A Constrained Text Revision Agent via Iterative Planning and Searching

Hannan Cao   **Hwee Tou Ng**
Department of Computer Science, National University of Singapore
caoh@u.nus.edu,nght@comp.nus.edu.sg

Abstract

Existing text revision systems are capable of generating fluent and coherent text, but struggle with constrained text revision (CTR), which requires adherence to specific constraints. Furthermore, adapting these systems to diverse constraints is challenging. To bridge this gap, we introduce **CRAFT**, a **C**onstrained **R**evision **A**gent **F**or **T**ext, focusing on CTR. CRAFT utilizes a planner, a reviser (*i.e.,* a large language model), and adaptable tools to generate revisions tailored to different scenarios. Specifically, we propose an iterative self-training alignment method to construct the planner, which generates tool usage and text revision plans. Furthermore, we propose Tool-Guided Monte Carlo Tree Search (TG-MCTS), a novel CTR algorithm that extends MCTS with tool-guided expansion and evaluation, enabling the search for optimal revision strategies across various scenarios. To evaluate CRAFT, we introduce CORD (**CO**nstrained **R**evision **D**ataset), a dataset with multi-level constrained instructions for paragraph-level revision. Experimental results show that CRAFT outperforms baselines in both constraint adherence and revision quality. Furthermore, CRAFT exhibits robust performance across diverse use cases, including plain text and LaTeX revision.[1]

1 Introduction

Large language models (LLMs) excel at generating fluent and coherent text, motivating researchers to develop various text revision systems (Raheja et al., 2023; Cao et al., 2023). In practice, users expect text revision systems to revise entire passages while adhering to specific constraints, such as word limits or keyword constraints (Chen et al., 2024). We define this task as *constrained text revision* (CTR), a subtask of constrained text generation (CTG) (Liang et al., 2024a).

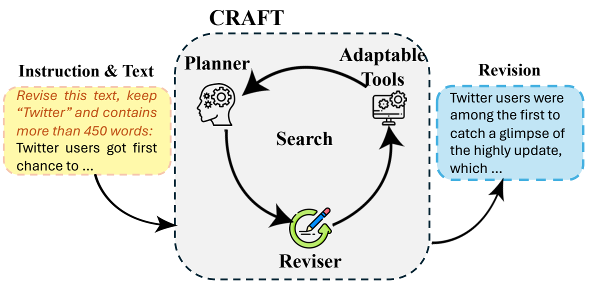

CRAFT
Instruction & Text
Revise this text, keep "Twitter" and contains more than 450 words: Twitter users got first chance to ...
Planner
Adaptable Tools
Search
Reviser
Revision
Twitter users were among the first to catch a glimpse of the highly update, which ...

Figure 1: Illustration of our proposed CRAFT for CTR.

CTR involves modifying text according to specific instructions, requiring LLMs to interpret diverse constraints, plan tasks (Liang et al., 2024b), and interact with tools (Schick et al., 2023), making it more complex than traditional text generation (Yao et al., 2024). Moreover, CTR applies to various scenarios, such as plain text and LaTeX revisions, making it more challenging to design a system that accommodates all possible constraints and use cases. An ideal solution would be a highly adaptable text revision system capable of handling diverse scenarios efficiently. However, existing text revision LLMs (Raheja et al., 2023; Shu et al., 2024) rely on supervised fine-tuning (SFT) with labeled in-domain data, limiting their adaptability to diverse constraints or use cases.

To bridge this gap, we employ a vanilla LLM (*i.e.,* not fine-tuned on task-specific data) as a text revision LLM (reviser), ensuring high adaptability. However, directly using the reviser often yields suboptimal performance across different constraints and domains. Therefore, we introduce **CRAFT** (Fig. 1), a **C**onstrained **R**evision **A**gent **F**or **T**ext. CRAFT operates iteratively in two phases: (1) the *planning* phase, where the ***planner*** formulates ***tool*** usage and revision plans based on different scenarios, and (2) a *searching* phase, where the selected tools guide the search algorithm to identify optimal revision plans for the ***reviser***.

In the *planning* phase, the planner is responsible

[1] https://github.com/nusnlp/CRAFT

for interpreting various instructions to formulate tool usage and text revision plans. However, verifiable constrained instructions often involve numerical symbols (Zhou et al., 2023; Jiang et al., 2024), which LLMs frequently misinterpret (Chen et al., 2024). To address this weakness and enhance the planner's ability to mimic human text revision plans, we propose a two-step approach to build the planner: (1) utilizing GPT-4o to generate planning and tool-usage data via in-context learning (ICL), which is then used to train an initial planner through SFT; and (2) iteratively refining the planner using a self-training alignment method, enabling it to learn from its own mistakes.

To enhance the reviser's performance under diverse constraints while ensuring adaptability, we introduce Tool-Guided Monte Carlo Tree Search (TG-MCTS) for the *search* phase. TG-MCTS is a novel CTR algorithm that integrates external tools to guide text revisions across various constraints and domains. It extends the traditional Monte Carlo Tree Search (MCTS) framework (Browne et al., 2012) through two key innovations: *tool-guided expansion* and *tool-based evaluation*. During *tool-guided expansion*, the reviser first generates revisions using plans formulated by the planner. TG-MCTS then leverages planner-selected tools to provide linguistic feedback, enabling the planner to refine future plans. During *tool-based evaluation*, TG-MCTS employs these tools to assess revision quality and constraint compliance, steering the algorithm towards an optimal plan.

Existing text revision datasets primarily focus on single-task or sentence-level revisions, making them unsuitable for paragraph-level revisions with constrained instructions. Therefore, we introduce a dataset named **CORD**, which incorporates complex, verifiable, and valid text revision constraints into paragraph-level text inputs. We also propose evaluation metrics to assess both constraint adherence and revision quality. We evaluate CRAFT's performance using the CORD dataset. The contributions of our paper are as follows:

- We introduce CTR, a novel and complex text revision task that closely reflects real-world scenarios, along with the CORD dataset.
- To the best of our knowledge, this is the first study to formulate text revision as an iterative planning and searching problem. Experimental results demonstrate that CRAFT significantly outperforms baseline approaches.
- CRAFT exhibits strong adaptability across diverse text revision tasks, such as LaTeX revision, consistently achieving superior performance.

2 Related Work

Text Revision Systems. Existing text revision systems (Raheja et al., 2023; Cao et al., 2025) generate suggestions based on user instructions but are primarily designed for single-task or single-sentence revisions. However, a significant gap remains between these systems and real-world applications. CRAFT bridges this gap by providing suggestions for paragraph-level inputs while adhering to diverse constraints and scenarios.

Constrained Text Generation. Our study on CTR is closely related to CTG for LLM, which involves generating text under specific constraints. Prior research (Zhou et al., 2023; Jiang et al., 2024) indicates that LLMs often struggle to comply with complex constraints. Existing work (Sun et al., 2024; Xu et al., 2024) mainly uses SFT and preference optimization with labeled data to improve LLM's constraint adherence. Chaffin et al. (2022) explored the use of traditional MCTS for CTG but focused only on simple constraints, such as binary sentiment constraints. In contrast, our method offers greater flexibility in handling complex constraints through iterative planning and searching. Appendix A shows more related work on writing-related agents and MCTS.

3 CORD

CORD is constructed in two steps: data selection and constrained instruction generation.

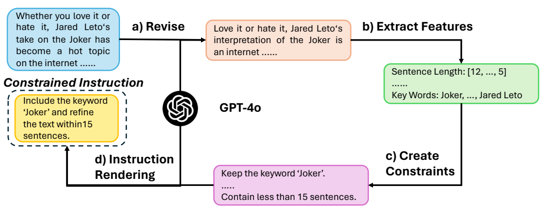

Whether you love it or hate it, Jared Leto's take on the Joker has become a hot topic on the internet ......
a) Revise
Love it or hate it, Jared Leto's interpretation of the Joker is an internet ......
b) Extract Features
Sentence Length: [12, ..., 5] ...... Key Words: Joker, ..., Jared Leto
Constrained Instruction
Include the keyword 'Joker' and refine the text within15 sentences.
GPT-4o
c) Create Constraints
d) Instruction Rendering
Keep the keyword 'Joker'. ..... Contain less than 15 sentences.

Figure 2: Constrained instruction generation pipeline.

Data Selection. To ensure broad domain coverage and build a robust evaluation framework, we build on prior research (Que et al., 2024; Tang et al., 2022; Mita et al., 2024; Ladhak et al., 2020) by selecting texts from academic papers, WikiHow articles, and human-written stories. We randomly sample 500 texts, each between 350 and 1,000 words, as input text.

Constrained Instruction Generation. Designing valid text revision constraints requires careful consideration, as some may be inapplicable or conflicting. To address this, we propose an instruction-generation pipeline (Fig. 2). Specifically, given a text, we first employ GPT-4o to *revise* the selected text, producing revised versions. We then *extract* sentence-level and word-level features from these revisions and *construct* constraint-based instructions through program templates. To increase instruction complexity (Pham et al., 2024), we randomly select M features to generate M single-constraint instructions. Finally, following Yao et al. (2024) we *render* these instructions with GPT-4o to improve fluency and diversity, yielding a text revision instruction with M constraints.

We categorize the 500 texts into five domains (C0–C4), each containing 100 texts. C0 texts are paired with unconstrained instructions (*i.e.,* standard text revision instructions), whereas C1–C4 texts are paired with instructions containing one to four constraints, respectively. Additional details are provided in Appendix C.

4 Preliminary Analysis

During CTR, humans read the full text to understand its context, and iteratively plan improvements (Flower and Hayes, 1980; Du et al., 2022). Inspired by this, we explore whether LLMs can similarly benefit from this behavior. Specifically, we address the following research questions:

- **RQ-1**: Can structured planning enhance LLM-generated revisions?
- **RQ-2**: Does the quality of LLM-generated revisions improve through iterative revision?

	PPL↓	SOME↑	BARTScore↑
w/o Plan	34.58	88.91	-2.46
w/ GPT-4o Plan	23.64	91.67	-1.92
w/ Human Plan	**21.31**	**93.28**	**-1.49**

Table 1: Revised text quality under three conditions: without plans (**w/o Plan**), with GPT-4o-generated plans (**w/ GPT-4o Plan**), and with human-labeled plans (**w/ Human Plan**). SOME is reported in %.

Setup. We analyze LLMs' performance in CTR using TETRA (Mita et al., 2024), a dataset containing human-labeled text revision plans for paragraph-level text. Since TETRA lacks explicit instructions, we generate C0–C4 constrained instructions following the method in §3. We use accuracy to measure constraint adherence. Following Kim and Kim (2024); Shao et al. (2024), we assess revision quality from fluency, coherence, and grammaticality perspectives. Consistent with prior research (Yuan et al., 2021; Goto et al., 2024; Qorib and Ng, 2023), we measure fluency with perplexity (PPL) through GPT-2 Large (Radford et al., 2019), coherence with BARTScore (Yuan et al., 2021), and grammaticality with SOME (Yoshimura et al., 2020).

