Progressive Content Refinement with Decaying Reward Joint LinUCB

Anonymous ACL submission

Abstract

Iterative refinement has significantly enhanced Large Language Model (LLM) performance; however, existing methods—ranging from feedback-based Self-Refine to traditional bandit approaches—often rely on static options or overlook the “saturation effect”. This neglect leads to over-exploitation, where the continuous use of identical prompts or arms results in diminishing rewards over time.

To address this challenge, we propose a novel contextual bandit algorithm that explicitly incorporates reward decay modeling. Utilizing an Expectation-Maximization (EM) algorithm, our method simultaneously estimates both arm-specific and decay parameters. Furthermore, by embedding prompts as arms, we facilitate the joint learning of arm values, distinguishing our approach from the traditional disjoint Linear Upper Confidence Bound (LinUCB) framework.

Experimental results on Sentiment Reversal and GSM8K benchmarks demonstrate that our method achieves significant performance gains over strong baselines. Finally, our ablation study confirms that the integration of reward decay modeling within the bandit framework is crucial for mitigating over-exploitation and optimizing the iterative refinement process.

1 Introduction

In recent years, Large Language Models (LLMs) have revolutionized the field of natural language processing, demonstrating remarkable performance across a wide array of tasks such as text generation, summarization, translation, and complex reasoning [Brown et al., 2020; Chung et al., 2024]. To further unlock the full potential of these models and obtain higher-quality, more reliable outputs, iterative refinement techniques have become increasingly crucial [Madaan et al., 2023; Hein et al., 2025]. These methods aim to systematically enhance LLM-generated content by applying successive improvements based on various feedback mechanisms.

While initial iterative refinement approaches like Self-Refine [Madaan et al., 2023] have shown promise by allowing LLMs to critique and improve their own outputs, they often rely on pre-defined or heuristically chosen refinement prompts. This static selection of prompts can lead to suboptimal performance, as the effectiveness of a refinement strategy can vary significantly across different contexts and over time. More sophisticated methods have begun to incorporate multi-armed bandit (MAB) frameworks. These frameworks treat different refinement strategies (e.g., prompt design strategies, contents to refine) as "arms" to be pulled [Ashizawa et al., 2025; Tang et al., 2024]. These bandit-based approaches effectively balance the exploration of novel refinement strategies with the exploitation of known effective ones, optimizing the overall refinement process.

However, a critical limitation in existing iterative refinement paradigms, particularly prevalent in bandit-based methods, is the implicit assumption of stationary reward distributions for each refinement strategy (arm). In reality, repeatedly applying the same refinement prompt or exploiting a single arm can lead to a "saturation effect." For instance, an LLM might quickly exhaust the utility of a specific grammatical correction prompt after a few iterations, or a mathematical reasoning prompt might become redundant once a certain level of accuracy is achieved. Continuing to apply such a prompt results in diminishing returns, or even negative impacts, akin to over-editing. Empirical evidence for this decay phenomenon is presented in Figure 4 of [Madaan et al., 2023]. This phenomenon, termed "reward decay" by prior work like Rotting Bandit [Levine et al., 2017], leads to inefficient over-exploitation of diminishingly effective strategies, hindering overall refinement efficiency. Traditional

bandit algorithms, designed for stationary environments, fail to adequately capture and adapt to this dynamic decay in reward.

While the concept of decaying rewards has been explored by Rotting Bandit, this approach suffers from inefficiencies due to their initial exploration strategies, such as relying on round-robin sampling to estimate decay rates.

To address these fundamental challenges, we propose a novel bandit algorithm that explicitly models and adapts to the reward decay phenomenon in LLM iterative refinement. Furthermore, our approach abstracts arms as embeddings, treating these as the context for each arm. Traditional LinUCB-like approaches [Li et al., 2010] often have arm features that are disjoint with each other (e.g., assuming distinct arm models for how arm context affects click-through rates in sports and political news recommendations). In contrast, our method assumes a unified underlying model regarding how prompt embeddings influence rewards. This means that when a new prompt (arm) is generated, its embedding representation allows for a more informed initial estimation of its parameters, effectively mitigating the LLM-specific "cold start" problem often encountered with new prompts.

Finally, by introducing an Expectation-Maximization (EM) algorithm [Moon, 1996], we achieve simultaneous learning of both decay and contextual parameters, thereby circumventing the inefficient exploration phase associated with round-robin sampling in methods like Rotting Bandit.

These techniques allow our system to dynamically balance exploration of potentially effective but less-used prompts with the exploitation of currently high-performing prompts, while actively accounting for their diminishing utility over time.

Our technical contributions are threefold:

- We propose a novel bandit algorithm DR-LinUCB [1] that explicitly incorporates a reward decay model for each arm, enabling adaptive exploration-exploitation in dynamic LLM refinement environments.

- We introduce a method to embed refinement prompts as contextual features for our bandit arms, enabling the joint learning of arm values. Unlike traditional disjoint LinUCB, our method assumes a unified arm model for each LLM task, which allows for a more informed initial estimation of parameters for new prompts, thereby mitigating the LLM-specific "cold start" problem.

- We leverage an Expectation-Maximization (EM) algorithm for the simultaneous learning of both arm contextual parameters and their associated decay parameters from observed rewards. This approach avoids inefficient initial exploration strategies such as round-robin sampling.

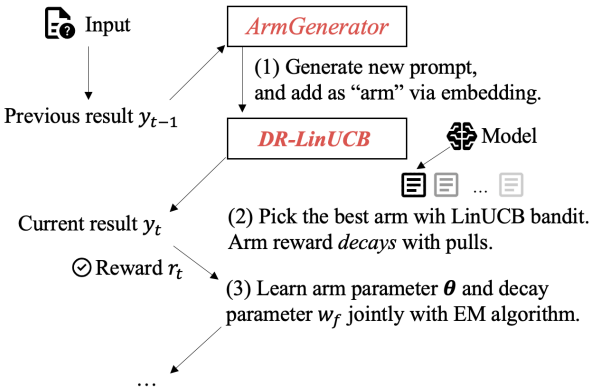

Figure 1: DR-LinUCB's Progressive Content Refinement. This flowchart details its core components: reward decay modeling, UCB-based arm selection and joint EM-based parameter learning.

2 Related works

2.1 Iterative Refinement for Large Language Models

The ability of LLMs to perform iterative self-correction has emerged as a key technique for enhancing generation quality. SELF-REFINE [Madaan et al., 2023] introduced LLMs generating, critiquing, and refining their own outputs using natural language feedback, significantly improving performance across tasks. This concept extended to specific domains: Chen et al. (2024) applied it to machine translation for human-preferred fluency [Chen et al., 2024] , and Hein et al. (2025) developed high-accuracy clinical information extraction pipelines, emphasizing task definition and human-in-the-loop refinement [Hein et al., 2025] . Despite these advances, a key limitation of existing methods is their limited prompt diversity, which our work mitigates through bandit explore-exploitation.

