

SELF-REFINE: ITERATIVE REFINEMENT WITH SELF-FEEDBACK

Aman Madaan¹, Niket Tandon², Prakhar Gupta¹, Skyler Hallinan³, Luyu Gao¹, Sarah Wiegrefe², Uri Alon¹, Nouha Dziri², Shrimai Prabhumoye⁴, Yiming Yang¹, Sean Welleck^{2,3}, Bodhisattwa Prasad Majumder⁵, Shashank Gupta², Amir Yazdanbakhsh⁶, Peter Clark²

¹Language Technologies Institute, Carnegie Mellon University

²Allen Institute for Artificial Intelligence

³University of Washington ⁴NVIDIA ⁵UC San Diego ⁶Google Research, Brain Team
amadaan@cs.cmu.edu, nikett@allenai.org

ABSTRACT

Like people, LLMs do not always generate the best text for a given generation problem on their first try (e.g., summaries, answers, explanations). Just as people then refine their text, we introduce SELF-REFINE, a framework for similarly improving initial outputs from LLMs through iterative feedback and refinement. The main idea is to generate an output using an LLM, then allow the same model to provide multi-aspect feedback for its own output; finally, the same model refines its previously generated output given its own feedback. Unlike earlier work, our iterative refinement framework does not require supervised training data or reinforcement learning, and works with a single LLM. We experiment with 7 diverse tasks, ranging from review rewriting to math reasoning, demonstrating that our approach outperforms direct generation. In all tasks, outputs generated with SELF-REFINE are preferred by humans and by automated metrics over those generated directly with GPT-3.5 and GPT-4, improving on average by absolute $\sim 20\%$ across tasks.¹

1 INTRODUCTION

Iterative refinement, a fundamental characteristic of human problem-solving, is a process that involves creating an initial draft and subsequently refining it through self-feedback (Simon, 1962; Flower and Hayes, 1981; Amabile, 1983). For example, when drafting an email to request a document from a colleague, an individual may initially write a direct request such as “send me the data ASAP.” Upon reflection, however, the writer might recognize the potential impoliteness of the phrasing and revise it to “could you please send me the data?”. In this paper, we demonstrate that large language models (LLMs) can effectively replicate this human cognitive process by employing iterative feedback and refinement.

Although LLMs can generate coherent outputs in the initial step, they often fall short in addressing more intricate requirements, especially for tasks with multifaceted objectives (e.g., dialogue response generation with criteria such as making the response relevant, engaging, and safe) or those with less defined goals (e.g., enhancing program readability). In such scenarios, modern LLMs may produce an intelligible output, but iterative refinement is necessary to ensure that all aspects of the task are met and the desired quality is achieved (Reid and Neubig, 2022; Schick et al., 2022a; Welleck et al., 2022). More advanced approaches that rely on external supervision and reward models require significantly large training data or expensive

¹Code, data, and demo at <https://selfrefine.info/>

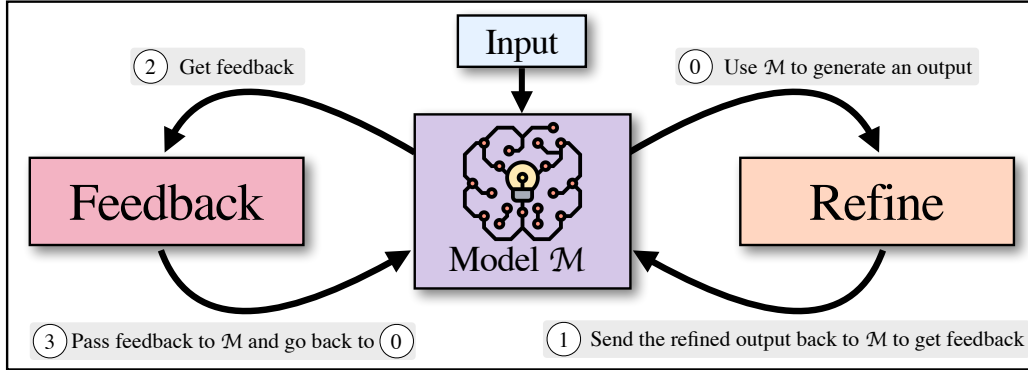


Figure 1: SELF-REFINE starts by taking an initially generated output (①), and passing it back to the same model \mathcal{M} (②) to get feedback (③); feedback on the initial output is passed back to the model (③), to iteratively refine (①) the previously generated output. SELF-REFINE is instantiated with a powerful language model such as GPT-3.5 and does not involve human assistance.

human annotations, which may not always be feasible to obtain (Madaan et al., 2021; Welleck et al., 2022). These limitations underscore the need for a more flexible and effective approach to generating text that can be applied to various tasks without requiring extensive supervision.