	C1	C2	C3	C4
w/o Plan	68.00	61.00	53.66	46.50
w/ GPT-4o Plan	71.00	67.00	61.00	54.00
Gain	**+3.00**	**+6.00**	**+7.34**	**+7.50**

Table 2: Constraint adherence accuracy (%) under different constraints for two settings: without plans (**w/o Plan**) and with GPT-4o-generated plans (**w/ GPT-4o Plan**). **Gain**: the performance gain with the plan.

Structured Planning. We compare two settings: (a) using human-labeled revision plans and (b) using GPT-4o-generated plans. In the first setting, GPT-4o revises text based on human-labeled plans from TETRA. In the second, GPT-4o first generates a revision plan and then uses it to revise the text. Table 1 indicates that planning significantly enhances fluency (lower PPL), coherence (higher BARTScore), and grammaticality (higher SOME), with human-labeled plans yielding greater improvements. Furthermore, Table 2 shows that planning enhances constraint adherence. The improvement increases as the number of constraints grows. Appendix D.1 shows more implementation details.

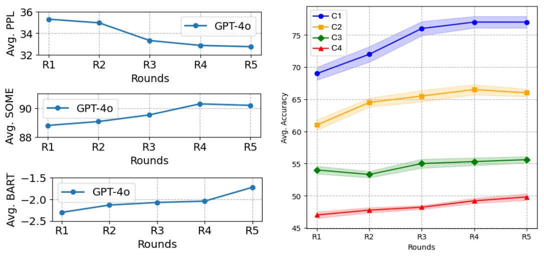


Figure 3: **Left:** Average PPL, SOME, and BARTScore for revised text across five revision rounds (R1–R5). **Right:** Average accuracy for different revision rounds.

Iterative Revision. Fig. 3 shows that iterative revisions enhance text quality (fluency, coherence, and grammaticality) and adherence to constraints. The results indicate that LLMs' CTR ability benefits from **structured planning (RQ-1)** and **iterative revision (RQ-2)**.

5 CRAFT

Building on the features identified in §4, we introduce CRAFT, which iteratively refines text through

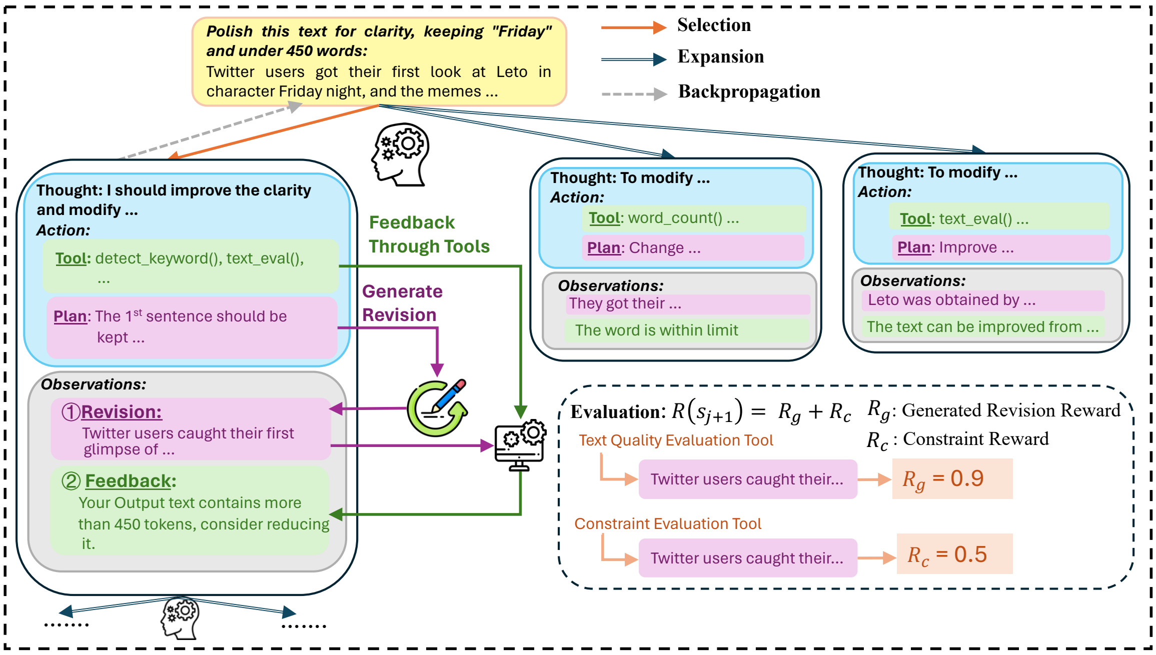

Polish this text for clarity, keeping "Friday" and under 450 words:
Twitter users got their first look at Leto in character Friday night, and the memes ...
Selection
Expansion
Backpropagation
Thought: I should improve the clarity and modify ...
Action:
Tool: detect_keyword(), text_eval(), ...
Plan: The 1st sentence should be kept ...
Observations:
①Revision:
Twitter users caught their first glimpse of ...
② Feedback:
Your Output text contains more than 450 tokens, consider reducing it.
Feedback Through Tools
Generate Revision
Thought: To modify ...
Action:
Tool: word_count() ...
Plan: Change ...
Observations:
They got their ...
The word is within limit
Thought: To modify ...
Action:
Tool: text_eval() ...
Plan: Improve ...
Observations:
Leto was obtained by ...
The text can be improved from ...
Evaluation: $R(s_{j+1}) = R_g + R_c$
R_g: Generated Revision Reward
R_c : Constraint Reward
Text Quality Evaluation Tool
Twitter users caught their...
$R_g = 0.9$
Constraint Evaluation Tool
Twitter users caught their...
$R_c = 0.5$

Figure 4: Illustration of the search process.

planning and **searching**, leveraging both the planner (Fig. 5) and the search algorithm (Fig. 4). The planner generates revision plans and tool usages, while the search algorithm utilizes tool evaluations and feedback to optimize the revision plans.

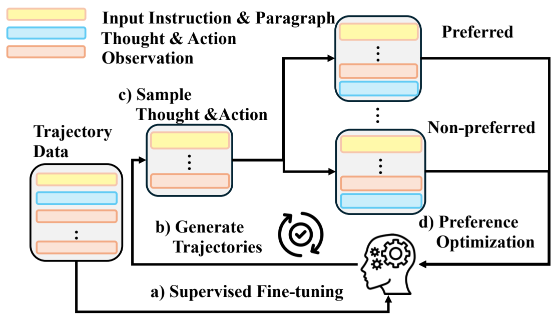

Input Instruction & Paragraph
Thought & Action
Observation
Preferred
Non-preferred
c) Sample Thought &Action
Trajectory Data
b) Generate Trajectories
d) Preference Optimization
a) Supervised Fine-tuning

Figure 5: Illustration of the planner construction process.

5.1 Planner

Table 1 shows that LLMs perform better with human-labeled plans. However, these plans are sparse, and LLMs struggle to interpret numerical constraints (Chen et al., 2024). Therefore, we employ GPT-4o to synthesize planning trajectories via ICL (§5.1.1). This synthesized trajectory is then used to fine-tune LLMs via SFT and iterative self-training alignment (§5.1.2). Consequently, the resulting planner, π_p, generates human-like revision plans with precise tool usage.

5.1.1 Trajectory Generation

Data Source. We use CNN-DailyNews (Nallapati et al., 2016) as the raw data ($\mathcal{D}_r$) to generate trajectories. Following the method in §3, we select texts containing 350 to 1,000 words and generate constrained instructions for each input.

Trajectory Format. We utilize the ReAct framework (Yao et al., 2023) to generate trajectories by feeding GPT-4o with the input text and its corresponding constrained instruction. The generation process follows an iterative three-step approach: (a) analyzing the input text (*observation*), (b) identifying constraints and areas requiring revision (*thought*), and (c) formulating tool usage and revision plans (*action*). The LLM then generates a revised output based on this plan. The revised output, along with its feedback from the tools, serves as the new *observation* for the next iteration.

This process iterates until a complete trajectory is generated, either upon reaching the maximum iteration limit or when further revisions fail to improve the output quality. To generate human-like revision plans, we randomly select an example from TETRA, apply the constraints according to §3, and augment its revision plan accordingly. This modified example serves as the in-context example for GPT-4o. Each trajectory comprises constrained instructions, input text, intermediate steps (*observations, thoughts, actions*), and the final revised text. Samples with incorrect tool usage are discarded to ensure the trajectory's quality. Ultimately, we generate 2k synthetic trajectories, denoted as $\mathcal{D}_s$. Appendix D.2 shows further details.

5.1.2 Planner Construction

SFT. We construct the initial planner, π_0, by fine-tuning LLM on $\mathcal{D}_s$ with SFT. The synthetic trajectory for each input is represented as $\langle o_0, t_1, a_1, o_1, \ldots, t_m, a_m, o_m \rangle$, where t_i, a_i, and o_i denote the thought, action, and observation at step i, respectively. Here, o_0 corresponds to the initial observation, which includes the instruction and input text, and o_m represents the final observation, containing the revised text and its corresponding tool feedback. At each step, the planner generates a thought and an action based on the historical trajectory $\mathcal{H}_{i-1} = \langle o_0, t_1, a_1, \ldots, o_{i-1} \rangle$. During SFT, we compute the cross-entropy loss only for t_i and a_i, masking o_i: $\mathcal{L}_{CE} = -\log \sum_{i=1}^{n} \pi_0(t_i, a_i | \mathcal{H}_{i-1})$.

Iterative Self-Training Alignment. SFT equips π_0 with reasoning capabilities (*i.e.,* tool usage, text-revision planning). However, SFT alone may result in suboptimal tool-usage performance. Therefore, we introduce an iterative self-training alignment method (Algorithm 1), which improves the planner by emphasizing its high-quality outputs while mitigating the low-quality ones.

Algorithm 1: Self-Training Alignment

```
Input: $\mathcal{D}_r$, Initial planner $\pi_0$ ($\pi_p$, when $p = 0$),
    trajectory step count $i$.
Output: The final planner $\pi_p$.
1: while $\pi_p$ keeps improving do
2:     Sample data from $\mathcal{D}_r$, and generate
       constraints following the method in §3.
3:     Generate trajectories ($\mathcal{H}_i$) up to $i$-th step.
4:     Sample responses: $(a_{i+1}, t_{i+1}) = \pi_p(\mathcal{H}_i)$.
5:     Score $a_{i+1}$ via Eq. 1 to build $\mathcal{D}_p$.
6:     Optimize $\pi_p$ on $\mathcal{D}_p$ with Eq. 2.
7: end while
```

At each iteration, we randomly sample 2k texts from $\mathcal{D}_r$ and generate constrained instructions to form text-instruction pairs following §3. For each pair, the current planner π_p at iteration p generates trajectories of up to i steps, producing a historical trajectory $\mathcal{H}_i$. Subsequently, multiple thought-action pairs, t_{i+1} and a_{i+1}, are sampled for each $\mathcal{H}_i$ using sampling-based decoding. Since actions with correct tool usage and better revision plans are preferred, we score them using $S_a(\cdot)$:

$$S_a(a_{i+1}) = \lambda_v \cdot S_v + \lambda_r \cdot S_r + \lambda_c \cdot S_c, \tag{1}$$

where S_v measures the tool usage quality, while both S_r and S_c measure the overall quality of the revised text[2]. Specifically, S_r measures the revision quality (*e.g.,* fluency and coherence), whereas S_c evaluates its adherence to constraints. λ_v, λ_r, and λ_c are the weights assigned to these metrics. Implementation details are in Appendix D.4.