[1] Source code attached for review. GitHub repository will be available at https://github.com/{anonymous_org} upon acceptance.

2.2 Bandit Algorithms for Large Language Model

In multi-step tasks, LLMs need to balance using proven prompts (exploitation) with trying new ones (exploration). Bandit algorithms help by intelligently guiding this choice, preventing over-reliance on old prompts and efficiently discovering better ones.

In code refinement, the REx (REfine, Explore, Exploit) algorithm [Tang et al., 2024] frames iterative LLM-based code repair as an arm-acquiring bandit problem. Each generated code is an arm, and refining it yields a reward (passing tests). REx uses Thompson Sampling with heuristic-informed priors to balance exploring new code and exploiting promising ones. It significantly improved problem-solving and reduced LLM calls across diverse coding tasks.

For prompt optimization, OPTS (Optimizing Prompts with sTrategy Selection) [Ashizawa et al., 2025] explicitly selects prompt design strategies using bandit algorithms. Traditional methods often rely on LLMs implicitly choosing strategies, which can be suboptimal. OPTS(TS), its most effective variant, treats each strategy as an arm, learning its efficacy via Thompson Sampling [Thompson, 1933]. This approach improved prompt performance by up to 50% by making strategy selection adaptive and data-driven.

Unlike REx and OPTS, which respectively refine content candidates with a fixed prompt and optimize a single prompt, our approach dynamically selects generated prompts. This enables prompt selection to be adapted based on the current output state, allowing for contextual content refinement.

2.3 Bandit Algorithms with Decaying Rewards

The "Rotting Bandits" framework [Levine et al., 2017] addresses Multi-Armed Bandit (MAB) problems where an arm's expected reward decays with its pull count, departing from classical stationary assumptions. They proposed non-parametric methods like Sliding-Window Average and parametric approaches such as Closest To Origin for this setting. However, their reliance on initial round-robin exploration for decay rate estimation can be inefficient.

Our work enhances this by introducing an Expectation-Maximization (EM) algorithm, enabling simultaneous learning of decay and contextual parameters, thus circumventing the inefficient initial exploration inherent in traditional Rotting Bandit approaches.

2.4 Action embeddings

Large action spaces demand efficiency and scalability. action embeddings schemes address this by simplifying action representations. Action embedding is vital for Off-Policy Evaluation (OPE). Marginalized IPS (MIPS) [Saito and Joachims, 2022] uses embeddings to reduce OPE variance in large action spaces, enhancing generalizability. Furthermore, action embeddings are useful in position bias estimation [Ishikawa et al., 2024].

Our work applies action embeddings for efficiency in large LLM prompt spaces. Further, using action embeddings within an arm-acquiring bandit framework, we enable adaptation to new "arms" (e.g., novel prompt strategies), facilitating robust LLM optimization.

3 Problem Setting

We consider an arm-acquiring contextual bandit problem with decaying rewards, where an agent's objective is to maximize cumulative reward over T_{max} timesteps. At each timestep t, an agent selects an arm $i(t)$ from the available set $\mathcal{A}_t$ according to a policy π. Each arm $i \in \mathcal{A}_t$ is characterized by a d-dimensional embedding $\mathbf{e}_i \in \mathbb{R}^d$. We assume these arm embeddings lie within a compact subset of $\mathbb{R}^d$, for example, a unit sphere or a hypercube. The set of available arms $\mathcal{A}_t$ can dynamically expand over time ($\mathcal{A}_{t+1} \supseteq \mathcal{A}_t$) with arm acquisition. Upon selection, the agent observes a real-valued reward $r_t \in [0, R_{\text{max}}]$ for some $R_{\text{max}} > 0$.

The observed reward r for arm i, after being pulled N_i times, follows an exponentially decaying model:

$$r = \tilde{\mu}_i \cdot \exp(-N_i \cdot f_i) \tag{1}$$

where $\tilde{\mu}_i = \mathbf{e}_i^\top \boldsymbol{\theta}$ is the initial expected reward (linearly modeled by a global parameter $\boldsymbol{\theta} \in \mathbb{R}^d$), and $f_i = \max(0, \mathbf{e}_i^\top \mathbf{w}_f)$ is the decay rate (determined by a global decay parameter $\mathbf{w}_f \in \mathbb{R}^d$).

The agent's ultimate objective is to find a policy π that maximizes the cumulative reward $\sum_{t=1}^{T_{\text{max}}} r_t$, equivalent to minimizing the cumulative regret $R(T_{\text{max}})$:

$$R(T_{\text{max}}) = \sum_{t=1}^{T_{\text{max}}} (r_t^* - r_t) \tag{2}$$

Here, r_t^* denotes the reward from an oracle selecting the optimal arm at timestep t considering its current decay.

4 Decaying Reward Joint LinUCB (DR-LinUCB)

DR-LinUCB is the core arm selection and parameter learning algorithm. It integrates a reward decay model and an EM algorithm into a joint linear UCB framework, where parameters are shared or learned across arms through their embeddings. Pseudo algorithm of DR-LinUCB is given by Algorithm 1.

DR-LinUCB consists of several key elements, detailed below sections.

4.0.1 UCB Selection with Reward Decay

At each timestep t, DR-LinUCB selects an arm $i(t) \in \mathcal{A}_t$ from the available arms that maximizes the following criterion:

$$i(t) = \text{argmax}_{i \in [K_t]} \left(\mathbf{e}_i^\top \hat{\boldsymbol{\theta}} + C\sqrt{\mathbf{e}_i^\top \mathbf{A}^{-1} \mathbf{e}_i} \right) \cdot d_i \tag{3}$$

where $d_i = \exp\left(-N_i \max(0, \mathbf{e}_i^\top \mathbf{w}_f)\right)$.

The first term, $\mathbf{e}_i^\top \hat{\boldsymbol{\theta}}$, represents the estimated undecayed expected reward for arm i, based on its embedding $\mathbf{e}_i$ and the globally learned parameter $\hat{\boldsymbol{\theta}}$. The second term, $C\sqrt{\mathbf{e}_i^\top \mathbf{A}^{-1} \mathbf{e}_i}$, is the exploration term (UCB term) that accounts for uncertainty, where C is a hyperparameter controlling the degree of exploration. Finally, the exponential term, $\exp\left(-N_i \max(0, \mathbf{e}_i^\top \mathbf{w}_f)\right)$, models the decay of reward based on the number of times arm i has been pulled, N_i, and its specific decay rate $\mathbf{e}_i^\top \mathbf{w}_f$. The $\max(0, \cdot)$ operation ensures the decay factor is non-negative. This comprehensive criterion considers the saturation effect from excessive exploitation, thereby encouraging the algorithm to explore new arms when existing ones show diminishing returns.

