)	33.62 (0.03)	33.56 (0.08)	33.56 (0.02)	33.63 (0.02)
ROUGE-2	11.12 (0.04)	11.17 (0.02)	11.12 (0.05)	11.19 (0.05)	11.20 (0.03)
ROUGE-L	26.26 (0.06)	26.33 (0.05)	26.28 (0.06)	26.34 (0.06)	26.36 (0.03)
Average	23.64	23.71	23.65	23.70	23.73



(a) Temperature β (SST-2) (b) Temperature β (CNN/DM) (c) Balancing Coeff. α (SST-2)

Figure 6: Line plots comparing the performance under different values of the hyperparameter β in Eq. (4) and α in Eq. (5). The plotted numbers are mean over three random seeds. Error bars show one standard deviation.

A.2 Further Ablation Study

In learning the preference-based *sequence-level* guidance in Section 4.3, the aggregation function $f(\cdot)$ in Section 2.1 is removed, since it is inapplicable and unnecessary to the sequence-level reward function. For the minimalist LM training objectives Eqs. (5) and (6) in Section 2.2, we change them to the corresponding versions that use sequence-level guidance. Self-normalization in reward-weighted MLE Eq. (6) is removed, since it is again inapplicable and unnecessary to the sequence-level setting.

In this section, we continue the discussion in Section 4.3 by answering the following additional questions on our method.

(a): *Is our method robust to the hyperparameter(s): temperature β and balancing coefficient α?*

To study the choice of the temperature parameter β in the soft-maximum/minimum aggregation Eq. (4), we vary the value of β in the MIN variant in Tables 1 and 2 from $\beta = 2$. Furthermore, to study the balancing coefficient α in the REINFORCE-style LM-training approach Eq. (5), we vary the α parameter in the AVG variant in Table 1 from $\alpha = 2^{-3}$. Fig. 6 respectively shows the prompt results on the SST-2 dataset and the summarization results on the CNN/DM dataset. For summarization, we again plot the average ROUGE scores, with the breakdown scores of the three ROUGE metrics in Table 9 below.

Recall that the best baseline result on SST-2 in Table 1 is 90.5, and on CNN/DM in Table 2 is 31.3. We see that our method can achieve competitive results on a relatively wide range of the temperature β. A too-small value of β, such as 0.25 and 0.5, may incur a harder optimization problem and thus an inferior performance on both prompt and summarization tasks.

For the choice of the balancing coefficient α, we see that our method provides competitive results in a relatively wide range of $\alpha \in [0.08, 0.15]$, when compared to the best baseline result of 90.5 in Table 1. A too-small value of α may not prevent the REINFORCE-style method from pre-mature convergence. The resulting LM therefore may not sufficiently explore the sampling space or capture multiple good behavior-modes, resulting in an inferior and highly varying performance. A too-large value of α distracts the optimization of the LM, and again leads to a worse result.

Table 9: Scores on each ROUGE metric for the summarization task on CNN/DM, where we vary the temperature parameter β in the *soft-minimum* aggregation Eq. (4). The reported numbers are mean (standard deviation) over three random seeds. The row "Average" shows the average of the three ROUGE scores, *i.e.*, (ROUGE-1 + ROUGE-2 + ROUGE-L) / 3.

	$\beta = 2^{-2}$	$\beta = 2^{-1}$	$\beta = 2^{0}$	$\beta = 2^{1}$	$\beta = 2^{2}$
ROUGE-1	40.77 (0.11)	40.74 (0.09)	40.79 (0.11)	40.78 (0.06)	40.80 (0.01)
ROUGE-2	18.67 (0.06)	18.68 (0.05)	18.68 (0.09)	18.67 (0.03)	18.71 (0.04)
ROUGE-L	38.00 (0.10)	37.98 (0.08)	38.03 (0.12)	38.01 (0.04)	38.02 (0.01)
Average	32.48	32.47	32.50	32.49	32.51

(b): *How does our method perform in generating longer prompts compared with the baseline?*

To further validate the harm of the delayed-feedback issue to the related LM-training methods that learn under the sequence-level feedback, we compare our method with RLPrompt [57] on generating prompts with length increased from 5 to 10 and to 20 tokens, on the SST-2 dataset. Table 10 below shows the results.

Table 10: Test accuracy on the prompt task on the SST-2 dataset, for our method and RLPrompt on generating prompts with length 5, 10, and 20 tokens. We report the mean and standard deviation over three random seeds.