Among these actions, the one with the highest $S_a(\cdot)$ score is selected as the preferred action, with its corresponding thought forming the preferred response, w_{i+1}. Conversely, the action with the lowest score is identified as the non-preferred action, with its associated thought forming the non-preferred response, l_{i+1}. This process generates a preference dataset, $\mathcal{D}_p$. We then apply SimPO (Meng et al., 2024), coupled with the cross-entropy loss computed on w_{i+1}, to optimize π_p with $\mathcal{L}_P$:

$$\begin{aligned} \mathcal{L}_P = {} & \mathcal{L}_{SimPO} - \log \pi_p(w_{i+1}|\mathcal{H}_i) \\ = {} & -\log \sigma\left(\frac{\beta \log \pi_p(w_{i+1}|\mathcal{H}_i)}{|w_{i+1}|} - \frac{\beta \log \pi_p(l_{i+1}|\mathcal{H}_i)}{|l_{i+1}|} - \gamma\right) \\ & -\log \pi_p(w_{i+1}|\mathcal{H}_i), \end{aligned} \tag{2}$$

where β and γ are hyper-parameters for SimPO. This process iterates until the tool usage and revision plans generated by π_p show no further measurable improvement.

5.2 Search

The iterative nature of text revision makes future plans highly dependent on previous revisions and feedback. This motivates us to use search algorithms, such as MCTS, to identify optimal revision plans. Furthermore, to enhance the adaptability of the reviser across various constraints and scenarios, we propose *Tool-Guided Monte Carlo Tree Search* (TG-MCTS), a novel CTR algorithm that guides the revision process according to different constraints. TG-MCTS integrates feedback and verification from self-selected tools while leveraging the reflection capabilities of LLMs to promptly correct errors and optimize the search process.

Within our framework, a tree is constructed using the reviser π_θ and the planner π_p. Each node at the j-th level is represented as $s_j = \{o_j, \mathcal{H}_j, N(s_j), V(s_j)\}$, where o_j includes the revised text y_j and tools' feedback, $\mathcal{H}_j$ represents the historical trajectory to the current node, and $N(s_j)$ and $V(s_j)$ denote the node's visit count and value

[2]Generated by feeding a_{i+1} into a vanilla LLM.

score, respectively. The root node, $s_0 = \{o_0\}$, contains the initial text and instructions (*i.e.,* starting observation). The TG-MCTS algorithm iteratively performs four operations: selection, tool-guided expansion, tool-based evaluation, and backpropagation.

Selection. The selection process identifies a node s_j for expansion based on the Upper Confidence Bounds applied to Trees (UCT) score (Kocsis and Szepesvári, 2006), defined as:

$$UCT(s_j) = V(s_j) + \alpha\sqrt{\frac{\ln N(p)}{N(s_j)}}, \qquad (3)$$

where p denotes the parent node of s_j, and the hyper-parameter α balances between exploitation (*i.e.,* the node value $V(s_j)$, which corresponds to the expected reward of s_j) and exploration (*i.e.,* the visit count $N(s_j)$).

Tool-Guided Expansion. The expansion phase consists of two key steps: **Revise** and **Feedback**. In the **Revise** step, the selected node s_j expands by generating a set of actions, a_{j+1}, sampled from the planner. These actions are then processed by π_θ to generate revised text: $y_{j+1} = \pi_\theta(a_{j+1}, y_j)$. In the subsequent **Feedback** step, the feedback is generated through pre-defined tools (Appendix E.2) and consists of two components: revision feedback and constraint feedback. Revision feedback is obtained using the *text-quality tool*, which prompts π_θ with the prompts shown in Appendix E.8. Constraint feedback is derived using the *condition-checking tools*. Leveraging LLMs' reflective capabilities, the planner incorporates this feedback to refine text revision plans. This enables TG-MCTS to promptly improve revision quality throughout the search.

Tool-Based Evaluation. During the evaluation, the selected tools estimate the expected reward $R(s_{j+1})$ for the new node s_{j+1}. This reward consists of two components: the generated revision reward (R_g) and the constraint reward (R_c). Specifically, the *text-quality tool* calculates R_g as the arithmetic mean of the normalized PPL, BARTScore, and SOME scores. Meanwhile, the *condition-checking tools* compute R_c. Finally, the overall reward is defined as: $R(s_{j+1}) = R_g + R_c$. Additional details can be found in Appendix D.5.

Backpropagation. After obtaining the reward for the new node s_{j+1}, TG-MCTS updates the values and visit counts of all nodes along the path from the root node to its ancestor nodes s_k $(0 \le k \le j)$ with the following equations:

$$N_{\text{new}}(s_k) = N_{\text{old}}(s_k) + 1, \qquad (4)$$

$$V_{\text{new}}(s_k) = \frac{V_{\text{old}}(s_k)N_{\text{old}}(s_k) + R(s_{j+1})}{N_{\text{new}}(s_k)}, \qquad (5)$$

where $N_{\text{old}}(s_k)$ and $V_{\text{old}}(s_k)$ denote the visit count and value of node s_k, respectively, prior to backpropagation.

6 Experiments

6.1 Setup

Dataset and Metrics. We evaluate CRAFT on the CORD dataset, measuring its performance in terms of constraint adherence and revision quality. Constraint adherence is assessed using accuracy. Revision quality is evaluated from the fluency, grammaticality, and coherence aspects, measured by PPL, SOME, and BARTScore, respectively.

Models. Two versions of CRAFT are developed: (a) **CRAFT-3.1**, which employs Llama-3.1-8B-Instruct (Dubey et al., 2024) as the reviser, and (b) **CRAFT-4o**, which uses GPT-4o as the reviser. In both versions, the planner (π_p) uses Llama-3.1-8B-Instruct as the base model. π_p is trained using trajectories generated for instructions containing up to three constraints (C0–C3), while instructions with four constraints (C4) are reserved for evaluating its generalization to unseen domains. During iterative self-training alignment, $\mathcal{H}_i$ is generated with a maximum of five steps. For CTR, we define a total of 8 tools, encompassing text quality evaluation tool, keyword detection tool, and various condition-checking tools. Detailed description of these tools is provided in Appendix E.2.

Baselines. We compare CRAFT against the following baselines based on GPT-4o and Llama-3.1-8B-Instruct: (a) **Direct:** The LLMs are directly prompted with instructions and text from CORD; (b) **CoT:** The LLMs are prompted to generate intermediate reasoning steps before providing the final answer. Specifically, we employ zero-shot Chain-of-Thought (CoT) (Wei et al., 2022); (c) **Plan:** The LLMs first receive a human-labeled plan as an in-context example (as described in §5.1.1) to generate text revision plans. LLMs then use these plans to revise the text; (d) **Iter:** The LLMs generate an initial response and iteratively revise it over multiple rounds. According to Fig. 3, we conduct 5 rounds of text revision.

System	Constraint Adherence (Acc.↑)				Text Quality		
	C1	C2	C3	C4	PPL↓	SOME↑	BARTScore↑
Evol-Ins	57.00	53.00	51.33	42.00	35.44	87.43	-1.92
Conifer	51.00	59.00	52.00	44.25	42.51	87.55	-1.85
DeepSeek-R1	81.00	70.00	66.66	58.00	40.95	87.60	-2.08
o1-preview	80.00	70.50	67.00	57.50	40.84	84.46	-2.07
				Llama 3.1 8B Instruct			
Direct	58.00	59.50	50.33	42.25	32.61	87.55	-3.33
CoT	60.00	57.50	51.00	46.00	32.11	88.09	-4.38
Plan	62.00	62.50	54.66	46.25	29.52	85.75	-3.26
Iter	65.00	63.50	57.33	48.25	28.95	87.99	-3.38
CRAFT-3.1	83.00	80.00	80.00	72.75	**28.33**	**88.80**	-1.90
				GPT-4o			
Direct	69.00	61.50	54.33	47.00	50.69	87.90	-2.02
CoT	68.00	63.00	55.66	48.75	48.63	87.74	-1.96
Plan	72.00	66.50	60.00	53.75	43.25	87.87	-1.96
Iter	77.00	67.50	62.33	54.75	45.41	87.53	-2.03
CRAFT-4o	**85.00**	**83.00**	**82.66**	**76.50**	34.74	86.85	**-1.82**

Table 3: Performance on CORD across C1-C4 domains. **Text Quality** denotes the averaged text quality across C1-C4. **Acc.** denotes accuracy. Both Acc. and SOME are shown in %. The best results are **bolded**, and the second-best results are underlined across all domains.

Additionally, we compare CRAFT's performance against the state-of-the-art (SOTA) text revision system, CoEDIT-Composite (CoEDIT-C) (Raheja et al., 2023), in the C0 domain. We also compare CRAFT with SOTA methods for CTG, including Evol-Ins (Xu et al., 2024) and Conifer (Sun et al., 2024), across the C1 to C4 domains. We report the average score over three runs. For C0 to C5, we also include comparison with reasoning LLMs DeepSeek-R1 and o1-preview, using the **Iter** method, which yields the best performance. Additional implementation details are provided in Appendix D.6.

6.2 Results

Table 3 highlights significant improvements in constraint adherence for both CRAFT-3.1 and CRAFT-4o compared to their respective baselines, with an average accuracy increase of over 16% across all domains. As the number of constraints increases from C1 to C4, both systems demonstrate progressively greater performance gains over baseline approaches. Notably, CRAFT-4o consistently achieves the highest accuracy across all test cases, with an average accuracy of 81.8%.

Moreover, Tables 3 and 4 show that CRAFT-3.1 and CRAFT-4o produce superior text revisions compared to the baselines in most scenarios, in fluency, grammaticality, and coherence aspects. Our statistical significance tests confirm that both models generate high-quality revisions, surpassing previous SOTA text revision and CTG systems in both constraint adherence and revision quality.

System	C0		
	PPL↓	SOME↑	BARTScore↑
CoEDIT-C	38.82	87.32	-2.16
DeepSeek-R1	35.97	88.10	-1.84
o1-preview	34.98	88.20	-1.92
	Llama 3.1 8B Instruct		
Direct	29.69	83.61	-4.97
CoT	27.38	84.58	-4.77
Plan	27.31	84.18	-4.58
Iter	26.55	84.21	-4.52
CRAFT-3.1	**25.82**	**88.96**	-1.92
	GPT-4o		
Direct	35.92	87.61	-2.18
CoT	36.16	88.62	-2.21
Plan	35.24	88.14	-1.87
Iter	34.74	88.21	-1.89
CRAFT-4o	33.07	88.80	**-1.76**

Table 4: Performance on CORD C0 domain. SOME is shown in %. The best and second-best results are highlighted in **bold** and underline, respectively.