To overcome these limitations and better mimic the human creative generation process without an expensive human feedback loop (Ouyang et al., 2022), in this paper we propose SELF-REFINE (Figure 1). SELF-REFINE consists of an iterative loop between two components: FEEDBACK, and REFINE, which work in tandem to generate high-quality outputs. Given an initial *draft* output generated by a model \mathcal{M} (①), we pass it back to the same model \mathcal{M} (②) to get *feedback* (③). Feedback on the initial output is passed back to the same model (③), to iteratively refine (①) the previously generated output. This process is repeated iteratively for a specified number of iterations, or until the model itself determines that no further refinement is necessary. The main idea in this work is that *the same underlying language model performs both feedback and refinement* in a few-shot setup. An illustration of the process is shown in Figure 1.

We apply SELF-REFINE to various tasks that span diverse domains and require different feedback and revision strategies, including review rewriting, acronym generation, constrained generation, story generation, code rewriting, response generation, and toxicity removal. We adopt a few-shot prompting (Brown et al., 2020) approach to instantiate our key components, allowing us to leverage a small number of examples to bootstrap the model’s learning. SELF-REFINE is the first to offer an iterative approach to effectively improve generation using NL feedback. We hope that our iterative framework, through experiments and analysis of various components, diversity of tasks, generating actionable feedback and stopping criteria, will help drive further research in this area.

In summary, our contributions are:

1. We propose SELF-REFINE, a novel approach that allows LLMs to iteratively refine their own outputs using their own feedback, along multiple dimensions to improve performance on diverse tasks. Unlike prior work, our approach does not require supervised training data or reinforcement learning, and uses a single LLM.
2. We conduct extensive experiments on 7 diverse tasks of review rewriting, acronym generation, story generation, code rewriting, response generation, constrained generation, and toxicity removal,

	Few-shot refiner	Few-shot feedback	Multi-aspect scored feedb.	Iterative framework
Learned Refiners: PEER (Schick et al., 2022b), Self-critique (Saunders et al., 2022b), Self-correct (Welleck et al., 2022),	✗	✓ or ✗	✗	✓ or ✗
RL-based: QUARK (Lu et al., 2022), RLHF (Bai et al., 2022a), CodeRL (Le et al., 2022b), Constitutional-AI (Bai et al., 2022b)	✗	✗	✗	✗
LLM-based: Self-ask (Press et al., 2022a), Augmenter (Peng et al., 2023), Re ³ (Yang et al., 2022), Reflexion (Shinn et al., 2023)	✓	✓ or ✗	✗	✗
SELF-REFINE (this paper)	✓	✓	✓	✓

Table 1: A comparison of SELF-REFINE to closely related prior approaches

demonstrating that SELF-REFINE outperforms direct generation from strong generators like GPT-3.5 and even GPT-4 by at least 5%, and up to more than 40% improvement.

2 RELATED WORK

Leveraging human- and machine-generated natural language (NL) feedback has been effective for a variety of tasks, including summarization (Scheurer et al., 2022), script generation (Tandon et al., 2021), program synthesis (Le et al., 2022a; Yasunaga and Liang, 2020), computer vision (Rupprecht et al., 2018), and other tasks (Bai et al., 2022a; Schick et al., 2022b; Saunders et al., 2022a; Bai et al., 2022b; Press et al., 2022a). This feedback can differ in the source of feedback, the representation format, and the utilization of the feedback in generating the refined output. Table 1 summarizes some of the related approaches.

Source of feedback Humans have been an effective source of feedback (Tandon et al., 2021; Elgohary et al., 2021; Tandon et al., 2022; Bai et al., 2022a). Reinforcement learning (RL) based approaches (Bai et al., 2022a; Liu et al., 2022; Lu et al., 2022; Le et al., 2022a) have been used to optimize for human preference or end task accuracy and generate feedback. To reduce the human cost, automated sources such as compilers (Yasunaga and Liang, 2020) or existing online sources like Wikipedia edits (Schick et al., 2022b) have been employed for specific tasks and domains. Recently, LLMs have been used to generate feedback for a general domain solution (Press et al., 2022a; Fu et al., 2023; Peng et al., 2023; Yang et al., 2022). However, different from these works, our goal is not only to generate feedback from LLMs. Rather, we propose soliciting feedback from an LLM on its own output, refining the output with feedback, and repeating this feedback-refine process. So far, machine-generated feedback from prompting has yet to be found to be beneficial (Saunders et al., 2022a; Bai et al., 2022b). However, in this work provides evidence that feedback generated with zero- or few-shots can be helpful. Typically, an LLM feedback provider evaluates one aspect, but for certain domains, evaluating multiple aspects has also been successful (Bai et al., 2022b; Yang et al., 2022). We extend this to an LLM being used as a source of multiple feedback across a diverse set of domains, and evaluating different aspects of the generated output.