4.0.2 Expectation-Maximization (EM) Algorithm for Parameter Learning

DR-LinUCB employs an EM algorithm to simultaneously learn both the global reward model parameters $\hat{\boldsymbol{\theta}}$ and the global decay model parameters $\mathbf{w}_f$. This approach is particularly effective for estimating the hidden variable, the undecayed reward $\tilde{\mu}_{i(t)}$, from the observed decayed reward r_t.

Algorithm 1 Decaying Reward Joint LinUCB (DR-LinUCB)

Require: $\mathcal{A}_0, h_0, T_{\text{max}}, C, \lambda, \lambda_f, \epsilon_{\text{log}}, \epsilon_{\text{decay}}$
1: **Initialize:** $N_i = 0$ for $i \in \mathcal{A}_0$
2: $\mathbf{A} = \lambda \mathbf{I}_d$, $\mathbf{b} = \mathbf{0}_d$, $\hat{\boldsymbol{\theta}} = \mathbf{0}_d$
3: $\mathbf{A}_f = \lambda_f \mathbf{I}_d$, $\mathbf{b}_f = \mathbf{0}_d$, $\mathbf{w}_f = \mathbf{0}_d$
4: **for** $t = 1, 2, \ldots, T_{\text{max}}$ **do**
5: $\quad \mathcal{A}_t \leftarrow \mathcal{A}_{t-1} \cup \text{ArmGenerator}(\mathcal{A}_{t-1}, h_{t-1})$
6: $\quad \boldsymbol{E} \leftarrow \{\text{Embed}(a) \mid a \in \mathcal{A}_t\}$
7: $\quad d_i = \exp\left(-N_i \max(0, \mathbf{e}_i^\top \mathbf{w}_f)\right)$
8: $\quad i(t) \leftarrow \text{argmax}_{i \in \mathcal{A}_t}(\mathbf{e}_i^\top \hat{\boldsymbol{\theta}} + C\sqrt{\mathbf{e}_i^\top \mathbf{A}^{-1} \mathbf{e}_i}) \cdot d_i$
9: $\quad$ Observe reward r_t for arm $i(t)$
10: $\quad$ **while** $\hat{\boldsymbol{\theta}}$, $\mathbf{w}_f$ not converged **do**
11: $\quad\quad \tilde{\mu}_{i(t)} \leftarrow r_t / d_{i(t)}$
12: $\quad\quad \hat{\boldsymbol{\theta}} \leftarrow (\mathbf{A} + \mathbf{e}_{i(t)} \mathbf{e}_{i(t)}^\top)^{-1} (\mathbf{b} + \tilde{\mu}_{i(t)} \mathbf{e}_{i(t)})$
13: $\quad\quad$ **if** $r_t > 0$ and $\tilde{\mu}_{i(t)} > 0$ and $r_t \neq \tilde{\mu}_{i(t)}$ **then**
14: $\quad\quad\quad y \leftarrow \log(\tilde{\mu}_{i(t)} + \epsilon_{\text{log}}) - \log(r_t + \epsilon_{\text{log}})$
15: $\quad\quad\quad \mathbf{x} \leftarrow N_{i(t)} \mathbf{e}_{i(t)}$
16: $\quad\quad\quad \mathbf{w}_f \leftarrow (\mathbf{A}_f + \mathbf{x}(\mathbf{x})^\top)^{-1} (\mathbf{b}_f + y\mathbf{x})$
17: $\quad\quad$ **end if**
18: $\quad$ **end while**
19: $\quad \mathbf{A} \leftarrow \mathbf{A} + \mathbf{e}_{i(t)} \mathbf{e}_{i(t)}^\top$
20: $\quad \mathbf{b} \leftarrow \mathbf{b} + \tilde{\mu}_{i(t)} \mathbf{e}_{i(t)}$
21: $\quad \mathbf{A}_f \leftarrow \mathbf{A}_f + \mathbf{x}(\mathbf{x})^\top$
22: $\quad \mathbf{b}_f \leftarrow \mathbf{b}_f + y\mathbf{x}$
23: $\quad h_t \leftarrow h_{t-1} \cup \{i(t), r_t\}$
24: $\quad N_{i(t)} \leftarrow N_{i(t)} + 1$
25: **end for**

The E-Step (Line 11) uses the current global decay model parameters $\mathbf{w}_f$ to *undecay* the observed reward r_t, thereby estimating the undecayed reward $\tilde{\mu}_{i(t)}$. The calculation is performed as $\tilde{\mu}_{i(t)} \leftarrow r_t / \exp\left(-N_{i(t)} \cdot \max(0, \mathbf{e}_{i(t)}^\top \mathbf{w}_f)\right)$. A small constant ϵ_{decay} is introduced for numerical stability, treating the decay rate as 1 if it's very small.

The M-Step (Lines 12-16) then updates the global reward model parameters $\hat{\boldsymbol{\theta}}$ and the global decay model parameters $\mathbf{w}_f$ using the $\tilde{\mu}_{i(t)}$ estimated in the E-Step. Specifically, $\hat{\boldsymbol{\theta}}$ is updated using standard linear ridge regression, with the undecayed reward $\tilde{\mu}_{i(t)}$ as the target: $\hat{\boldsymbol{\theta}} \leftarrow (\mathbf{A} + \mathbf{e}_{i(t)} \mathbf{e}_{i(t)}^\top)^{-1} (\mathbf{b} + \tilde{\mu}_{i(t)} \cdot \mathbf{e}_{i(t)})$. The decay model parameters $\mathbf{w}_f$ are updated by transforming the exponential decay into a linear regression problem, a process detailed in the subsequent subsection.