	RLPrompt	Ours (AVG)	Performance Gap
5 Tokens	90.5 (1.5)	92.6 (1.7)	2.1
10 Tokens	75.8 (7.6)	86.0 (2.9)	10.2
20 Tokens	65.2 (6.0)	80.9 (4.5)	15.7

We see that RLPrompt performs worse than our method on generating longer prompts. In particular, the performance gap increases as the prompt length (feedback delaying) increases. This comparison can further show the harm of the delayed-feedback issue in training text-generation LMs, and that our framework, in particular our preference-grounded token-level guidance for LM training, is a viable solution to it.

It is intrigued that the results of both methods deteriorate with the prompt length. After checking the generated prompts from our method, we find that longer prompts mostly contain many repeated tokens, as shown by the following example prompt of length 20

```
PerformanceExceptionMovieMovieMovieMovieMovieMovieMovieVideoVideoVideoVideo\
VideoVideoVideoImageVideoImageImage
```

which is separated into two lines at the location of "\" due to the page width. In this prompt example, the tokens `Movie` and `Video` are each consecutively repeated seven times, and the bi-gram `ImageVideo` is repeated two times. Such prompts with heavy repetitions may confuse the downstream classifier.[5] This aligns with our intuition that a clear and succinct instruction is preferable than a long but verbose one.

As a side note, in generating Table 10, we use the default hyperparameters for both our method and RLPrompt. It is possible that RLPrompt requires careful tuning for generating longer prompts, due to the delayed-feedback issue that we try to address. We leave a thorough tuning of RLPrompt on long-prompt generation as a future work.

(c): *Is the efficacy of our framework tied to the specific preference sources considered in Section 4?*

To investigate whether the performance of our framework is tied to the specific preference-sources considered in the experiment section (Section 4), inspired by RL4LMs [58], we simulate the sequence-level preference on the summarization task by using another two automatic metrics "Rouge-avg" and "Rouge-avg2", rather than the classical Meteor score [82] in Section 4. Table 11 below presents the ROUGE scores of our method under each of the three preference sources on the CNN/DM dataset under the T5-base LM. For a more thorough investigation, we provide the results for our method both with and without the guidance re-estimation scheme. The baseline results in Table 11 below come from the best baseline method in Table 4 of Appendix A.1.

Table 11: Results for our method on CNN/DM summarization under T5-base LM when using different automatic metrics to simulate the sequence-level preference. We provide the detailed ROUGE scores for our method both with and without guidance re-estimation. "Baseline" denotes the results of the best baseline method in Table 4 of Appendix A.1. The reported numbers are mean over three random seeds. The row "Average" shows the average of the three ROUGE scores, *i.e.*, (ROUGE-1 + ROUGE-2 + ROUGE-L) / 3.

	Baseline	With Guidance Re-estimation			Without Guidance Re-estimation		
		Rouge-avg	Rouge-avg2	Meteor	Rouge-avg	Rouge-avg2	Meteor
ROUGE-1	42.9	43.14	43.07	43.09	42.96	42.98	42.98
ROUGE-2	19.4	20.18	20.12	20.17	20.07	20.05	20.09
ROUGE-L	36.1	39.93	39.89	39.99	39.80	39.77	39.87
Average	32.8	34.42	34.36	34.42	34.28	34.27	34.31

[5] A detailed description of the prompt task is deferred to Appendix D.

Concretely, these two new automatic metrics "Rouge-avg" and "Rouge-avg2" are constructed as

$$\text{Rouge-avg} = 0.5 \times \text{ROUGE-1} + 0.5 \times \text{ROUGE-2} + 0.5 \times \text{ROUGE-L}\,,$$
$$\text{Rouge-avg2} = 0.5 \times \text{ROUGE-1} + 0.5 \times 2 \times \text{ROUGE-2} + 0.5 \times \text{ROUGE-L}\,,$$

where the "Rouge-avg" metric is exactly the same as that in the RL4LMs [58]. The "Rouge-avg2" metric is constructed by multiplying ROUGE-2 by 2 to make its numerical value similar to the others.

It is clear that changing the preference source from Meteor to these two alternative metrics does not significantly alter the performance of our method, especially when compared to the performance improvement of our method over the best baseline method in Table 4 of Appendix A.1. This set of comparisons confirms that the efficacy of our framework is generally not tied to a specific preference source. It could also further corroborate the effectiveness of our preference-grounding perspective on guiding the LM training.

B Additional Experiment Details

B.1 Prompt Generation

Implementation Details. To ensure a fair comparison, the implementation of our framework is based on the official codebase of RLPrompt available at https://github.com/mingkaid/rl-prompt, and the Hugging Face library [67]. We have already provided some implementation details in Section 4.1. Here we continue the discussion.