SOTA CTG systems, Evol-Ins and Conifer, are built by training Llama-3.1-8B-Instruct with labeled constrained instruction data. However, these models do not exhibit significant advantages over their base model on CRAFT, likely due to the domain gap between their training data and CORD. In contrast, CRAFT-3.1 achieves greater improvements by leveraging a planner and reviser (*i.e.,* a vanilla LLM).

7 Analysis

In this section, we refer to the best-performing GPT-4o baseline, GPT-4o (Iter), as GPT-4o for brevity.

	CRAFT-4o	GPT-4o
F (↑)	**4.93**	4.87
C (↑)	**4.82**	4.67
G (↓)	**0.02**	0.06

	# Cases
F	67
C	72
G	85

Table 5: LLM-as-a-Judge using GPT-4. **Left**: Average scores assigned by GPT-4. **Right**: Number of cases (**# Cases**) where CRAFT-4o outperformes GPT-4o.

LLM-as-a-Judge. Text revision is subjective, and traditional metrics may fail to capture quality accurately. While human evaluations provide insights, they are biased and irreproducible. Prior research (Sottana et al., 2023; Zhou et al., 2024) shows that GPT-4 closely aligns with human judgments on fluency, coherence, and grammaticality.

We compare the revision quality of CRAFT-4o and GPT-4o by randomly selecting 100 outputs from each system. Following Sottana et al. (2023), we score the fluency (F), grammaticality (G), and coherence (C) of the revision with GPT-4. As shown in Table 5 (Left), CRAFT-4o outperforms GPT-4o on these aspects. Additionally, following Zhou et al. (2024), we conduct a pairwise comparison using GPT-4, using the prompt in Table 15. Table 5 (Right) confirms that CRAFT-4o produces superior revisions more often than GPT-4o. Details of the experimental setup are in Appendix D.7.

System	**C0**		
	PPL↓	**SOME↑**	**BARTScore↑**
CRAFT-4o	**33.07**	**88.80**	-1.76
w/o Plan	34.93	88.16	-1.91
w/o Feedback	34.21	88.24	-1.88
w/o R_g	33.95	88.56	-1.82
w/o R_c	33.09	88.78	**-1.74**

Table 6: Revision quality on the CORD C0 domain.

Ablation Study. We analyze CRAFT-4o by removing several key elements: 1) *w/o Plan*: removes the planner and rely solely on iterative text revision; 2) *w/o Feedback*: removes the tool feedback in TG-MCTS; 3) *w/o R_g*: omits the generated revision reward R_g; 4) *w/o R_c*: omits the constraint reward R_c. For revision quality analysis, we focus on the C0 domain (Table 6), while constraint adherence is analyzed using the C1-C4 domains (Table 7). Tables 6 and 7 confirm that both planning and feedback play a crucial role in providing revisions with better text quality and adherence to constraints. Specifically, R_g contributes to improving text quality, while R_c strengthens constraint adherence.

	C1	C2	C3	C4
CRAFT-4o	**85.00**	**83.00**	**82.66**	**76.50**
w/o Plan	76.00	65.50	60.66	54.25
w/o Feedback	79.00	69.00	62.00	56.00
w/o R_g	84.00	82.50	81.66	75.25
w/o R_c	81.00	73.00	68.33	62.75

Table 7: Constraint adherence accuracy on CORD across C1 to C4 domains.

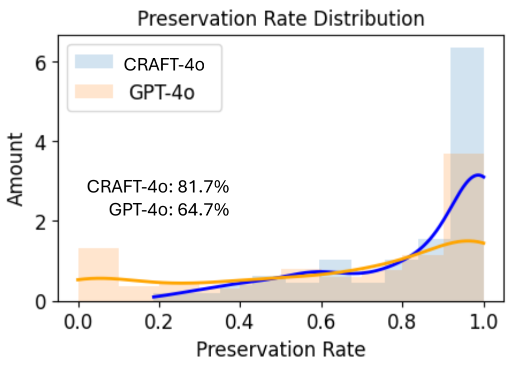


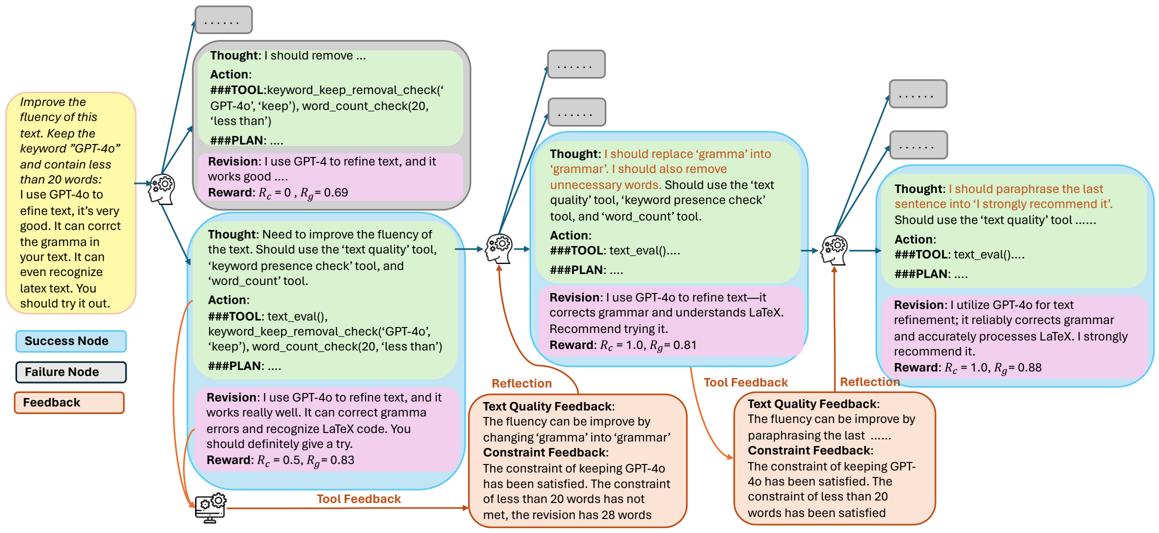
Figure 6: The LaTeX keyword preservation rate distribution.

	AvgCE. ↓	**Text Quality**		
		PPL ↓	SOME ↑	BARTScore ↑
GPT-4o	0.24	48.72	85.37	-1.92
CRAFT-4o	**0.06**	**35.65**	**88.21**	**-1.61**

Table 8: Revised text generated by CRAFT-4o and GPT-4o. **AvgCE.**: the average compilation error. **Text Quality**: the quality of the revision after compilation.

Adaptability. We evaluate CRAFT-4o's adaptability for LaTeX revision, a more complex task than plain text revision due to LaTeX-specific keywords. To evaluate its performance, we randomly select 100 LaTeX paragraphs from papers with LaTeX source files in the TETRA dataset. More details are shown in Appendix D.8. To facilitate LaTeX keyword detection, we integrate a LaTeX keyword detection program into the *keyword detection tool* (Appendix E.2). Fig. 6 shows CRAFT-4o achieves a higher LaTeX keyword preservation rate than GPT-4o. Table 8 shows that CRAFT-4o produces fewer compilation errors and higher-quality LaTeX revisions compared to GPT-4o. This suggests that CRAFT-4o can be effectively adapted to other scenarios with minor modifications.

Tool Usage. Tool usage plays a crucial role in



......
Thought: I should remove ...
Action:
###TOOL:keyword_keep_removal_check(‘GPT-4o’, ‘keep’), word_count_check(20, ‘less than’)
###PLAN: ....
Revision: I use GPT-4 to refine text, and it works good ....
Reward: R_c = 0 , R_g= 0.69
Improve the fluency of this text. Keep the keyword ”GPT-4o” and contain less than 20 words:
I use GPT-4o to efine text, it’s very good. It can corrct the gramma in your text. It can even recognize latex text. You should try it out.
Success Node
Failure Node
Feedback
Thought: Need to improve the fluency of the text. Should use the ‘text quality’ tool, ‘keyword presence check’ tool, and ‘word_count’ tool.
Action:
###TOOL: text_eval(), keyword_keep_removal_check(‘GPT-4o’, ‘keep’), word_count_check(20, ‘less than’)
###PLAN: ....
Revision: I use GPT-4o to refine text, and it works really well. It can correct gramma errors and recognize LaTeX code. You should definitely give a try.
Reward: R_c = 0.5, R_g= 0.83
Tool Feedback
Reflection
Text Quality Feedback:
The fluency can be improve by changing ‘gramma’ into ‘grammar’
Constraint Feedback:
The constraint of keeping GPT-4o has been satisfied. The constraint of less than 20 words has not met, the revision has 28 words
......
......
Thought: I should replace ‘gramma’ into ‘grammar’. I should also remove unnecessary words. Should use the ‘text quality’ tool, ‘keyword presence check’ tool, and ‘word_count’ tool.
Action:
###TOOL: text_eval()....
###PLAN: ....
Revision: I use GPT-4o to refine text—it corrects grammar and understands LaTeX. Recommend trying it.
Reward: R_c = 1.0, R_g= 0.81
Tool Feedback
Reflection
Text Quality Feedback:
The fluency can be improve by paraphrasing the last ......
Constraint Feedback:
The constraint of keeping GPT-4o has been satisfied. The constraint of less than 20 words has been satisfied
......
......
Thought: I should paraphrase the last sentence into ‘I strongly recommend it’. Should use the ‘text quality’ tool ......
Action:
###TOOL: text_eval()....
###PLAN: ....
Revision: I utilize GPT-4o for text refinement; it reliably corrects grammar and accurately processes LaTeX. I strongly recommend it.
Reward: R_c = 1.0, R_g= 0.88

Figure 7: A qualitative example illustrating CRAFT’s constrained text revision process.

providing feedback and evaluation. We assess the planner’s (π_p) tool usage quality using the F_1 score. Fig. 8 (Left) shows that π_p significantly outperforms both its base model and GPT-4o. Notably, π_p achieves strong performance on C4, a domain excluded from training, demonstrating π_p’s ability to generalize to unseen scenarios. Furthermore, Fig. 8 (Right) highlights a consistent improvement in tool usage across iterations, underscoring the effectiveness of the iterative self-training alignment method. Our case study in Fig. 10 illustrates that, unlike GPT-4o which frequently generates redundant or incorrect tool calls and misinterprets mathematical symbols, π_p effectively mitigates these errors.

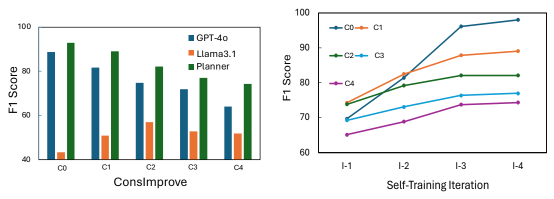


Figure 8: F_1 score (in %) for tool usage quality. **Left:** Tools usage generated by GPT-4o, Llama-3.1-8B-Instruct, and the planner. **Right:** Tool usage quality across four iterations (I-1 to I-4).