Transformation to a Linear Regression Problem for $\mathbf{w}_f$ Estimation

To estimate the global decay parameter $\mathbf{w}_f$, the exponential decay model $r_t = \tilde{\mu}_{i(t)} \cdot \exp\left(-N_{i(t)} \cdot (\mathbf{e}_{i(t)}^\top \mathbf{w}_f)\right)$ is transformed into a linear regression problem. Taking the natural logarithm of both sides yields:

$$\begin{aligned} \log(r_t) &= \log(\tilde{\mu}_{i(t)}) - N_{i(t)} \cdot (\mathbf{e}_{i(t)}^\top \mathbf{w}_f) \\ \log(\tilde{\mu}_{i(t)}) - \log(r_t) &= N_{i(t)} \cdot (\mathbf{e}_{i(t)}^\top \mathbf{w}_f) \end{aligned} \tag{4}$$

Rearranging this equation, we define the target variable y' and feature vector $\mathbf{x}'$ for a linear regression of the form $y' = \mathbf{x}'^\top \mathbf{w}_f$:

$$\begin{aligned} y' &= \log(\tilde{\mu}_{i(t)} + \epsilon_{\text{log}}) - \log(r_t + \epsilon_{\text{log}}) \\ \mathbf{x}' &= N_{i(t)} \cdot \mathbf{e}_{i(t)} \end{aligned} \tag{5}$$

Here, for numerical stability in computation, we introduce a small positive constant $\epsilon_{\text{log}} \ll r_t, \tilde{\mu}_{i(t)}$, preventing issues with $\log(0)$.

The online update of $\mathbf{w}_f$ proceeds as follows. First, a crucial condition is checked in Line 15: 'If $r_t > 0$ and $\tilde{\mu}_{i(t)} > 0$ and $r_t \neq \tilde{\mu}_{i(t)}$ then'. This ensures that logarithmic transformation is valid and that actual decay has occurred, making the update meaningful.

If the conditions are met, y' and $\mathbf{x}'$ are computed using the formulas in Equations (5). Then we update the global decay parameter $\mathbf{w}_f$ through a standard online ridge regression framework (Line 16). After the loop of EM algorithm, sufficient statistics $\mathbf{A}$, $\mathbf{A}_f$ $\mathbf{b}$ and $\mathbf{b}_f$ are updated (Lines 19-23), utilizing the closed-form solutions for online ridge regression [Li et al., 2010].

Application to Progressive Content Refinement

In the preceding sections, we introduced DR-LinUCB, a versatile bandit algorithm designed for dynamic environments where rewards decay, and both arm context and decay parameters are learned using an EM algorithm. Now, we will demonstrate how this general algorithmic framework can be specifically applied to enhance LLMs through progressive content refinement.

We represent an LLM task as a tuple $(p_{\text{init}}, \text{Evaluator}())$, where p_{init} is the original prompt that defines the LLM's primary objective (e.g., "Summarize the following article"), and Evaluator() is a function that quantifies the quality or utility of the LLM's output (y_t) in response to a given prompt.

To apply DR-LinUCB to LLM tasks, each refinement prompt is treated as an "arm" within its framework. The process initializes $\mathcal{A}_0$ with p_{init}. In each step, the *ArmGenerator* function (Algorithm 2) expands the set of available arms.

Algorithm 2 ArmGenerator for Content Refinement

```
function ARMGENERATOR(A_{t-1}, h_{t-1})
    if t > 1 then h_{t-1} ← h_{t-1} ∪ {y_{t-1}}
    else return {}
    end if
    p_feedback ← LLM(h_{t-1})
    p_new ← LLM(p_init, p_feedback, h_{t-1})
    return {p_new}
end function
```

This function leverages the LLM to generate a feedback prompt p_{feedback} from the history h_{t-1}, and then generate a new refinement prompt p_{new} based on p_{init}, p_{feedback}, and h_{t-1}. This self-refinement process draws inspiration from [Madaan et al., 2023].

Additionally, we introduce an *EvoArmGenerator* that employs EvoPrompt [Guo et al., 2025] as a variant of arm generator. This evolutionary algorithm-based strategy explores a broader, more diverse set of refinement prompts. While self-refinement generation can converge to local optima, evolutionary algorithms excel at systematic exploration of wider solution spaces through mutation and crossover. This diversity is crucial for ablation studies comparing DR-LinUCB's learning-based selection against simpler strategies like random selection, especially when a rich pool of diverse arms is needed.

In Algorithm 1, the step for "Observe reward r_t" works as follows: First, the chosen prompt, $i(t)$, is given to the LLM, which then generates an output, y_t. The reward r_t is subsequently calculated by feeding this output y_t into the task's predefined Evaluator() function, resulting in $r_t \leftarrow \text{Evaluator}(y_t)$. These steps enable the application of the DR-LinUCB framework to LLM tasks for progressive content refinement.

Prompt Embedding

To leverage our refinement prompts as arms within the DR-LinUCB framework, we employ a prompt

embedding strategy. We begin by utilizing a fine-tuned MINILM [Wang et al., 2020] model [2] to transform prompts into a high-dimensional embedding space. Then, we embed 200 prompts from the 'awesome-chatgpt-prompts' [3] dataset into this space. Following this, we apply Principal Component Analysis (PCA) [Maćkiewicz and Ratajczak, 1993] to these embeddings to reduce their dimensionality, obtaining a 5-dimensional principal component vector. In this research, we leverage this fine-tuned sentence transformer model and the trained PCA model to embed new prompts and compress their dimensions before use in our bandit algorithm.

5 Evaluation

We study two different domains that each involve complex natural language reasoning and generation.

1. **Math Reasoning**: GSM8K (Grade School Math 8k) [Cobbe et al., 2021] is a challenging dataset of elementary-level math word problems. The task is to read a natural-language problem description and generate a step-by-step solution, involving numerical calculations and logical reasoning, to find the final numerical answer. It assesses a model's multi-step mathematical reasoning and its ability to convert linguistic information into a solvable procedure.

   As a metric, we report the average success rate. A "success" is defined as generating the exact numerical answer defined in the dataset.

2. **Sentiment Reversal**: Sentiment Reversal is a long-form text style transfer task [Madaan et al., 2023; Zhang et al., 2015]. Given a text passage with a specific sentiment (e.g., negative), the goal is to rewrite the entire passage to a target sentiment (e.g., very positive), not merely reversing it. This task evaluates a model's ability for fine-grained content editing, requiring not only a complete shift in emotional tone but also an adjustment of sentiment intensity. It necessitates understanding the original meaning and tone, and then systematically altering vocabulary, phrasing, and narrative to achieve the specified opposing sentiment and intensity while maintaining coherence.

   As a metric, we report the average scores. We first calculate the sentiment using a ModernBert-based Multilingual Sentiment Classification Model [tabularisai et al., 2025]. If the predicted sentiment matches the target sentiment, a score of 1.0 is assigned. Otherwise, we employ an LLM as a judge to calculate a score reflecting the alignment of the generated text's sentiment to the target. The LLM model version used for judging in each experiment matches the version employed within that experiment.