The LM π_θ is parametrized as a frozen distilGPT-2 model with parameter θ being one MLP-layer of size 2048 inserted right before the output head. The token-level reward function r_ϕ is implemented as a distilGPT-2 with a two-layer projection-MLP of sizes 2048 and 1 on top. The LM π_θ is trained by a maximum of 12000 steps with early stopping on the validation set. The reward training is reconducted every 1000 steps during the first 6000 steps of the LM training process and is (and almost always) early stopped. RoBERTa-large is used [8] as the pre-trained downstream LM π_{DLM}.

Datasets. We use the standard datasets provided in the RLPrompt codebase [57]. We test on three popular few-shot classification datasets in prior work [*e.g.*, 71, 72], *i.e.*, two sentiment binary-classification datasets SST-2 [73] and Yelp Polarity [74], and the topic four-way-classification dataset AG News [74]. In keeping with the standard few-shot setting [70], both the training and the validation sets have 16 examples per class. To mitigate the randomness in the few-shot setting, each dataset is subsampled into five few-shot training-validation sets, while the test set is standard. We train our models on each few-shot (sub-)dataset with three random seeds and evaluate three generated prompts in each case. For all three tested datasets, we report the average test accuracy and standard deviation across all evaluated prompts in all random seeds and all few-shot (sub-)datasets.

Hyperparameters. Apart from the hyperparameters discussed in the ablation study (Section 4.3 and Appendix A.2), most other hyperparameters as well as the training and evaluation procedures of our framework follow RLPrompt. Additionally, we list the important hyperparameters for training our reward model in Table 12, and important hyperparameters for training our LM in Table 13. The generated prompts have a fixed length of 5. The same hyperparameters are used in all tested datasets.

Baselines. For the baseline results in Table 1, we rerun the codebase of RLPrompt under the same random seeds and evaluation script as our method. Other baseline results are from the literature [57, 80]. We note that our reported RLPrompt results have some small discrepancies compared to the original paper's results. We have confirmed our reproduced results with RLPrompt's authors and with Table 2 of the recent TEMPERA paper [80].

Table 12: Hyperparameters for training our reward model in the prompt-generation task.

Hyperparameter	Value
Gradient clipping norm	5.0
Max train steps	10000
Steps per epoch	100
Number of epochs	100
Learning rate	5e-5
Batch size	64
Learning-rate decay	0.8
Learning-rate scheduler	`ReduceLROnPlateau`
Scheduler patience	2
Early-stop count	7
Optimizer	Adam [86]
Backbone	distilGPT-2

Table 13: Hyperparameters for training our LM in the prompt-generation task.

Hyperparameter	Value
Gradient clipping norm	5.0
Max train steps	12000
Steps per epoch	500
Number of epochs	24
Learning rate	5e-5
Batch size	32
Learning-rate decay	0.8
Learning-rate scheduler	`ReduceLROnPlateau`
Scheduler patience	2
Early-stop count	7
Optimizer	Adam
Backbone	distilGPT-2
Reward retrain period	1000 steps

B.2 Text Summarization

Implementation Details and Hyperparameters. The implementation of our framework is based on the Hugging Face library [67]. We have provided some implementation details in Section 4.2. The discussion is continued here.

Due to our limited computational resources, unless explicitly mentioned, we use the standard T5-small model [81] for the LM. Similar to the prompt tasks, the token-level reward function is implemented also as a T5-small model, with a two-layer projection-MLP on top with sizes 2048 and 1. The LM π_θ is trained for a standard 5 epochs. Apart from the hyperparameters discussed in the ablation study (Section 4.3 and Appendix A.2), most other hyperparameters as well as the training and evaluation procedure of our framework follow the standard setting of using a T5 model for text summarization on the Hugging Face library. Additionally, we list the important hyperparameters for training our reward model in Table 14, and important hyperparameters for training our LM in Table 15. The same hyperparameters are used in both the CNN/DailyMail and the XSum datasets.

We note that the ROUGE-L metric we report is technically the `rougeLsum` metric from the Hugging Face interface and in the RL4LMs' codebase [58]. This one matches the result scales in prior work especially on texts with newlines ("`\n`"), as reported in this GitHub issue.

Baselines. For the baseline methods' results in Table 2, we rerun the codebase of RL4LMs [58] with a T5-small model as our method. We have carefully tuned the (supervised+) PPO/NLPO in RL4LMs on several hyperparameters, such as `learning_rate`, `kl_div:coeff`, `kl_div:target_kl`, and so on. Furthermore, we ran these baseline methods on the same random seeds as our method and we provide error bars. Since we use the T5-small model and the same random seeds for both our method and the baselines, our reported results are therefore (more) fair comparisons.