Case Study. Fig. 7 presents a qualitative example of the CRAFT workflow. CRAFT begins by using a planner to generate a sequence of thoughts and actions based on the given instruction and input text. The reviser then produces a draft revision according to the plan specified in the actions. These actions are evaluated by external tools identified by the planner, which assign rewards and provide feedback. CRAFT then applies TG-MCTS to select the optimal revision. Tool feedback serves as a reflection signal, enabling the planner to refine its future strategies. In the example in Fig. 7, the tools highlight ways to improve fluency and suggest reducing unnecessary words. This reflective process helps the planner generate more effective plans in subsequent iterations. Repeating this loop results in progressively higher-quality revisions that better satisfy the specified constraints.

8 Conclusion

This paper introduces the CTR task, which better reflects real-world text revision scenarios. To support this task, we developed the CORD dataset, a comprehensive benchmark for evaluating systems on constrained text revisions. Furthermore, we conceptualize CTR as an iterative planning and searching problem and propose CRAFT to address the complexities of paragraph-level text revisions under diverse constraints. Experimental results demonstrate that CRAFT consistently outperforms baseline approaches and exhibits robustness across various text revision scenarios.

9 Limitations

Despite the comprehensive analysis and experimental results presented in this work, our study is limited by the computational cost of the tree-based search method. Additionally, while our approach does not rely on specific features of a particular text environment, its effectiveness has only been evaluated on English plain text and LaTeX revision.

Acknowledgments

This research is supported by the National Research Foundation Singapore under its AI Singapore Programme (Award Number: AISG3-RP-2022-030). We thank the anonymous reviewers for their insightful comments. We are also grateful to Junyi Li for his helpful discussions.

References

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*.

Hannan Cao, Wenmian Yang, and Hwee Tou Ng. 2021. Grammatical error correction with contrastive learning in low error density domains. In *Findings of EMNLP*.

Hannan Cao, Hai Ye, and Hwee Tou Ng. 2025. Rationalize and align: Enhancing writing assistance with rationale via self-training for improved alignment. In *Findings of ACL*.

Hannan Cao, Liping Yuan, Yuchen Zhang, and Hwee Tou Ng. 2023. Unsupervised grammatical error correction rivaling supervised methods. In *Proc. of EMNLP*.

Antoine Chaffin, Vincent Claveau, and Ewa Kijak. 2022. PPL-MCTS: Constrained textual generation through discriminator-guided MCTS decoding. In *Proc. of NAACL*, pages 2953–2967.

Yihan Chen, Benfeng Xu, Quan Wang, Yi Liu, and Zhendong Mao. 2024. Benchmarking large language models on controllable generation under diversified instructions. In *Proc. of AAAI*, pages 17808–17816.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *Preprint*, arXiv:2210.11416.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. Ultrafeedback: Boosting language models with scaled ai feedback. *Preprint*, arXiv:2310.01377.

Wanyu Du, Vipul Raheja, Dhruv Kumar, Zae Myung Kim, Melissa Lopez, and Dongyeop Kang. 2022. Understanding iterative revision from human-written text. In *Proc. of ACL*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Linda Flower and John R Hayes. 1980. The cognition of discovery: Defining a rhetorical problem. *College Composition & Communication*, 31(1):21–32.

Takumi Goto, Justin Vasselli, and Taro Watanabe. 2024. Improving explainability of sentence-level metrics via edit-level attribution for grammatical error correction. *Preprint*, arXiv:2412.13110.

Fantine Huot, Reinald Kim Amplayo, Jennimaria Palomaki, Alice Shoshana Jakobovits, Elizabeth Clark, and Mirella Lapata. 2025. Agents' room: Narrative generation through multi-step collaboration. In *Proc. of ICLR*.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2024. Follow-Bench: A multi-level fine-grained constraints following benchmark for large language models. In *Proc. of ACL*.

Seungyoon Kim and Seungone Kim. 2024. Can language models evaluate human written text? case study on korean student writing for education. *Preprint*, arXiv:2407.17022.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *Proc. of ECML*.

Faisal Ladhak, Esin Durmus, Claire Cardie, and Kathleen McKeown. 2020. WikiLingua: A new benchmark dataset for cross-lingual abstractive summarization. In *Proc. of EMNLP Findings*.

Nayeon Lee, Wei Ping, Peng Xu, Mostofa Patwary, Pascale N Fung, Mohammad Shoeybi, and Bryan Catanzaro. 2022. Factuality enhanced language models for open-ended text generation. In *Proc. of NeurIPS*.

Huang Lei, Jiaming Guo, Guanhua He, Xishan Zhang, Rui Zhang, Shaohui Peng, Shaoli Liu, and Tianshi Chen. 2024. Ex3: Automatic novel writing by extracting, excelsior and expanding. In *Proc. of ACL*.

Xun Liang, Hanyu Wang, Yezhaohui Wang, Shichao Song, Jiawei Yang, Simin Niu, Jie Hu, Dan Liu, Shunyu Yao, Feiyu Xiong, et al. 2024a. Controllable text generation for large language models: A survey. *arXiv preprint arXiv:2408.12599*.

Yi Liang, You Wu, Honglei Zhuang, Li Chen, Jiaming Shen, Yiling Jia, Zhen Qin, Sumit Sanghai, Xuanhui Wang, Carl Yang, et al. 2024b. Integrating planning into single-turn long-form text generation. *arXiv preprint arXiv:2410.06203*.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. SimPO: Simple preference optimization with a reference-free reward. In *Prof. of NeurIPS*.

Masato Mita, Keisuke Sakaguchi, Masato Hagiwara, Tomoya Mizumoto, Jun Suzuki, and Kentaro Inui. 2024. Towards automated document revision: Grammatical error correction, fluency edits, and beyond. In *Proc. of BEA*.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Guĺçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proc. of CoNLL*.

Chau Minh Pham, Simeng Sun, and Mohit Iyyer. 2024. Suri: Multi-constraint instruction following in long-form text generation. In *Proc. of EMNLP Findings*.

Muhammad Reza Qorib and Hwee Tou Ng. 2023. System combination via quality estimation for grammatical error correction. In *Proc. of EMNLP*.

Haoran Que, Feiyu Duan, Liqun He, Yutao Mou, Wangchunshu Zhou, Jiaheng Liu, Wenge Rong, Zekun Moore Wang, Jian Yang, Ge Zhang, et al. 2024. Hellobench: Evaluating long text generation capabilities of large language models. *arXiv preprint arXiv:2409.16191*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*.

Vipul Raheja, Dhruv Kumar, Ryan Koo, and Dongyeop Kang. 2023. CoEdIT: Text editing by task-specific instruction tuning. In *Findings of EMNLP*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Proc. of NeurIPS*.

Yijia Shao, Yucheng Jiang, Theodore Kanell, Peter Xu, Omar Khattab, and Monica Lam. 2024. Assisting in writing Wikipedia-like articles from scratch with large language models. In *Proc. of NAACL*.

Wentao Shi, Zichun Yu, Fuli Feng, Xiangnan He, and Chenyan Xiong. 2025. Efficient multi-agent system training with data influence-oriented tree search. *arXiv preprint arXiv:2502.00955*.

Lei Shu, Liangchen Luo, Jayakumar Hoskere, Yun Zhu, Yinxiao Liu, Simon Tong, Jindong Chen, and Lei Meng. 2024. Rewritelm: An instruction-tuned large language model for text rewriting. In *Proc. of AAAI*.

Andrea Sottana, Bin Liang, Kai Zou, and Zheng Yuan. 2023. Evaluation metrics in the era of GPT-4: Reliably evaluating large language models on sequence to sequence tasks. In *Proc. of EMNLP*.

Haoran Sun, Lixin Liu, Junjie Li, Fengyu Wang, Baohua Dong, Ran Lin, and Ruohui Huang. 2024. Conifer: Improving complex constrained instruction-following ability of large language models. *arxiv preprint arXiv:2404.02823*.

Tianyi Tang, Junyi Li, Zhipeng Chen, Yiwen Hu, Zhuohao Yu, Wenxun Dai, Wayne Xin Zhao, Jian-yun Nie, and Ji-rong Wen. 2022. TextBox 2.0: A text generation library with pre-trained language models. In *Proc. of EMNLP*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Proc. of NeurIPS*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *Proc. of ICLR*.

Shunyu Yao, Howard Chen, Austin W. Hanjie, Runzhe Yang, and Karthik R Narasimhan. 2024. COLLIE: Systematic construction of constrained text generation tasks. In *Proc. of ICLR*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *Proc. of ICLR*.

Ryoma Yoshimura, Masahiro Kaneko, Tomoyuki Kajiwara, and Mamoru Komachi. 2020. SOME: Reference-less sub-metrics optimized for manual evaluations of grammatical error correction. In *Proc. of COLING*.

Weizhe Yuan, Graham Neubig, and Pengfei Liu. 2021. Bartscore: Evaluating generated text as text generation. *Proc. of NeurIPS*.

Urchade Zaratiana, Nadi Tomeh, Pierre Holat, and Thierry Charnois. 2024. GLiNER: Generalist model for named entity recognition using bidirectional transformer. In *Proc. of NAACL*.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. ReST-MCTS*: LLM self-training via process reward guided tree search. In *Proc. of NeurIPS*.

Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, et al. 2024b. Llama-berry: Pairwise optimization for o1-like olympiad-level mathematical reasoning. *arXiv preprint arXiv:2410.02884*.

Han Zhou, Xingchen Wan, Yinhong Liu, Nigel Collier, Ivan Vulić, and Anna Korhonen. 2024. Fairer preferences elicit improved human-aligned large language model judgments. In *Proc. of EMNLP*.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

Appendix

A More Related Work

Writing Related Agents. Previous studies (Lei et al., 2024; Shao et al., 2024; Huot et al., 2025) have investigated the use of agents for generating long-form content. Specifically, Lei et al. (2024); Huot et al. (2025) focused on employing agents for novel writing, while Shao et al. (2024) explored their application in generating Wikipedia-style articles. Unlike these approaches, our work develops an agent designed to revise paragraph-level text while adhering to various constraints.

Monte Carlo Tree Search for LLM. Monte Carlo Tree Search (MCTS) (Browne et al., 2012) has been explored from various perspectives to enhance LLMs. Specifically, Zhang et al. (2024b) employ MCTS for solving mathematical problems, while Zhang et al. (2024a); Shi et al. (2025) investigate its use in generating synthetic data to improve LLM performance. In this work, we are the *first* to apply MCTS to constrained text revision task, introducing a Tool-Guided Monte Carlo Tree Search framework that enables flexible adaptation to diverse use cases.

B Further Analysis

Named Entity Analysis. Preserving named entities is essential for maintaining the original meaning during text revision. We compute the named entity preservation rates of GPT-4o and CRAFT-4o using GLiNER (Zaratiana et al., 2024), a SOTA named entity recognition (NER) model. As shown in Fig. 9 (Left), CRAFT-4o demonstrated a higher preservation rate (81.7%) compared to GPT-4o (64.7%). Further analysis shows that excluding preservation calculations from R_c reduces the preservation rate to 76.5%. Additionally, omitting feedback results in a significant decrease, lowering the rate to 65.2%. This highlights the critical role of tool feedback and evaluation in TG-MCTS.