We use these tasks to study following research questions:
RQ1: Can iterative-refinement achieve higher performance than a single-call strategy?
RQ2: Can *reward decay* reduce over-exploitation and contribute to total performance?
RQ3: Does *reward decay* accelerate the convergence and performance improvement of iterative refinement?
RQ4: Which approach provides better performance: ArmGenerator or EvoArmGenerator?
RQ5: Does the combination of joint LinUCB and reward decay achieve superior overall performance?

To investigate these questions, we study a range of baselines:

- **Single Call**: LLM output from initial prompt only. Serves as a performance lower bound.
- **Random Exploration**: Uses *EvoArmGenerator* but selects prompts uniformly at random. Evaluates the benefit of learning over pure exploration.
- **JointLinUCB**: Uses *EvoArmGenerator* and LinUCB for prompt selection, but assumes stationary rewards, lacking decay modeling. Isolates decay's contribution.
- **EvoLinUCB**: This is a variant of the DR-LinUCB method that uses *EvoArmGenerator*. This approach offers diverse prompt generation, but unlike the *ArmGenerator*, it does not use LLM feedback during generation.
- **Self-Refine** [Madaan et al., 2023]: Iterative refinement where LLM generates feedback

[2] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[3] https://huggingface.co/datasets/fka/awesome-chatgpt-prompts

Task	Algorithm	Model	
		ChatGPT3.5-turbo	ChatGPT4o
GSM8K	Single Call	0.187 ± 0.059	0.537 ± 0.055
	Random Exploration	0.310 ± 0.035	0.647 ± 0.049
	JointLinUCB	0.323 ± 0.053	0.610 ± 0.061
	EvoLinUCB	0.340 ± 0.053	0.630 ± 0.087
	Self-Refine	0.687 ± 0.075	$\mathbf{0.913 \pm 0.021}$
	REx	0.530 ± 0.017	0.880 ± 0.010
	DR-LinUCB(Ours)	$\mathbf{0.790 \pm 0.035}$	0.900 ± 0.010
Sentiment Inverse	Single Call	0.839 ± 0.015	0.950 ± 0.016
	Random Exploration	0.935 ± 0.028	0.980 ± 0.008
	JointLinUCB	0.922 ± 0.010	0.973 ± 0.013
	EvoLinUCB	0.944 ± 0.014	0.984 ± 0.002
	Self-Refine	0.879 ± 0.009	0.989 ± 0.004
	REx	0.931 ± 0.005	0.985 ± 0.008
	DR-LinUCB(Ours)	$\mathbf{0.952 \pm 0.009}$	$\mathbf{1.000 \pm 0.000}$

Table 1: Performance summary (mean $\pm$ standard deviation) for each task, algorithm, and model. Metrics are defined in Section 5.

and prompts. The prompt selection is greedy, typically choosing the latest generated prompt without adaptive learning. This corresponds to repeated *ArmGenerator* use without sophisticated selection.

- **REx** (REfine, Explore, Exploit) [Tang et al., 2024]: Uses Thompson Sampling for content refinement. The refinement prompt is fixed, unlike our dynamic prompt selection.

For each experiment, we evaluated the models on 100 distinct samples for each of three random seeds. The reported results are averaged over these three independent runs, effectively evaluating performance on a total of 300 samples for each task and algorithm. For each sample, algorithm has up to six time steps.

The Value of Iterative LLM Interaction

The comparison between Single Call and Random Exploration in Table 1 demonstrates that multi-step iterative processes significantly enhance performance.

The Single Call baseline consistently yields the lowest scores. For instance, in GSM8K with ChatGPT3.5-turbo, Single Call scored 0.187. In contrast, Random Exploration shows a substantial improvement. For the same task, "Random Exploration" achieves 0.310, an approximate 65% increase over Single Call.

This clearly indicates that engaging the LLM in a multi-step iterative process leads to significant performance improvement over a single-shot approach. This observation answers RQ1, confirming that iterative LLM usage, even in its most basic form, yields higher performance than a single call strategy. It underscores the fundamental value of employing LLMs iteratively to refine and optimize their outputs.

DR-LinUCB Outperforms Baselines and SOTA

The results in Table 1 demonstrates that our proposed DR-LinUCB consistently achieves superior performance against both our ablation study baselines and state-of-the-art (SOTA) methods like Self-Refine and REx.

Specifically, with ChatGPT3.5-turbo, DR-LinUCB leads in both GSM8K (0.790) and Sentiment Inverse (0.952), significantly surpassing Self-Refine (0.687 for GSM8K, 0.879 for Sentiment Inverse) and REx (0.530 for GSM8K, 0.931 for Sentiment Inverse). This highlights DR-LinUCB's capability in enhancing less powerful LLMs. With ChatGPT4o, DR-LinUCB achieves 0.900 for GSM8K, outperforming REx (0.880) and closely trailing Self-Refine (0.913). In particular, for Sentiment Inverse with ChatGPT4o, DR-LinUCB achieves a perfect score of 1.000, outperforming all other methods. We further discuss the feasibility and implications of this perfect score in Section 6.

These results unequivocally answer RQ5. The integration of these mechanisms allows DR-LinUCB to effectively learn optimal prompt arms while preventing over-exploitation, leading to robust and often best-in-class performance.

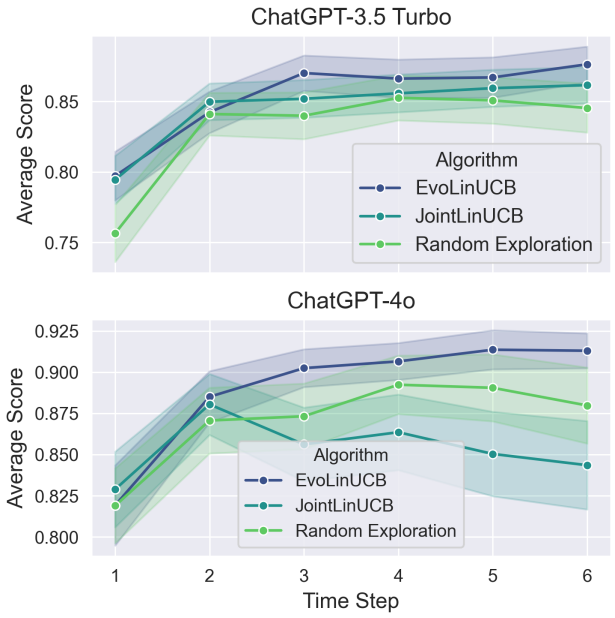

Figure 2: History of average scores in the Sentiment Reversal Task for ChatGPT-3.5 Turbo (top) and ChatGPT-4o (bottom). EvoLinUCB incorporates both arm decay and arm learning, while JointLinUCB only features arm learning. Random Exploration uniformly selects arms generated by EvoArmGenerator. Only instances not correctly answered in the first attempt are included, allowing for a comparison of iterative improvement effects. The shaded area indicates the standard error from three experimental runs.