Table 14: Hyperparameters for training our reward model in the text-summarization task.

Hyperparameter	Value
Gradient clipping norm	5.0
Number of epochs	1
Amount of training data	10% of training set
Learning rate	5e-5
Batch size	32
Optimizer	Adam
Backbone	T5-small

Table 15: Hyperparameters for training our LM in the text-summarization task.

Hyperparameter	Value
Gradient clipping norm	5.0
Number of epochs	5
Learning rate	5e-5
Batch size	32
Optimizer	AdamW [87]
Weight decay	0.0
Backbone	T5-small
Reward retrain period	0.5 epoch

C A Naïve Numeric Example for the *Average* Aggregation

This section provides a naïve numeric comparison that the *average* aggregation in Section 2.1 will not automatically favor longer sequences, while the classical *summation* will.

Suppose we have $K = 2$ sequences τ^1 and τ^2 for preference learning, respectively having length $T^1 = 5$ and $T^2 = 15$. For simplicity, assume that all tokens in τ^1 and τ^2 are the same and all have reward 1, *i.e.*, $r_\phi(s_t^k, a_t^k) = 1, \forall\, k, t$. The average sequence length C is then $C = (1/2) \times (5 + 15) = 10$. For the first sequence τ^1, the *average*-aggregated sequence-level evaluation $e_\phi^{\text{avg}}(\tau^1) = (10/5) \times \sum_{t=0}^{4} 1 = (10/5) \times 5 = 10$. And for the second sequence τ^2, $e_\phi^{\text{avg}}(\tau^2) = (10/15) \times \sum_{t=0}^{14} 1 = (10/15) \times 15 = 10$. Therefore, no sequence will be automatically preferred based only on the length.

By contrast, when using the classical *summation* as the aggregation function, τ^1 will be evaluated as $\sum_{t=0}^{4} 1 = 5$ while τ^2 will be evaluated as $\sum_{t=0}^{14} 1 = 15$. So, indeed, the longer sequence τ^2 will be automatically preferred.

D Details on the Prompt Generation Task

Task Description. In discrete text-prompt generation [*e.g.*, 9, 68], we input a discrete text-prompt $\boldsymbol{a}$ and an observation sequence o to a large pre-trained downstream LM $\pi_{\text{DLM}}(y_{\text{DLM}} \mid \boldsymbol{a}, o)$ to directly classify text o, without finetuning π_{DLM}. Here, y_{DLM} denotes the output of the large downstream LM π_{DLM} on the observation text o prompted by text $\boldsymbol{a}$. We follow the classical prompt setting [*e.g.*, 9, 69, 57] that solves the classification problem by an encoder-only downstream LM via token infilling. Classification is reduced to selecting tokens corresponding to some predefined class labels, known as verbalizers, such as "happy" for positive and "sad" for negative. The set of verbalizers is denoted as $\mathcal{C}$. As an example, to classify an observation text o by prompt $\boldsymbol{a}$ using an encoder-only downstream LM π_{DLM}, we input a template such as "`[o] [a] [MASK]`" to π_{DLM}, and select the most probable verbalizer token that fills into `[MASK]`.

Setting. In our input-agnostic setting, the generated prompt is independent of the observation text o. During inference time, only the learned prompts are used and the LM π_θ is discarded. The initial input x to π_θ is a dummy, and the target y is the class label in the mask position. We also adopt the few-shot setting, where the training set consists of a small number of samples per class. There is a larger standard test set for evaluation. With a fixed length T, the goal is to find discrete text-prompts $\boldsymbol{a} = (a_0, \ldots, a_{T-1})$ that have high test accuracy.

Source of the Preference. For learning the token-level guidance, we simulate the sequence-level preference by the recently proposed stepwise metric $\mathcal{R}_{\text{step}}$ in Deng et al. [57], *i.e.*, the higher the metric value the better prompt. This choice ensures a fair comparison with RLPrompt [57] and avoids a potential overfitting that we train and evaluate the LM on the same evaluation metric "accuracy".