B.1 Other Easy Constraints

We further analyze constraint types that are relatively easy for LLMs to follow, specifically focusing on Domain & Style and Semantic constraints.

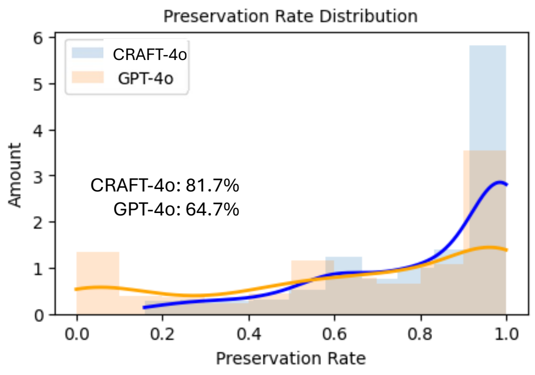


Figure 9: The Named Entity preservation rate distribution.

These constraints are constructed based on the guidelines and definitions provided by Chen et al. (2024). As shown in Table 9, although the baseline model already performs well on these constraints, our agent still achieves additional performance gains.

	Cons.	**Text Quality**		
	Acc.↑	**PPL↓**	**SOME↑**	**BARTScore↑**
DomainStyle				
Iter.	96.7	45.3	85.3	-1.32
CRAFT-4o	98.2	43.1	86.1	-1.20
Semantic				
Iter.	95.4	42.9	86.7	-1.93
CRAFT-4o	97.9	40.2	86.9	-1.87

Table 9: Performance of CRAFT-4o and GPT-4o (Iter.) on easy constraints.

B.2 Factuality Analysis

Following Lee et al. (2022), we evaluate factuality using both the named entity error rate (NE_{ER}) and the entailment ratio (Entail). Table 10 shows that CRAFT-4o exhibits a lower factual error rate compared to GPT-4o.

Factuality	**NE_{ER}↓**	**Entail↑**
Iter	31.4	13.4
CRAFT-4o	16.2	24.3

Table 10: Factuality Evaluation of CRAFT-4o.

B.3 Case Study

Fig. 10 presents a case study on tool usage.

B.4 Computation Cost

Table 11 presents the average computational cost of our TG-MCTS search algorithm across 5 domains.

Instruction	Please refine the following text:
π_p's tool	text_eval()
GPT-4o's tool	text_eval(), word_count_check(350, 'less than'), sentence_count_check(20, 'less than'), sentence_length_check(25, 'less than')
Instruction	Refine the following text for better fluency and coherence while ensuring the output exceeds 400 tokens. Keep the third and nineteenth sentences unchanged, and ensure each sentence contains more than six words:
π_p's tool	text_eval(), word_count_check(400, 'more than'), sentence_modification_check([3, 19], 'unchange'), sentence_length_check(6, 'more than')
GPT-4o's tool	text_eval(), sentence_count_check(400, 'more than'), sentence_modification_check([2, 18], 'unchange'), sentence_length_check(6, 'more than')

Figure 10: Tool usage output. Blue indicates correct tool usage, while red denotes wrong tool usage.

Notably, our method introduces minimal computational overhead compared to the best baseline, while achieving strong performance in both text quality and constraint adherence.

	C0	C1	C2	C3	C4
Iter Cnt.	×0.8	×1.1	×1.2	×1.2	×1.3
Inf.	×0.9	×1.1	×1.3	×1.4	×1.6
# Input.	×0.9	×1.2	×1.4	×1.4	×1.7
# Output.	×0.9	×1.1	×1.2	×1.2	×1.3

Table 11: Computation Cost relative to the Iter baseline. **Iter Cnt.** denotes the number of iterations, **Inf.** indicates the inference time, **# Input.** refers to the number of input tokens, and **# Output.** refers to the number of output tokens.

C CORD Details

C.1 Dataset Information

Source. We select the academic papers from the TETRA dataset (Mita et al., 2024), and select the WikiHow from the English portion of WikiLingua (Ladhak et al., 2020). We follow Que et al. (2024) to select human-written stories from the subreddit r/shortstories collections.

Statistics. We list the detailed information about the number of tokens and sentences in each domain in Table 12. We also list the number of constraints for each subtype in Table 13.

C.2 Program Template and Instruction Rendering

Program Template. A list of constrained instructions generated by the program template is shown in the 'Instruction' column of Table 14.

Instruction Rendering. Program templates allow us to generate constrained instructions based on extracted features. However, the instructions produced by these templates may lack fluency. Additionally, combining multiple sentences under a single constraint can further reduce readability. To address this issue, we follow Yao et al. (2024) and use GPT-4o to refine the instructions. Specifically, we employ the following prompt to refine the instructions:

```
Please rewrite the following paragraph
to improve fluency without altering the
original meaning.  You should provide
the revised paragraph directly. Original
paragraph: {prompt}
```

C.3 Constraint Type

Chen et al. (2024) examined the ability of LLMs to follow various constrained instructions, including length, keyword, sentiment, and topic constraints. Among these, keyword and length constraints pose the greatest challenges, yielding the lowest accuracy rates. In contrast, sentiment and topic constraints achieve nearly 90% accuracy.

Text revision primarily aims to preserve the original meaning while maintaining consistency in topic and sentiment. As a result, topic and sentiment constraints are less relevant to text revision tasks. Furthermore, as noted by Chen et al. (2024), unlike length and keyword verification, which are deterministic, topic and sentiment verification rely on probabilistic models such as BERT, making them less reliable. Therefore, our focus is on length and keyword constraints, as they are **more challenging**, **easier to verify**, and **more pertinent** to text revision scenarios.

Table 14 outlines 19 clearly defined and easily verifiable text revision constraints used in the CORD dataset. These constraints are selected based on their verifiability and prevalence in real-world applications. C1 includes a single constraint, while C2–C4 encompasses multiple constraints: C2 contains two, C3 contains three, and C4 contains four constraints from this Table.

D Implementation Details

D.1 Preliminary Details

We show the prompts for GPT-4o to generate the text revision plans in Appendix E.4, and prompts for GPT-4o to generate revisions according to the text revision plan in Appendix E.7.

Domain	#Essays	#Sentences	#Tokens
C0	100	3,256	60,897
C1	100	3,182	59,654
C2	100	3,075	59,349
C3	100	3,133	58,015
C4	100	3,106	58,636

Table 12: The detailed information about the number of essays, sentences, and tokens in each domain.

Constraint Type	Count
Keep Sentence	103
Modify Sentence	108
Word Count	155
Sentence Count	148
Per Sentence Length	167
Include Keyword	106
Remove Keyword	103
Keyword Frequency	110
Total	1000

Table 13: Detailed breakdown of constraint sub-types.

D.2 Synthetic Trajectory Details

We show our ICL example in Appendix E.3, and the prompt used by GPT-4o to generate trajectory is shown in Appendix E.1.

D.3 Scaling

We normalize SOME to a scale of 0-100 using the formula $\text{SOME}_{\text{nom}} = 100 \times \text{SOME}$. Similarly, the min-max normalization method is applied to scale the PPL and BARTScore values to the same range. The normalization equations are as follows:

$$\text{PPL}_{\text{nom}} = 100 \times \frac{\text{PPL}_{\text{min}} - \text{PPL}}{\text{PPL}_{\text{min}} - \text{PPL}_{\text{max}}}, \quad (6)$$

$$\text{BART}_{\text{nom}} = 100 \times \frac{\text{BART} - \text{BART}_{\text{min}}}{\text{BART}_{\text{max}} - \text{BART}_{\text{min}}}, \quad (7)$$

where PPL refers to the current perplexity score, while PPL_{min} and PPL_{max} denote the minimum and maximum perplexity scores, respectively. Similarly, BART refers to the current BARTScore, with BART_{min} and BART_{max} representing the minimum and maximum BARTScore values. During the implementation, BART_{max} is set to the value of 'BARTScore(init_text, init_text)' as specified by the authors [3]. Here, 'init_text' refers to the original input text. Additionally, BART_{min} is assigned a value of -10. For perplexity, PPL_{max} is set to 0, and PPL_{min} is defined as 'PPL(init_text)', which represents the perplexity score of the original input text.

[3]https://github.com/neulab/BARTScore/issues/11#issuecomment-1025988744

D.4 Action Scoring

To evaluate the quality of verification tool usage (S_v), we calculate the F_1 score by comparing the predicted tool usage with the predefined tool usage for the given constrained instructions. Revised text is generated by feeding a_i into a text-editing LLM. The quality of the revised text (S_r) is assessed using the arithmetic mean of normalized perplexity (PPL), BARTScore, and SOME, following the method described in Appendix D.3. Finally, the quality score S_c is computed by employing the predefined tool usage for each instruction to verify the accuracy of the revised text. We treat Llama-3.1-8B-Instruct as the vanilla LLM to generate revisions according to a_{i+1}.

D.5 TG-MCTS implementation details

To calculate R_c, we utilize tools generated by π_p to verify whether the revised text meets the specified requirements, assessing performance using accuracy as the metric. For normalized PPL, SOME, and BARTScore calculations, we follow the method outlined in Appendix D.3. Our TG-MCTS terminates when either the revision quality (in terms of both text quality and constraint adherence) ceases to improve or the maximum number of iterations is reached.

D.6 Experimental Details

We use a one-tailed sign test with bootstrap resampling for statistical significance tests (Cao et al., 2021).

Hyper-parameters. To emphasis more accurate tool usage by π_p, we set λ_v to 0.5, and both λ_r and λ_c to 0.25. Following Meng et al. (2024), the parameters β and γ in Eq. 2 are set to 2.5 and 1.375, respectively. For TFG-MCTS, α in Eq. 3 is set to 0.2, with three child nodes expanded per parent node. The maximum depth of the tree is 6 layers, and the maximum number of iterations is 30. For GPT-4o models, we set the temperature to 0.7 and used the 'gpt-4o-2024-08-06' model card, and for GPT-4, we used the 'gpt-4-0613' model card.

Implementations and Computation Hardware. Our experiments were conducted on four A100

Constraint Group	Sub Group	Instruction
Sentence Constraint	Keep Sentence	Do not change the {I}-th sentence. Do not change the {I}-th, and {J}-th sentence. Do not change the {I}-th, {J}-th, and {K}-th sentence.
	Modify Sentence	Only change the {I}-th sentence. Only change the {I}-th, and {J}-th sentence. Only change the {I}-th, {J}-th, and {K}-th sentence.
Length Constraint	Word Count	Output contain more than {N} words. Output contain less than {N} words. Output contain less than {N} words and more than {M} words.
	Sentence Count	Output contain more than {N} sentences. Output contain less than {N} sentences. Output contain should contain exactly {N} sentences.
	Per Sentence Length	Each sentence should contain more than {N} words. Each sentence should contain less than {N} words.
Keyword Constraint	Include Keyword	Do not change the word '{A}'.
	Remove Keyword	Do not use the word '{A}'.
	Keyword Frequency	The word '{A}' should appear {N} times. The word '{A}' should appear at least {N} times. The word '{A}' should appear less than {N} times.