Addressing RQ2 and RQ3: Insights from Algorithmic Component Analysis

To ensure consistent conditions for prompt generation, our comparison focuses on three models that utilize the EvoPrompt Generator. Furthermore, only instances answered incorrectly in the first attempt are included to allow a comparison of iterative improvement effects.

As depicted in Figure 2, all three algorithms show comparable improvements up to the second iteration. However, from the third iteration onwards, EvoLinUCB consistently outperforms the others for both ChatGPT-3.5 Turbo and ChatGPT-4o. This superior performance of EvoLinUCB can be attributed to its incorporation of both reward decay and arm learning.

In particular, JointLinUCB performs worse than Random Exploration in the case of ChatGPT-4o. This observation supports our hypothesis that mechanisms preventing over-exploitation, like EvoLinUCB's decay or Random Exploration, improve effectiveness over solely exploiting estimated "good" arms. This finding directly addresses RQ2, indicating that reward decay effectively mitigates over-exploitation, leading to enhanced performance.

Moreover, by observing the score history, we can see that the scores converge relatively quickly. This rapid convergence provides an answer to RQ3, suggesting that effective improvements can be achieved within a limited number of iterative steps.

Suitability of Arm Generation Approaches

DR-LinUCB (with *ArmGenerator*) significantly outperforms EvoLinUCB in Sentiment Inverse (0.952 vs. 0.944 with GPT-3.5) and achieves a substantial gain in GSM8K over JointLinUCB (0.790 vs. 0.323). These results suggest that feedback-driven prompt exploration is generally more effective than evolutionary modification.

However, this effectiveness is task-dependent; while Self-Refine excels in GSM8K with GPT-4o (0.913), it underperforms JointLinUCB in Sentiment Inverse, likely because iterative correction can introduce noise or suboptimal steering in simpler tasks. We conclude that *ArmGenerator* is most robust when integrated with exploration-exploitation strategies.

6 Conclusion

This paper introduces Decaying Reward Joint LinUCB (DR-LinUCB), a novel bandit algorithm designed for iterative LLM content refinement. DR-LinUCB explicitly models the "saturation effect" through reward decay and jointly learns contextual and decay parameters. Our experiments on GSM8K and Sentiment Reversal tasks demonstrate DR-LinUCB's superior performance compared to various baselines and state-of-the-art methods. We confirmed that incorporating reward decay effectively prevents over-exploitation, leading to enhanced performance and accelerated convergence.

DR-LinUCB offers a significant advancement in optimizing LLM iterative refinement, providing a principled approach to balance exploration and exploitation in dynamic environments where the utility of refinement strategies can diminish over time.

7 Limitations

While our proposed DR-LinUCB framework shows promising results, several limitations remain to be addressed in future work.

Feasibility of Score 1.0 in Sentiment Reversal Task Historically, LLMs have struggled to achieve 100% accuracy on simple tasks, largely due to inherent flaws in existing benchmarks. The Platinum benchmark [Vendrow et al., 2025] demonstrates that many benchmarks contain significant errors; for instance, in SVAMP [Patel et al., 2021], a question was mislabeled with an incorrect solution. In contrast, our sentiment reversal task evaluates performance using a pretrained sentiment analysis model and an LLM judge, effectively eliminating the issue of mislabeled ground truth.

While the task is relatively simple—with even the Self-Refine model achieving a high score of 0.989—the fact that 100% accuracy is achievable aligns with the findings of the Platinum Benchmarks paper regarding carefully curated tasks. However, it remains a limitation that our current evaluation is focused on this feasible task, and further validation on more complex benchmarks where 100% accuracy is not yet reachable is required.

Minimizing Cost of LLM Interactions The iterative nature of our approach, combined with multiple LLM calls for ArmGenerator and reward evaluation within each timestep, can lead to significant computational overhead and increased API costs. In real-world, high-throughput scenarios, the latency and expenses associated with these repeated inferences could be limiting factors. While our reward decay model implicitly helps minimize unnecessary calls by discouraging the over-exploitation of ineffective prompts, the framework is not yet explicitly optimized for this purpose. Future research should investigate how to leverage decay modeling more strategically to reduce the total number of LLM interactions required, improving overall cost-effectiveness.

Sensitivity to Hyperparameters Like many bandit algorithms, DR-LinUCB relies on hyperparameters such as C, λ, λ_f, ϵ_{log}, and ϵ_{decay}. The performance of the algorithm can be sensitive to the tuning of these parameters. While we have identified effective settings for our specific tasks, optimal tuning for diverse LLM applications might require extensive experimentation. The development of adaptive or self-tuning mechanisms for these hyperparameters remains an important direction to enhance the robustness and ease of use of DR-LinUCB.

References

Rin Ashizawa, Yoichi Hirose, Nozomu Yoshinari, Kento Uchida, and Shinichi Shirakawa. 2025. Bandit-based prompt design strategy selection improves prompt optimizers. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 20799–20817, Vienna, Austria. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, and et al. Ryder. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Pinzhen Chen, Zhicheng Guo, Barry Haddow, and Kenneth Heafield. 2024. Iterative translation refinement with large language models. In *Proceedings of the 25th Annual Conference of the European Association for Machine Translation (Volume 1)*, pages 181–190, Sheffield, UK. European Association for Machine Translation (EAMT).

Hyung Won Chung, Le Hou, and et al. Longpre. 2024. Scaling instruction-finetuned language models. *J. Mach. Learn. Res.*, 25(1).

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2025. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. *Preprint*, arXiv:2309.08532.

David Hein, Alana Christie, Michael Holcomb, Bingqing Xie, AJ Jain, Joseph Vento, Neil Rakheja, Ameer Hamza Shakur, Scott Christley, Lindsay G. Cowell, James Brugarolas, Andrew R. Jamieson, and Payal Kapur. 2025. Iterative refinement and goal articulation to optimize large language models for clinical information extraction. *npj Digital Medicine*, 8(1):301.

Shion Ishikawa, Yun Ching Liu, Young-Joo Chung, and Yu Hirate. 2024. Position bias estimation with item embedding for sparse dataset. In *Companion Proceedings of the ACM Web Conference 2024*, WWW '24, page 895–898, New York, NY, USA. Association for Computing Machinery.

Nir Levine, Koby Crammer, and Shie Mannor. 2017. Rotting bandits. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3077–3086, Red Hook, NY, USA. Curran Associates Inc.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 661–670, New York, NY, USA. Association for Computing Machinery.