Given a prompt $\boldsymbol{a}$, observation text o, and the true class label $y \in \mathcal{C}$, $\mathcal{R}_{\text{step}}$ measures the gap between the true class's probability and the highest probability in other classes. The gap is defined as

$$\text{Gap}_o(\boldsymbol{a}, y) = \pi_{\text{DLM}}(y \mid \boldsymbol{a}, o) - \max_{y' \in \mathcal{C}, y' \neq y} \pi_{\text{DLM}}(y' \mid \boldsymbol{a}, o),$$

where $\text{Gap}_o(\boldsymbol{a}, y) > 0$ when the prediction $y_{\text{DLM}}(\boldsymbol{a}, o)$ for text o is correct and < 0 otherwise. Define the indicator for correct prediction for o, Corr_o, as $\text{Corr}_o = \mathbf{1}\{\text{Gap}_o(\boldsymbol{a}, y) > 0\}$. The stepwise metric $\mathcal{R}_{\text{step}}$ for prompt $\boldsymbol{a}$ on observation text o and true class label y is define as

$$\mathcal{R}_{\text{step}}(y_{\text{DLM}}(\boldsymbol{a}, o), y) = \lambda_1^{1-\text{Corr}_o} \lambda_2^{\text{Corr}_o} \times \text{Gap}_o(\boldsymbol{a}, y),$$

where $\lambda_1 = 180$ and $\lambda_2 = 200$. In the experiments (Section 4 and Appendix A.2), we report test accuracy as in prior works.

LM Training. Since the prompt-generation task does not assume the availability of supervised data — the ground-truth prompts, the LM π_θ is trained by the REINFORCE-style update in Section 2.2 to automatically discover highly-accurate prompts.

E More Related Work

Prompt Generation. Prior works [*e.g.*, 6, 9, 77, 88] have shown that manual prompts can steer LMs to perform NLP tasks in the few/zero-shot setting. In general, prompts can be discrete, consisting of real token-strings; or can be continuous, where the prompts are entirely free word-embeddings that do not map to real tokens. Several works [*e.g.*, 89–93, 75] tune continuous soft prompts using gradient descent, which typically requires some expensive gradient information [72, 94]. In this work, we apply our framework to the task of input-agnostic discrete-prompt optimization due to its challenging setting, better human understandability of the learned prompts [95, 96], potential transferability across LMs [97, 70, 57], and more robustness in the low-data regime [90]. Recent works propose some new settings such as input-dependent prompt-tuning [80], which are potential further applications of our framework and are left for future work.

Text Summarization. Apart from using RL techniques discussed in Sections 3, prior works on text summarization [*e.g.*, 7, 98, 81, 99, 100] mainly focus on structural designs of the LMs and improvements on the source of the (pre-)training data, where the LMs are typically trained by vanilla MLE on the supervised data. In this paper, we apply our preferenced-grounded token-level guidance to this task by considering a weighted-MLE objective for LM training. The weights given by the learned reward function reflect some sequence-level preference among multiple candidate summaries. Our framework thus has the potential to learn and improve from lower-quality data, and generate summaries fulfilling more general evaluation metrics, such as human preference.

Align LMs with Preference. As our paper, prior works on aligning LMs with preference typically focus on adjusting the pretrained LMs, where preference comes from human feedback or from some automatic metrics. A classical strategy is to add external filters on top of the pretrained LMs to the generated text sequences or to the training sequences [*e.g.*, 17], where the LMs are trained using MLE on abundant supervised data. Another classical approach finetunes LMs using supervised learning (vanilla MLE) on some curated/improved datasets [18–20], or on massive highly-curated collections of tasks phrased as instructions for supervised finetuning the LMs [101, 102]. Apart from supervised learning, reinforcement learning techniques have also been applied to learn from human feedback (RLHF). Similar to the discussion in Section 3, these works typically learn a *sequence-level* classifier that predicts human (pairwise) preferences and during LM training add a general-purpose KL penalty that is less-targeted to the specific LM task and feedback (preference, metric scores, *etc.*) [*e.g.*, 21, 12, 22, 23], such as a token-level KL penalty towards the initial LM prior to training.

Alternatively, the divergence of the LMs from a target distribution can also be used as the finetuning objectives. This line of research [*e.g.*, 103–105] formalizes controlled text generation as a constraint satisfaction problem over LM's probability distribution, with an additional divergence-minimization objective that the LMs should have a minimal KL- or f-divergence from the original pretrained LM. These approaches, however, require explicit functional specification on the constraints or on the human preference, rather a more vague form of (binary) comparison between LM samples. For example, Go et al. [105] consider human preference as a probability distribution measuring how well the generated text-sequence satisfies the preference. Apart from this more demanding requirement, these approaches further require special methods to sample from the resulting LM.

To sum up, prior works on aligning LMs with preference mostly focus on an ungrounded *sequence-level* guidance, which can suffer from the delay-feedback issue in LM training, as discussed in Sections 1 and 3. By contrast, our preference-grounding perspective can provide a stable, data-driven, task-specific *token-level* guidance on LM training, and can potentially improve on vanilla MLE, especially when the quality of the supervised data cannot be guaranteed. We experimentally validate this intuition in Section 4 and Appendix A.2.