Representation of the feedback The two main common forms of feedback are natural language (NL) and non-NL feedback. Non-NL feedback can come in human-provided example pairs (Dasgupta et al., 2019) or dense signals (Liu et al., 2022; Le et al., 2022b). NL feedback can be prescriptive, e.g., Elgohary et al. (2021) apply syntactic edit operations expressed by the user via NL: “replace course id with program

id”. Alternatively, NL feedback can provide hints to nudge the output, e.g., [Mehta and Goldwasser \(2019\)](#) give a robot directional feedback such as “move down”. Effective use of NL feedback is an emerging ability ([Saunders et al., 2022a](#)), and LLMs can utilize more complex feedback to explain why the output is undesirable, even if the correct answer is unknown ([Shinn et al., 2023](#)). Examples include [Tandon et al. \(2021\)](#) and [Madaan et al. \(2022\)](#), which use background or corrective knowledge feedback such as “You cannot drive a car without entering it”. In this work, we use NL feedback, since this allows to easily provide *self*-feedback using the same LM that generated the output, while leveraging existing pretrained LLMs such as GPT-4.

Utilization of feedback Pairs of feedback and revision have been employed to learn supervised refiners and correctors ([Schick et al., 2022b](#); [Yasunaga and Liang, 2020](#); [Welleck et al., 2022](#); [Bai et al., 2022b](#)). However, gathering supervised data from humans is costly; to overcome this, RL-based refiners have been proposed for effective feedback ([Le et al., 2022a](#); [Yang et al., 2022](#); [Stiennon et al., 2020a](#)). However, out-of-domain generalization for RL approaches is still challenging, and these models must be *trained* for new domains. Recent work used the feedback using a refiner that is *trained* using RL ([Peng et al., 2023](#); [Yang et al., 2022](#)). In this work, we avoid training a separate refiner and employ a few-shot LLM refiner for multiple domains.

Iterative refinement A corrector can either be used once ([Saunders et al., 2022a](#)) or multiple times ([Yasunaga and Liang, 2020](#); [Schick et al., 2022b](#); [Peng et al., 2023](#); [Yang et al., 2022](#)). A related concept is an iterative refinement, which does not require a corrector, such as in [Press et al. \(2022a\)](#), and instead asks successive questions to develop its answers. However, when using a refiner iteratively, there is no guarantee that the model will converge or improve. To address this, [Welleck et al. \(2022\)](#) selected the best output by relying on knowing the ground truth at test time. In contrast, our SELF-REFINE approach includes a scalar value-based stopping criteria that overcomes this issue.

See Table 7 for an additional detailed comparison of related work.

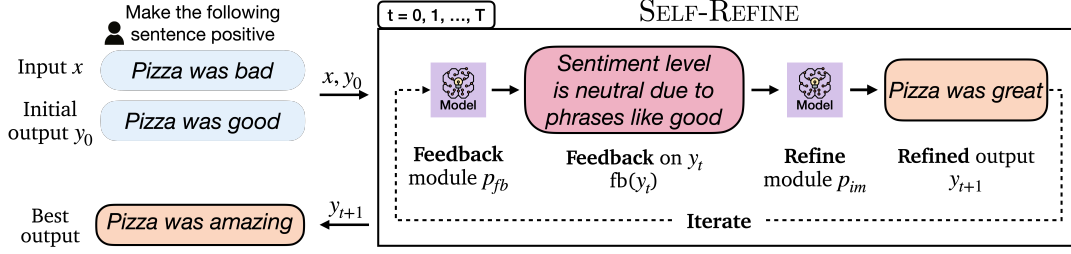
3 SELF-REFINE FRAMEWORK

In this section, we provide an overview of our approach. Given an initial task and prompt, SELF-REFINE generates a first output attempt, then provides feedback to its own output, and finally enhances the generated output according to the feedback. Furthermore, we can *iteratively* apply this feedback-and-refine approach for each subsequent generation, allowing for iterative refinement.