Aman Madaan, Niket Tandon, and et al. Gupta. 2023. Self-refine: iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Andrzej Maćkiewicz and Waldemar Ratajczak. 1993. Principal components analysis (pca). *Computers Geosciences*, 19(3):303–342.

T.K. Moon. 1996. The expectation-maximization algorithm. *IEEE Signal Processing Magazine*, 13(6):47–60.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Yuta Saito and Thorsten Joachims. 2022. Off-policy evaluation for large action spaces via embeddings. *Preprint*, arXiv:2202.06317.

tabularisai, Samuel Gyamfi, Vadim Borisov, and Richard H. Schreiber. 2025. multilingual-sentiment-analysis (revision 69afb83).

Hao Tang, Keya Hu, Jin Peng Zhou, Sicheng Zhong, Wei-Long Zheng, Xujie Si, and Kevin Ellis. 2024. Code repair with llms gives an exploration-exploitation tradeoff. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

William R. Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.

Joshua Vendrow, Edward Vendrow, Sara Beery, and Aleksander Madry. 2025. Do large language model benchmarks test reliability? *Preprint*, arXiv:2502.03461.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 649–657, Cambridge, MA, USA. MIT Press.

8 Appendix

8.1 Implementation Details

This appendix provides a comprehensive overview of the experimental setup and specific hyperparameter configurations used in our study to ensure reproducibility.

8.1.1 Hardware and Software Environment

All experiments were conducted on a MacBook Pro (13-inch, 2020) equipped with a 1.7 GHz Quad-Core Intel Core i7 processor, Intel Iris Plus Graphics 645 (1536 MB), and 16 GB of 2133 MHz LPDDR3 memory. For the LLM interactions, we used OpenAI's API and client library.

8.1.2 Hyperparameters

The specific hyperparameters used for each algorithm are detailed below.

- **DR-LinUCB:** For the DR-LinUCB algorithm, we set the maximum number of iterations to `max_itr=5`, and the early stopping score to `early_stop_score=1.0`.The number of iterations for parameter learning was `T_em=2`. The dimension of the context vector was `d_dim=5`, and the constant for the confidence bound was `C_constant=0.5`. Regularization parameters were set to `lambda_theta=0.1` and `lambda_wf=0.1`. Small epsilon values for logarithmic and decay calculations were `epsilon_log_val=`1×10^{-6} and `epsilon_decay_val=`1×10^{-6} respectively.

- **Rex:** The Rex algorithm utilized a confidence parameter `C=20.0`, a maximum of `max_llm_calls=5` to the LLM, and an early stopping score of `early_stop_score=1.0`.

- **Self-Refine:** For the Self-Refine process, we configured `max_iterations=5` and an `early_stop_score=1`.

8.2 Prompts

8.2.1 GSM8K

For the GSM8K (Grade School Math 8K) task, we employ a similar multi-stage prompting framework designed to enhance the accuracy and robustness of problem-solving. This framework consists of an initial generation prompt, a feedback generation prompt, and a refinement prompt, iteratively improving the mathematical reasoning.

8.2.2 Initial Problem Solving Prompt

The first stage aims to generate an initial solution to the mathematical word problem. The prompt guides the LLM to provide a detailed Chain-of-Thought (CoT) before stating the final answer:

Listing 1: Initial Problem Solving Prompt

```
Solve the following mathematical word problem.
    Output the final answer only after showing
    your detailed step-by-step reasoning (Chain-
    of-Thought).

Question: {question}

Answer:
```

This prompt instructs the LLM to solve the given ‘question‘ by first producing a step-by-step reasoning process, followed by the final numerical answer.

8.2.3 Feedback Generation Prompt

After the initial solution attempt, a critical feedback stage is introduced. This prompt directs the LLM to act as a reviewer, identifying potential issues in the generated reasoning without revealing the correct answer:

Listing 2: Feedback Generation Prompt

```
Review the given mathematical word problem and
    the current reasoning/answer. Provide
    detailed, step-by-step feedback focused on
    identifying potential calculation errors,
    logical flaws, or misinterpretations of the
    question. Do not state the final answer.
    Current reasoning/answer:
```

The LLM is tasked with providing constructive feedback on the ‘current reasoning/answer‘, specifically looking for ‘calculation errors‘, ‘logical flaws‘, or ‘misinterpretations‘ of the problem. A crucial directive is to ‘not state the final answer‘, ensuring the feedback mechanism does not directly provide the solution.

Refinement Prompt The refinement stage utilizes the generated feedback and the history of previous attempts to iteratively improve the solution. This prompt is designed to guide the LLM towards a correct and well-structured answer:

Listing 3: Refinement Prompt

```
**[Refinement Task: Math Word Problem (GSM8K)]**

**Question:** {question}

Review the provided question, the previous
    attempts at solving it, and the detailed
    feedback history. Your task is to **refine
    the entire reasoning chain and the final
    answer**.

**Refinement Directives:**
* **Calculation Accuracy:** Strictly re-check
    every arithmetic step. Identify and correct
    any calculation errors (miscounts, incorrect
     multiplication/division, etc.).
* **Logical Consistency:** Verify the logical
    flow of the Chain-of-Thought (CoT). Ensure
    each step is derived correctly from the
    previous one and aligns with the question's
    premise.
* **Answer Format:** After the refined step-by-
    step reasoning, output the final numerical
    answer clearly. The final output must
    conclude with the phrase: **'The final
    answer is [NUMBER]'**.

**History of Attempts and Feedback:**
---
{history}
---

**Refined Reasoning and Final Answer:**
```

This comprehensive refinement prompt explicitly defines the task as ‘[Refinement Task: Math Word Problem (GSM8K)]‘ and provides specific ‘Refinement Directives‘. These directives guide the LLM to focus on ‘Calculation Accuracy‘, ‘Logical Consistency‘ of the Chain-of-Thought (CoT), and ‘Answer Format‘. It emphasizes the need to re-check all arithmetic steps and ensure the logical flow aligns with the ‘question‘’s premise. Crucially, it mandates a specific output format for the final answer: `'The final answer is [NUMBER]'`. The prompt also incorporates a ‘History of Attempts and Feedback‘, allowing the LLM to learn from past errors and improve its reasoning iteratively.

8.2.4 Sentiment Reversal

Our approach to sentiment reversal leverages a multi-stage prompting strategy with a Large Language Model (LLM). This strategy involves an initial sentiment inversion prompt, a feedback generation prompt, and a refinement prompt, designed to iteratively improve the quality and accuracy of the inverted review.