Apart from fine-tuning the pretrained LMs, Korbak et al. [106] recently apply preference alignment to the pre-training stage of the LMs. As with prior works, the sparse sequence-level evaluation (without KL penalty/stabilizer) is directly used, to learn a token-level value function, to condition the LM generation on, or for a reward-weighted regression objective. The pre-training stage in Korbak et al. [106] is a potential further application of our framework since we make no assumption on the zero-shot ability of the initialized LMs, as discussed in Sections 2.2 and 4.3.

We also notice that a recent robotics paper [107] proposes to *learn* a *weighted-sum* aggregation together with the per-step reward, to form the sequence-level evaluation in learning the reward function, based on pairwise preference over two trajectories of equal length. Compared with this recent work, our aggregation functions in Section 2.1 do not require additional modeling and training, and therefore can be more efficient and more stable for the reward-function learning. Additionally, we do not assume that trajectory lengths are equal, as this may be infeasible for LM tasks such as text summarization. Furthermore, our framework allows utilizing the preference among more than two trajectories, rather than the classical pairwise preference. In this particular aspect, our framework can be more general than this recent work of Kim et al. [107].

F A Discussion on Applying RL Methods to LM Tasks

F.1 LM Generation as a Token-level MDP

In most LM generation tasks, there is a dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^N$ of N supervised examples, where x is the input to the LM that can be a dummy, and $y \in \mathcal{Y}$ is the target text sequence. Viewing the LM as a token-level RL policy, LM generation can be formulated as a sequential decision-making problem, specified by the Markov Decision Process (MDP) $\mathcal{M} = (\mathbb{S}, \mathbb{A}, P, \mathcal{R}, \gamma, \mu_0)$ [108]. Specifically, $\mathbb{S}$ is the state space, where the state at timestep t, s_t, consists of the LM input x and the previously generated tokens $a_{<t} = (a_0, \ldots, a_{t-1}), t > 0$, *i.e.*, $s_0 = x$ and $\forall\, t > 0, s_t = (x, a_{<t})$. $\mathbb{A}$ is the action space, which is the vocabulary $\mathcal{V}$, and an action a_t at timestep $t \geq 0$ is a token from $\mathcal{V}$. $P(s_t, a_t) : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$ is the transition function that deterministically appends the newly sampled token to the end of the current state, *i.e.*, $\forall\, t \geq 0, s_{t+1} = (s_t, a_t) = (x, a_{\leq t})$. $\mathcal{R}(s_T, y) : \mathbb{S} \times \mathcal{Y} \to \mathbb{R}$ is the environmental reward (task-specific evaluation metric) that depends on the *final state* s_T of the LM-generation trajectory and the target sequence y. Here T is the ending time of the trajectory, *i.e.*, the length of the full generated text sequence; and $s_T = (x, a_0, \ldots, a_{T-1})$ is the final state of the generation trajectory consisting of the LM input x and the full generated text sequence $\boldsymbol{a} = (a_0, \ldots, a_{T-1})$. $\gamma \in [0, 1]$ is the discount factor. And $\mu_0(x) : \mathbb{S} \to [0, 1]$ is the distribution of the initial input x.

We denote the LM as $\pi_\theta(a_t \mid s_t)$, parametrized by θ. At each timestep t, $\pi_\theta(a_t \mid s_t)$ generates the next token a_t given the current state $s_t = (x, a_{<t})$. The ultimate goal of policy learning (LM training) is to maximize the expected environmental reward $\mathcal{R}$, which can be expressed as

$$\max_\theta \mathbb{E}_{(x,y)} \mathbb{E}_{\boldsymbol{a} \sim \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t)} \left[\mathcal{R}(s_T = (x, \boldsymbol{a}), y)\right] \,,$$

where (x, y) is drawn from the corresponding sampling distribution.

F.2 Delayed Feedback in RL-based LM Training

As discussed in Appendix F.1, the environmental reward $\mathcal{R}(s_T, y)$ is only defined on the full generated text sequence $\boldsymbol{a}$. The token-level MDP formulation of LM generation thus meets the problem of sparse reward-signal or the delayed feedback issue discussed in Section 1. Hereafter, we will use "sparse reward (signal)" and "delayed feedback" interchangeably depending on the context, as they are used synonymously in the RL literature.