Few-shot Prompting Few-shot prompting (sometimes referred to as “in-context learning”) involves providing the model with a prompt consisting of k in-context examples, each in the form of input-output pairs $\langle x_i, y_i \rangle$ ([Brown et al., 2020](#)). The number of pairs depends on the maximum input sequence length of the model, and is thus typically limited to $k \leq 10$. Each $\langle x_i, y_i \rangle$ pair represents the target task and provides context for the model to generate the desired output. The test input x_t is appended to the end of the prompt, expecting the model to generate y_t .

3.1 THE SELF-REFINE FRAMEWORK

Given an input x , and an initial output y_0 , SELF-REFINE successively refines the output in a FEEDBACK \rightarrow REFINE \rightarrow FEEDBACK loop. We assume that the initial output y_0 is produced by a generator model, which could be a specialized fine-tuned model or a few-shot prompted model. For example, for the task of Sentiment Reversal, when provided with an input review “*the pizza was bad*” and a target sentiment of *positive*, the generator might produce “*the pizza was good*” (Figure 1). This output y_0 is then passed on for iterative refinement through the SELF-REFINE loop, comprised of introspection with feedback (FEEDBACK) and improvement (REFINE) stages (Algorithm 1, Figure 1), explained below.



Algorithm 1 SELF-REFINE algorithm

Require: input x , initial output y_0 , feedback module p_{fb} , refine module p_{im}

- 1: **for** iteration $t \in 0 \dots T$ **do**
- 2: $fb, fb_{score} \sim p_{fb}(y_t)$ ▷ Get Feedback
- 3: **if** $\text{stop}(fb_{score})$ **then**
- 4: **break** ▷ Early stopping
- 5: **else**
- 6: $y_{t+1} \sim p_{im}(y_t | y_{\leq t}, x, fb, fb_{score})$ ▷ Get Refined output
- 7: **end if**
- 8: **end for**
- 9: **return** $\arg \max_t fb_{score}(y_t)$ ▷ Best output selection

Figure 2: SELF-REFINE algorithm and an illustrated example. (Top) SELF-REFINE applied to Sentiment Reversal. (Bottom) our algorithm, see §3.1 for details.

FEEDBACK receives the initial output y_0 and provides feedback on how to enhance it. This feedback is task-dependent and generally addresses multiple aspects of the input. In the given example, the feedback might concern the sentiment level and vividness of the review. Importantly, our feedback is *actionable*, as we encourage the model to identify specific areas that can be refined (e.g., “the sentiment level is neutral” or “the sentiment is neutral due to phrases like good”).

REFINE is responsible for refining an output y_t based on the received feedback and the previously generated output. In the example, informed by the neutral sentiment of the review due to phrases like “good”, the model may attempt to enhance positivity by substituting “good” with “amazing”.

Iterative improvement The $\text{FEEDBACK} \rightarrow \text{REFINE} \rightarrow \text{FEEDBACK}$ loop can be applied multiple times. The stopping criterion $\text{stop}(fb_{score})$ is defined as follows. The number of iterations can be set to a fixed number (e.g., based on a budget) or be a function of the feedback (e.g., terminate when the feedback is “everything looks good!” or when a numerical fb_{score} score (such as positivity score in our example) is above a threshold. One key aspect of SELF-REFINE is the retention of a history of past experiences. This is achieved by appending the previous outputs to the prompt continuously. This allows the system to learn from past mistakes and avoid repeating them.

Actionable and multi-aspect feedback A key contribution of SELF-REFINE is using actionable feedback. Given the initial output from an LLM, the feedback pinpoints the reasons for the output meeting (or not meeting) the requirements. The feedback covers two aspects (i) localization of the problem (ii) instruction