Table 14: A list of 19 verifiable text revision constraints. '{I}', '{J}', and '{K}' represent the sentence ids, while '{N}' and '{M}' denote the number of words, sentences, or occurrences. '{A}' represents the keyword.

80GB GPUs with CUDA version 12.1. The planner's code is based on the Hugging Face TRL package[4], and CPO_SIMPO[5]. We employed DeepSpeed's Zero-Offload[6] and LoRA techniques for fine-tuning the planner. The supervised fine-tuning (SFT) process took approximately 6 hours, while each self-training alignment process required about 12 hours. Inference for CRAFT-4o and CRAFT-3.1 was performed using the vLLM package and took approximately four hours in total.

Baselines. We compare the CRAFT with several strong baselines for the text revision and instruction-following task as follows:

- **CoEDIT-C** (Raheja et al., 2023). CoEDIT-Composite is a SOTA text revision LLM fine-tuned on the Flan-T5 (Chung et al., 2022) using composite text revision instructions. These instructions encompass grammatical error correction, paraphrasing, and simplification tasks. However, rather than processing paragraph-level input, CoEDIT-C is trained to handle only sentence-level input. To establish a CoEDIT-C baseline, we first segment texts into individual sentences, apply CoEDIT-C to revise each sentence using detailed instructions and then recombine the revised sentences into texts.

- **Evol-Instruct** (Xu et al., 2024). Evol-Instruct is the publicly available WizardLM-Evol-Instruct dataset, comprising 143k samples that integrate Alpaca and ShareGPT-evolved data. Following the methodology outlined in the original paper, we fine-tune Llama-3.1-8B-Instruct on this dataset to establish the Evol-Instruct baseline.

- **Conifer** (Sun et al., 2024). Conifer is a language model optimized for following complex, constraint-based instructions. It employs a progressive learning strategy, gradually increasing task complexity to enhance its ability to handle intricate instructions. The dataset, curated using GPT-4, provides diverse and challenging instruction sets, making Conifer particularly effective in real-world applications requiring precise adherence to instructions. Following the original paper's methodology, we fine-tune Llama-3.1-8B-Instruct on this dataset and further fine-tune it using the UltraFeedback (Cui et al., 2024) dataset with DPO to establish the Conifer baseline.

[4]https://github.com/huggingface/trl
[5]https://github.com/fe1ixxu/CPO_SIMPO/tree/main
[6]https://github.com/microsoft/DeepSpeed

D.7 LLM-as-a-Judge Setting

The detailed prompts used for GPT-4 scoring from fluency, coherence, and grammaticality aspects are provided in Appendix E.5. For pairwise comparisons, we utilize the prompt template presented in Table 15. To mitigate potential position bias during evaluation, we randomly assign the generated revisions by CRAFT-4o as either "System A" or "System B".

```
Source text: [Input]

Revised Text Candidate A: [Candidate_A]

Revised Text Candidate B: [Candidate_B]

Question: [Aspect_Prompt]

Answer:
```

Table 15: Prompt template for pairwise comparison between WRA-4o and GPT-4o. The `[Input]` represents the original text, `[Candidate_A]` denotes the revision produced by System A, `[Candidate_B]` denotes the revision produced by System B, and `[Aspect_Prompt]` for different aspects are provided in Appendix E.6.

D.8 LaTeX Selection Details

TETRA (Mita et al., 2024) comprises 64 research papers written by non-native speakers. Among them, we identified nine papers with available LaTeX source code. Table 16 presents the URLs for these papers. From these papers, we extracted 100 paragraphs containing at least two LaTeX keywords.

ID	URL
1	https://arxiv.org/abs/1805.11267
2	https://arxiv.org/abs/1603.03116
3	https://arxiv.org/abs/1705.00823
4	https://arxiv.org/abs/1704.04859
5	https://arxiv.org/abs/1606.01323
6	https://arxiv.org/abs/1810.05104
7	https://arxiv.org/abs/1804.10959
8	https://arxiv.org/abs/1705.00316
9	https://arxiv.org/abs/1805.07043

Table 16: URLs of papers from which we obtained the LaTeX source.

```
###THOUGHT: [Thought_1]
###ACTION:
  - ###TOOLS: [Tool_1]
  - ###PLAN: [Plan_1]
###OBSERVATION: [Observation_1]
...
###THOUGHT: [Thought_n]
###ACTION:
  - ###TOOLS: [Tool_n]
  - ###PLAN: [Plan_n]
###OBSERVATION: [Observation_n]
```

Table 17: An example of `[In Context Examples]` includes **n** rounds of text refinements.

E Prompts

E.1 Trajectory Generation Prompt

Below, we present the prompt used by GPT-4o and the Planner, π_p to generate the trajectory. The `[Tool Descriptions]` are provided in Appendix E.2, and the `[In-Context Examples]` are shown in Table 17.

```
You are an agent tasked with improving text according to the user's specific
instructions, using the framework outlined below.
—
### Text Improvement Framework
1. Identify Areas for Improvement:
  - Based on the user's instructions, determine the specific aspects that need
enhancement (e.g., grammar, clarity, style, word choice).
  - Decide which text quality evaluation tools to use from the provided list.
2. Evaluate Text Quality:
  - Select appropriate evaluation tools to obtain feedback on the text's quality.
  - Use the tools as specified to check for grammaticality, fluency, coherence, and
other conditions.
3. Analyze Each Sentence:
  - For each sentence in the input text, perform the following steps:
    - Sentence: "[Original Sentence]"; Improvement Plan: "[Your plan for improving
the sentence]"
—
### Available Evaluation Tools
[Tool Descriptions]
—
### Response Format
When you respond, strictly follow this format to present your thoughts and actions:
1. ###THOUGHT:
  - Describe your thought process on how to improve the text.
2. ###ACTION:
  - ###TOOLS:
    - **Instructions:** List any tool function calls you are making, using the
exact function call format as specified in the "Available Evaluation Tools" section,
including the function name and parameters.  Write them as code lines without
additional explanations.
    - **Example:**
    "‘
    word_count_check(300, "less than")
    keyword_keep_removal_check("few years.", "remove")
    sentence_count_check(18, "more than")
    sentence_length_check(20, "less than")
    sentence_modification_check(3, "change")
    keyword_frequency_check("eat", 2, "less than")
    "’
  - ###PLAN:
    For each sentence:
    Sentence: "[Original Sentence]"; Improvement Plan: "[Your plan for improving
the sentence]"
—
Note:
- After each action, the user will provide the tools' output in the following format:
```

```
“###OBSERVATION: Tool’s output result.”
[In Context Examples]
```

E.2 Tool Description

```
#### 1. Text Quality Tool
- Function: ‘text_eval() → score’
  - Purpose: Evaluates the text’s overall quality in terms of grammaticality,
fluency, and coherence.
  - Output: Returns a score reflecting the overall text quality.

#### 2. Keyword Detection Tool
- Function: ‘detect_keyword() → word’
  - Purpose: Evaluates the keywords to be preserved in the text.
  - Output:
    - ‘word’: returns the detected keyword in the text.

#### 3. Condition Checking Tools
- a. Word Count Check
  - Tool: ‘word_count_check(count: int, relation: str) → count, label’
    - Purpose: Checks if the word count meets a specified condition.
    - Parameters:
      - ‘count’: Target word count.
      - ‘relation’: “less than”, “more than”, or “equal”.
    - Output:
      - ‘count’: Actual word count.
      - ‘label’: ‘0’ if the condition is met, ‘1’ otherwise.
- b. Keyword Presence Check
  - Tool: ‘keyword_keep_removal_check(keyword: str, relation: str) → label’
    - Purpose: Checks if a keyword is present or absent in the text.
    - Parameters:
      - ‘keyword’: The keyword to check.
      - ‘relation’: “keep” (keyword should be present) or “remove” (keyword should
be absent).
    - Output:
      - ‘label’: ‘0’ if the condition is met, ‘1’ otherwise.
- c. Keyword Frequency Check
  - Tool: ‘keyword_frequency_check(keyword: str, frequency: int, relation: str) →
occurrence, label’
    - Purpose: Counts the occurrences of a keyword and checks if it meets the
specified frequency condition.
    - Parameters:
      - ‘keyword’: The keyword to count.
      - ‘frequency’: Target number of occurrences.
      - ‘relation’: “less than”, “more than”, or “equal”.
    - Output:
      - ‘occurrence’: Actual occurrence count.
      - ‘label’: ‘0’ if the condition is met, ‘1’ otherwise.
- d. Sentence Modification Check
  - Tool: ‘sentence_check(sentence_id: list, relation: str) → label’
    - Purpose: Checks if specified sentences have been changed or remain unchanged.
    - Parameters:
      - ‘sentence_id’: List of sentence indices (e.g., ‘[1, 3]’ means the target
sentence are the 1st and the 3rd sentences).
      - ‘relation’: “change” (sentences should be modified) or “unchange” (sentences
should remain the same).
    - Output:
      - ‘label’: ‘0’ if the condition is met, ‘1’ otherwise.
- e. Sentence Count Check
  - Tool: ‘sentence_count_check(count: int, relation: str) → label’
    - Purpose: Checks if the total number of sentences meets a specified condition.
    - Parameters:
      - ‘count’: Target number of sentences.
      - ‘relation’: “less than”, “more than”, or “equal”.
    - Output:
      - ‘count’: Actual sentence count.
      - ‘label’: ‘0’ if the condition is met, ‘1’ otherwise.
- f. Sentence Length Check
  - Tool: ‘sentence_length_check(length: int, relation: str) → label’
    - Purpose: Checks if each sentence’s length meets a specified condition.
    - Parameters:
      - ‘length’: Target sentence length (in words).
      - ‘relation’: “less than”, “more than”, or “equal”.
    - Output:
      - ‘label’: ‘0’ if all sentences meet the condition, ‘1’ otherwise.
```