Specifically, prior works [*e.g.*, 29, 57, 30] often manually interpolate the intermediate rewards by some non-informative values such as 0 or -1, *i.e.*, $\forall\, t \geq 0$

$$\mathcal{R}(s_t, y) = \begin{cases} 0 \text{ or } -1, & t < T \\ \mathcal{R}(s_T, y), & t = T \end{cases} \,. \tag{7}$$

It is clear that the reward signal is sparse. In other words, the feedback to intermediate actions/tokens is delayed until the full text-sequence has been generated.

We note that this sparse-reward/delayed-feedback problem will not be addressed by the standard actor-critic or Q-learning methods in RL. With only sparse reward-signals, it can be difficult to estimate the token-level value functions in these RL methods.

Specifically, the standard Monte Carlo estimate of the value functions is known to have high variance due to the large sampling space [108]. This problem is even severe in the LM tasks where there are exponentially many text sequences that can follow a partial sequence.

Further, as discussed in Guo et al. [29], the sparse-reward/delayed-feedback problem can also hurt the bootstrapping-style method for learning the value functions, since the standard value-function learning can suffer from "the unstable per-step bootstrapping-style training with sparse reward signals." This can subsequently harm the LM training since many actor-critic or Q-learning methods rely heavily on how accurately the learned value functions assess the quality of intermediate text sequences [108, 29].

F.3 Sparse Reward with KL Penalty

With the sparse-reward/delayed-feedback issue in Appendix F.2, prior works typically add a token-level KL-penalty to the sparse sequence-level environmental rewards Eq. (7). For simplicity, assume that in Eq. (7) the intermediate rewards are interpolated by 0. The KL-stabilized reward signal $R(s_t, a_t, y)$ is

$$R(s_t, a_t, y) = \begin{cases} -c \cdot \mathrm{KL}(\pi_\theta(a_t \mid s_t) \,||\, \pi_0(a_t \mid s_t)), & t < T-1 \\ \mathcal{R}(s_T, y) - c \cdot \mathrm{KL}(\pi_\theta(a_t \mid s_t) \,||\, \pi_0(a_t \mid s_t)), & t = T-1 \end{cases}, \tag{8}$$

where c is a hyper-parameter and π_0 is some prior distribution, such as the uniform distribution [29, 57], the initial LM prior to training [21, 58], the supervised-fine-tuned model [59, 60, 10, 12], or the base momentum model [61]. For a concrete example, see Line 224-235 of the popular trlx package's implementation.

With this KL-stabilized reward signal $R(s_t, a_t, y)$, the action-value function for the policy/LM π_θ is

$$\begin{aligned} Q(s_t, a_t, y) &= \mathbb{E}_{\{a_{t'}\}_{t'=t+1}^{T-1} \sim \pi_\theta} \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} R(s_{t'}, a_{t'}, y) \mid s_t, a_t \right] \\ &= \mathbb{E}_{\{a_{t'}\}_{t'=t+1}^{T-1} \sim \pi_\theta} \left[\gamma^{T-1-t} \mathcal{R}(s_T, y) - c \cdot \sum_{t'=t}^{T-1} \gamma^{t'-t} \mathrm{KL}(\pi_\theta(a_{t'} \mid s_{t'}) \,||\, \pi_0(a_{t'} \mid s_{t'})) \mid s_t, a_t \right] \end{aligned} \tag{9}$$

It is clear from Eq. (9) that the environmental reward $\mathcal{R}(s_T, y)$ is multiplied by a factor exponentially decayed with respect to the length of the remaining horizon $T - 1 - t$. Without the KL penalty, the action-value $Q(s_t, a_t, y)$ could be tiny when t is small, *i.e.*, at the beginning of the text-sequence generation. This could make it hard to accurately model and learn the action values, echoing the previously-stated harm of the sparse-reward/delayed-feedback problem mentioned by Guo et al. [29]

Recall that the standard actor-critic and Q-learning methods in RL use the action-value function $Q(s_t, a_t, y)$ as the token-level guidance (per-step critic) for policy/LM training. Due to the exponentially decaying factor γ^{T-1-t}, when the discount factor γ in Eq. (9) is not sufficiently large, this token-level guidance $Q(s_t, a_t, y)$ in RL-based LM training mainly reflects the (discounted) sum of future KL-penalty, rather than the actual goal of LM training — the environmental reward $\mathcal{R}(s_T, y)$. This phenomenon can be more evident at the beginning of the text-sequence generation, *i.e.*, when the length of the remaining horizon $T - 1 - t$ is long. On the other hand, learning the action-value function $Q(s_t, a_t, y)$ under a large discount factor γ is known to be challenging [108], since the highly varying (late) future can significantly affect the current action value $Q(s_t, a_t, y)$. The selection of the discount factor γ, therefore, becomes a tradeoff and a challenge. Note that $\mathcal{R}(s_T, y)$ here is generic and can represent automatic evaluation metrics or (human) preference, and that the beginning of text generation can affect all subsequent token selections. Intuitively, using Eq. (9) as the token-level guidance for policy/LM training can thus be less successful in the concrete LM task, especially when generating longer sequences, as we verified in Appendix A.2.