8.2.5 Initial Sentiment Inversion Prompt

The first stage involves generating an initial inverted review. The prompt is structured to clearly instruct the LLM on the primary task:

Listing 4: Initial Sentiment Inversion Prompt

```
Invert the sentiment of the following review.

Original Review: {original_review_text}

Inverted Review:
```

This prompt directly asks the LLM to perform the sentiment reversal on the provided `original_review_text` and to output the `Inverted Review`.

8.2.6 Feedback Generation Prompt

Following the initial inversion, a feedback mechanism is employed to evaluate the generated inverted review. This feedback is crucial for guiding subsequent refinements. The prompt for generating feedback is as follows:

Listing 5: Feedback Generation Prompt

```
Review the given original review ({
    original_review_text}...) and the current
    inverted review.
Provide concrete suggestions for improvement,
    focusing on whether the sentiment is
    properly
inverted and if the expression is natural.
Consider the Original Sentiment: {
    original_sentiment} and the Target Sentiment
    : {target\_sentiment}.
Current inverted review:
```

This prompt instructs the LLM to act as a critic, providing constructive feedback. It explicitly asks for suggestions focusing on sentiment inversion accuracy and naturalness of expression, taking into account the `original_sentiment` and the desired `target_sentiment`.

8.2.7 Refinement Prompt

The final stage utilizes the feedback to refine the inverted review. This iterative refinement process aims to converge on a high-quality, sentiment-inverted text. The refinement prompt is designed to guide the LLM through this process:

Listing 6: Refinement Prompt

```
Please return only inverted reviews and don't
    return suggestions.
For the task of inverting the sentiment of the
    Original Review: {original\_review\_text
    }...,
refine the review based on the history of
    previous inverted reviews and feedback.
Consider all provided feedback and ensure the
    refined text has a {target\_sentiment}
    sentiment.
Current sentiment is {self.sentiment}.
if sentiment is not enough (e.g., target is very
     positive is current is positive),
change sentiment polarity and change
    Intensifiers / Amplifiers
Please consider current history:
```

This prompt emphasizes that the LLM should only output the refined inverted review, without additional suggestions. It directs the LLM to leverage the `history of previous inverted reviews` and `feedback` to ensure the refined text aligns with the `target_sentiment`. It also provides explicit instructions for cases where the current sentiment is not sufficiently strong, suggesting the modification of sentiment polarity and the use of intensifiers/amplifiers, thereby promoting more robust sentiment manipulation. The prompt also implicitly references an internal `self.sentiment` variable and encourages consideration of the `current history` of refinements.

8.2.8 Prompts for generating arms

8.2.9 `EvoArmGenerator`

This meta-prompt guides the Large Language Model (LLM) to perform evolutionary operations (mutation and crossover) on a target prompt. It leverages information from three "donor" prompts and an experimental history to create a refined prompt. The process is designed to optimize existing prompts while preserving their core objective.

Listing 7: EvoArmGenerator

```
You are an expert prompt optimizer. Your task is
     to perform an evolutionary operation
on the target prompt using three donor prompts.
1.Identify the key elements and style
    differences between Donor 1 and Donor 2.
Donor 1 (Pr1): {donor1_prompt}
Donor 2 (Pr2): {donor2_prompt}
2.Mutate the different parts, considering the
    history of experiment: {history}
3. Combine the different parts with Prompt 3,
    selectively replace it with the different
    parts in Step 2 and generate a new prompt.
    Don't use Donar 1 and Doner 2 as pronoun
4. Cross over the prompt in the Step 3 with the
    following basic prompt and generate a final
    prompt bracketed with <prompt> and </prompt>
5. Do not break original goal (if it's
    mathmatical calculation, don't mutate number
     and formula). Mutate expression and
    approach to accomlish original goal.
6. When mutating, simplify prompt as well (
    especially for mathmatical calculation)
```

The LLM, acting as an "expert prompt optimizer," is instructed to analyze the stylistic and elemental differences between ‘donor1_prompt‘ and ‘donor2_prompt‘. It then mutates relevant parts based on the ‘history‘ of experiments, combines these with an implicit "Prompt 3," and performs a crossover operation with a "basic prompt." Critical directives include maintaining the ‘original goal‘ (e.g., preserving numerical values and formulas in mathematical calculations) while focusing on mutating ‘expression and approach‘. Additionally, simplification of the prompt during mutation is encouraged, particularly for mathematical tasks. The final output prompt is required to be enclosed within ‘<prompt>‘ and ‘</prompt>‘ tags.

8.2.10 History-Based Novel Prompt Generator (`ArmGenerator`)

This meta-prompt aims to generate entirely new prompts that fundamentally deviate from previous solutions, ensuring novelty while strictly adhering to the original task’s objective. It leverages the full ‘Experiment History‘ to strategically guide the LLM towards innovative approaches.

Listing 8: ArmGenerator

```
You are a seasoned strategist overseeing prompt
    evolution. For the given "original prompt,"
    leverage the following historical
    information to its fullest extent and
    generate a **novel prompt that distinctly
    deviates from previous approaches**.

Experiment History: {history}

Instructions for generating the new prompt:
1. **Fundamental Shift in Approach**: Deeply
    analyze the history above and devise a
    completely new approach that intentionally
    departs from the expressions, structures, or
     thought patterns adopted in previous
    attempts.
2. **Absolute Maintenance of Goal**: The
    ultimate goal defined by the original prompt
    , including specific numbers, formulas, or
    data in mathematical calculations, must not
    be altered for any reason. These must be
    entirely preserved in the new prompt.
3. **Refinement and Simplification of Expression
    **: The new approach should be more refined
    and concise to achieve the original goal.
    Especially in cases involving complex
    calculations or logic, eliminate redundancy
    and pursue the most efficient and clear
    expression possible.

4. Generate a final prompt bracketed with <
    prompt> and </prompt>
```

Acting as a "seasoned strategist," the LLM is tasked with generating a ‘novel prompt‘ that ‘distinctly deviates from previous approaches‘ by thoroughly analyzing the ‘Experiment History‘. The core instructions emphasize a ‘Fundamental Shift in Approach‘, meaning a deliberate departure from past expressions, structures, or thought patterns. Despite this push for novelty, the ‘Absolute Maintenance of Goal‘ is paramount, ensuring that critical elements like numbers and formulas in mathematical contexts remain unchanged. Furthermore, the new approach should exhibit ‘Refinement and Simplification of Expression‘, eliminating redundancy and promoting efficiency, especially for complex tasks. As with the evolutionary prompt, the final output must be enclosed within $< prompt >$ and $< /prompt >$ tags.