E.3 ICL Example for Synthetic Trajectory

Due to the extensive nature of paragraph-level text revision plans, we provide a representative, human-labeled partial example from TETRA (Mita et al., 2024) below:

```
Sentence: Large-scale parsing-based statistical machine translation (MT) has made
remarkable progress in the last few years.
Improvement Plan: Clarify that it is the field or work of large-scale parsing-based
statistical machine translation that has made progress, not the translation
itself. Remove the phrase ’in the last few years’ and consider replacing it with a
synonym or phrasing that communicates on-going progress without a specific time frame.

Sentence: The systems being developed differ in whether they use source- or
target-language syntax.
Improvement Plan: Clarify that the systems vary based on their reliance on the
syntax of either the source language or the target language.

Sentence: For instance, the hierarchical translation system of Chiang (2007)
extracts a synchronous grammar from pairs of strings, Quirk et al. (2005), Liu et al.
(2006) and Huang et al. (2006) perform syntactic analyses in the source language,
and Galley et al. (2006) use target-language syntax.
Improvement Plan: Break down the sentence to improve clarity by listing the specific
contributions of each referenced work separately. Ensure the sentence clearly
explains how each work approaches translation, focusing on the use of synchronous
grammar, source-language syntactic analysis, and target-language syntax.

Sentence: A critical component in parsing-based MT systems is the decoder,
which is complex to implement and scale up.
Improvement Plan: Simplify the wording to make the sentence more accessible and
clarify the complexity of implementing and scaling the decoder in parsing-based
machine translation systems.

Sentence: Most of the systems described above employ tailor-made, dedicated
decoders that are not open-source, which results in a high barrier to entry for
other researchers in the field.
Improvement Plan: Simplify the sentence structure and clarify that the use of
proprietary decoders limits access for researchers, thus hindering collaboration and
innovation in the field.

Sentence: However, with the algorithms proposed in (Huang and Chiang, 2005;
Chiang, 2007; Huang and Chiang, 2007), it is possible to develop a general-purpose
decoder that can be used by all the parsing-based systems.
Improvement Plan: Simplify the sentence structure and clarify the references to
make the statement more concise and easier to understand. Use more straightforward
language to convey the idea that the algorithms allow for the creation of a versatile
decoder applicable to various parsing-based systems.
```

E.4 Plan Generation Prompt for GPT-4o

Below, we present the prompt used by GPT-4o to generate text revision plans for a given text.

```
You are an expert writing assistant specializing in text revision. Your task is to
analyze a given text and generate a revision plan for each sentence while following
the specific format:

Sentence: [Original sentence]; Improvement Plan: [Suggested revision strategy].

For each sentence, identify any issues related to clarity, grammar, conciseness,
tone, or logical flow. Then, propose a concrete improvement plan to enhance the
sentence while maintaining its original intent.

Example Output Format:
Original Text:
“[Insert text here]”

Sentence-by-Sentence Revision Plan:

Sentence: “[Original sentence]”; Improvement Plan: “[Brief but clear revision
strategy]”.
Sentence: “[Original sentence]”; Improvement Plan: “[Brief but clear revision
strategy]”.
Sentence: “[Original sentence]”; Improvement Plan: “[Brief but clear revision
strategy]”.
(Continue for all sentences in the text.)

The improvement plan should be specific and actionable, explaining what should
be changed and why. Avoid vague feedback—focus on how to improve clarity,
conciseness, structure, and readability. If necessary, suggest alternative phrasing
or restructuring.
```

E.5 GPT-4 Scoring Prompt for Grammaticality, Fluency and Coherence

Grammaticality. Prompt from Sottana et al. (2023) for GPT-4 to score the **Grammaticality**:

You’re GPT4 and are about to start a task where you will be shown some sentences written by learners of English. Some of these sentences will contain errors, and alongside each sentence you will be shown 2 possible corrections, and you will be asked to evaluate the quality of the correction based on some metrics defined below. This task is called Grammatical Error Correction (GEC), and is the task of automatically detecting and correcting errors in text. The task not only includes the correction of grammatical errors, such as missing prepositions and mismatched subject-verb agreement, but also orthographic and semantic errors, such as misspellings and word choice errors respectively. Note that not all sentences you will see include grammatical errors; if they do not, we would expect the corrected version to be identical to the source. We ask that you carefully read the original sentence and rank each of the 4 corrections according to the following metrics, which are defined below.

Semantics. This assesses whether the meaning of the text is preserved following the GEC. Semantic preservation is assessed on a 5-point Likert scale from 1 (Meaning Not Preserved) to 5 (Meaning fully preserved). NOTE: You should penalize corrections which change the meaning unnecessarily. For example, the sentence “I wentt at Rome for my birthday” should be corrected to “I went to Rome for my birthday”. A correction such as “I went to Rome for my anniversary” should be penalised in this category as it introduces unnecessary changes to the meaning.

Grammaticality. This assesses the quality of the correction and answers the question “How many errors are left in the corrected sentence?”. Please provide a count of the remaining errors, regardless of whether they were present in the source or they were newly introduced errors in the supposed corrected version. The options are “0”, “1”, “2 or more”. Note that, unlike for semantics where a score of 5 is better than a score of 1, here a score of “0” is better than a score of “1” which is better than a score of “2 or more” (this is because if there are 0 errors remaining, the GEC task has been fulfilled).

Over-correction. Since there can be multiple ways to correct a sentence, this assesses whether the correction is unnecessarily verbose or makes unnecessary syntax changes. The best correction should be done with the minimum number of edits. For example, if the sentence “I wentt at Rome for my birthday” is corrected to “I decided to go to Rome for my birthday” this should be penalized under this category because it contains unnecessary syntax changes, even though the final sentence is grammatically correct. This metric answers the question: Is the system over-correcting or making unnecessary syntax changes? The answers should be “No”, “Minor over-correction”, “Moderate over-correction” or “Substantial over-correction”.

We will pass you the input you need to rank in json format.
Please reply with the scores in json format.
This is an example json query where “original_input” is the source sentence, “id” is the unique identifier, and all other keys represent the output corrected sentences which you need to evaluate.
[Input_Example]

```
Your answer should contain the id and the scores, for example, using the example
given above, if you wish to give llama3 a semantics score of 5, a grammaticality
score of “0”, an overcorrection score of “No”, and you wish to give llama3_agent
a semantics score of 4, a grammaticality score of “1”, an overcorrection score of
“Minor over-correction”, then you should return the following output (note how the
id item needs to be preserved to allow for identification):
“llama3”: “semantics”: 5, “grammaticality”: “0”, “overcorrection”: “No”,
“llama3_agent”: “semantics”: 4, “grammaticality”: “1”, “overcorrection”: “Minor
over-correction”, “id”: “12”

Is this clear? Do you have any questions or are you ready to start?
```

Fluency and Coherence. Prompt adapted from Sottana et al. (2023) for GPT-4 to score the **Fluency** and **Coherence** are shown below:

```
You’re GPT-4 and are about to start a task where you will be shown some pieces of text
taken mostly from older articles, alongside 2 different possible text refinement
options, and you will be asked to evaluate the quality of the refined text based
on some metrics defined below. The purpose of text refinement is to make the text
more fluent and more grammatical without changing its overall meaning, omitting
unimportant details while retaining the key content of the original text. We ask
that you carefully read the original text and rank each of the refined text according
to the following metrics, which are defined below.

Fluency. This assesses the quality of individual sentences. Sentences in the
refined text should have no formatting problems, capitalization errors, or obviously
ungrammatical sentences (e.g., fragments, missing components) that make the text
difficult to read. Fluency is assessed on a 5-point Likert scale from 1 (Not Fluent)
to 5 (Super Fluent)

Coherence. This assesses the collective quality of all sentences. The refined text
should be well-structured and well-organized. The refined text should not just be a
heap of related information but should build from sentence to sentence to a coherent
body of information about a topic. Coherence is assessed on a 5-point Likert scale
from 1 (Not Coherent) to 5 (Super Coherent)

Consistency. This assesses the factual alignment between the refined text and
the source. A factually consistent refinement contains only statements that are
entailed by the source document. Refinements that contain hallucinated facts
(information which is not present in the source document) should be penalized.
Consistency is assessed on a 5-point Likert scale from 1 (Not Consistent) to 5 (Super
Consistent)

We will pass you the input you need to rank in json format.
Please reply with the scores in json format.
This is an example json query where “original_input” is the source text, “id” is
the unique identifier, and all other keys represent output texts which you need to
evaluate.
[Input_Example]

Your answer should contain the id and the scores, for example, using the
example given above, if you wish to give llama3 a fluency score of 5, a coherence
```

score of 4, and a consistency score of 4, and you wish to give llama3_agent a fluency score of 5, a coherence score of 1 and a consistency score of 3, then you should return the following output (note how the id item needs to be preserved to allow for identification):
“llama3”: “fluency”: 5, “coherence”: 4, “consistency”: 4, “llama3_agent”: “fluency”: 5, “coherence”: 1, “consistency”: 3, “id”: “12”

Is this clear? Do you have any questions, or are you ready to start?

E.6 GPT-4 Pairwise Comparision Prompt

Fluency. The prompt used to conduct pairwise comparison from the fluency perspective:

```
Assess and contrast the fluency of the two improved text options provided for the given input. Determine which text option demonstrates superior fluency. If candidate A excels, respond with 'A'; if candidate B is better, respond with 'B'. Your reply must solely indicate the chosen option.
```

Grammaticality. The prompt used to conduct pairwise comparison from the grammaticality perspective:

```
Assess the grammatical quality of the two revised text options based on the provided input. A text is considered grammatical when it is free from grammar errors. Among the two options, the text with fewer grammar errors is more grammatical, while the one with more errors is less grammatical. Determine which revised text demonstrates superior grammar. If candidate A has better grammar, respond with 'A'. If candidate B has better grammar, respond with 'B'. Your response must strictly indicate the choice only.
```

Coherence. The prompt used to conduct pairwise comparison from the coherence perspective:

```
Assess the coherence of the two refined text options based on the provided input text. Evaluate coherence in terms of clarity and logical progression. A coherent text effectively conveys the essential information from the input while maintaining a clear and organized structure. Determine which refined text option demonstrates superior coherence. If candidate A is better, respond with 'A'. If candidate B is better, respond with 'B'. Provide only your selection.
```

E.7 Generate Revision According to Plan

The prompt used to generate the revision based on the text revision plan is shown below:

```
### INSTRUCTIONS:

Using the information provided in each text editing plan (### INPUT), generate the polished version of each sentence by applying the specified improvements. Maintain the original order of sentences.

***In your output, provide only the final polished sentences, one after another, without any prefixes, numbering, or additional text.**

### INPUT:
```

E.8 Feedback Prompts

Fluency. The prompt used to generate the linguistic feedback from the fluency perspective:

```
Please analyze the following text for fluency issues, including awkward phrasing,
unnatural word choices, sentence flow, and readability problems. For each of the
sentences that contain fluency problems, please format the output strictly as follows:
‘Original: [original text]; Suggestion: [corrected text]’. If a sentence has no
issues, do not include it in the output. Do not include any additional content.
Text:
```

Grammaticality. The prompt used to generate the linguistic feedback from the grammaticality perspective:

```
Please analyze the following text for grammatical errors, including issues with
sentence structure, punctuation, subject-verb agreement, tense consistency, pronoun
usage, and any other common grammar mistakes. For each of the sentences that contain
grammar errors, please format the output strictly as follows: ‘Original: [original
text]; Suggestion: [corrected text]’. Do not include any additional content. Text:
```

Coherence. The prompt used to generate the linguistic feedback from the coherence perspective:

```
Please analyze the following text for coherence problems, such as unclear connections
between ideas, lack of logical flow, abrupt transitions, or inconsistencies in the
overall message. For each sentence or section that contains a coherence problem,
format the output strictly as follows: ‘Original: [original text]; Suggestion:
[corrected text]’. If a sentence or section has no issues, do not include it in the
output. Do not include any additional content. Text:
```