In the experiments (Section 4 and Appendix A.2), we compare our preference-grounding approach with RL-based baselines that estimate a standard value function similar to Eq. (9) from sparse environmental reward with KL penalty, such as the RLPrompt method [57] and the (supervised+) PPO/NLPO methods in RL4LMs [58]. We leave as future work the potential combination of our preference-grounded guidance with actor-critic and Q-learning methods in RL-based LM training.

G Further Discussion on the Guidance Re-estimation Scheme

As discussed in Section 2.2, in this paper, we deal with the most general setting where the LM training directly starts from a raw pre-trained LM, rather than an initial LM that has been fine-tuned via supervised learning on the desired dataset, such as in Stiennon et al. [10]. We also make no assumptions about the zero-shot ability of the raw pre-trained LM. We choose this setting because it is more general and naturally fits into the task of text-prompt generation, where supervised datasets of good prompts are not available and the initial LM cannot generate good prompts.

As discussed before, under this general setting, the LM π_θ can evolve from a less-preferred distribution to a highly-preferred one, over the training process. Since our reward function r_ϕ is trained by text sequences sampled from π_θ, there is a distribution shift between the sequences used to train r_ϕ during reward-function learning, and the sequences evaluated by r_ϕ during LM training, especially after π_θ has been sufficiently improved. To keep r_ϕ as accurate guidance for LM training, a natural idea is to refine r_ϕ periodically on the text generations from the latest LM, leading to our reward-function retraining scheme.

We emphasize that *the reward-function retraining scheme does not give our method an unfair advantage over the baseline methods*. In particular, RLPrompt [57] and RL4LMs' methods [58] retrain their value-functions in every optimization step, and thus, they query the environmental reward in every optimization step. Specifically, in Algorithm 1 of the RL4LMs paper, the penalized reward $\hat{R}_t$ is calculated in each optimization step, whose calculation requires the true environmental reward R (Eq. (1) of the RL4LMs paper). Besides, in the codebase of RLPrompt, this environmental interaction is implemented in this line, which is queried in every optimization step, as seen in this line. In the notion of Reinforcement Learning from Human Feedback (RLHF), this every-step interaction is similar to asking humans to score the LM generations in every training step, which can be infeasible. By contrast, in our paper, we reduce the frequency of these environmental interactions by retraining the guidance model only periodically and only during the first half of the LM-training process.

Though the motivation of this reward-function retraining scheme comes from model-based RL (Section 2.2), we notice that some prior RLHF works do implement similar ideas. For example, Page 2 of Ziegler et al. [21] mentions that "..., we continue to collect additional data and retrain our reward model as the policy improves (online data collection)." Page 2 of Stiennon et al. [10] mentions that "We can then gather more human data using samples from the resulting policy, and repeat the process." Page 5 of Menick et al. [23] and Page 20 of Bai et al. [22] also have similar discussions. Based on these, our reward-function retraining scheme is both well-motivated and practical, even with human rankings in RLHF.

H Potential Negative Societal Impacts

Since our framework can ground the sequence-level preference into token-level guidance for LM training and can be not tied to a specific preference source, it is possible that this framework may be used to train ill-intended LMs by grounding some malicious or unethical preferences. This potential negative impact may be mitigated by closer monitoring the datasets on which our framework operates.

I Limitations

Since our token-level guidance is learned by grounding sequence-level preference, a potential failure case of our framework will be when the preference orderings are very noisy. In this situation, the learned guidance may not be meaningful and hence could even deteriorate the subsequent utilization of it in LM training.

Even though we have shown in Section 4.3 that it can be beneficial to use more than two sequences to learn the token-level guidance, it can be practically challenging to obtain a high-quality ranking among many candidate text sequences, *e.g.*, when the number of sequences is more than seven.

Besides, the reward-function retraining scheme may incur some additional computational complexity, compared with training the reward function only once and fixing it throughout the LM-training process.

J Computational Resources

The experiments are conducted on NVIDIA GeForce RTX 3090 and NVIDIA A100 GPUs. Depending on the specific task and setting, several experiments could be run concurrently on a single GPU.