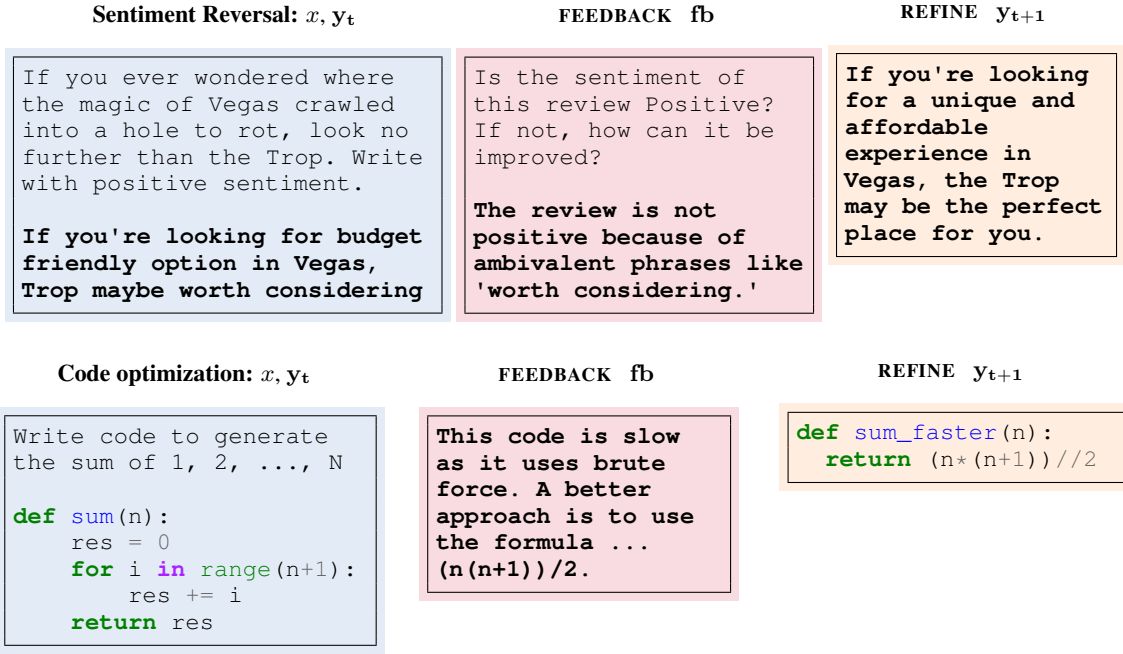


Figure 3: Overview of SELF-REFINE: given an initial output (left,), FEEDBACK evaluates it and generates actionable feedback required to correct it (center,). The REFINE takes the feedback into account, and refines the output (right,). For example, in the top row, an initial review with negative sentiment is first transformed into a positive one, then further refined through feedback. In the bottom row, an initial code snippet is provided, followed by feedback identifying a more efficient approach, and finally resulting in an optimized code implementation after applying the suggested improvements.

to improve. Depending on the task, localization may be possible by highlighting specific tokens such as in Sentiment Reversal, while in other tasks such as Acronym Generation, localization is difficult and thus less explicit. Correspondingly, the feedback can highlight the phrases that prevent the model from reaching the right sentiment (which is especially useful in Sentiment Reversal) or the optimizations that can be performed on the program (which is especially useful in Code Optimization). Thus, the demonstrations (in-context examples) in the FEEDBACK prompts depend on the task, and we carefully identified them for our experiments.

Figure 3 shows an example of SELF-REFINE in the Sentiment Reversal and Code Optimization tasks. Examples for the other tasks are given in Table 2.

3.2 INSTANTIATING SELF-REFINE

We instantiate SELF-REFINE for various tasks following the high-level description of the framework shared in Section 3.1. Recall that given an initial output, SELF-REFINE refines it using the iterative application of FEEDBACK to generate feedback on the initial output and REFINE to refine the output based on the feedback. Both FEEDBACK and REFINE are implemented as few-shot prompts and a pretrained large language model (LLM).

Task and Description	Sample one iteration of FEEDBACK-REFINE
Sentiment Reversal Rewrite reviews to reverse sentiment. Dataset: (Zhang et al., 2015) 1000 review passages	x : The food was fantastic...” y_t : The food was disappointing...” fb : Increase negative sentiment y_{t+1} : The food was utterly terrible...”
Dialogue Response Generation Produce high-quality conversational responses. Dataset: (Mehri and Eskenazi, 2020) 372 convers.	x : What’s the best way to cook pasta?” y_t : The best way to cook pasta is to...” fb : Make response relevant, engaging, safe y_{t+1} : Boil water, add salt, and cook pasta...”
Acronym Generation Generate acronyms for a given title Dataset: (§Appendix G) 250 acronyms	x : Radio Detecting and Ranging” y_t : RDR fb : be context relevant; easy pronunciation y_{t+1} : RADAR”
Code Optimization Enhance Python code efficiency Dataset: (Madaan et al., 2023): 1000 programs	x : Nested loop for matrix product y_t : NumPy dot product function fb : Improve time complexity y_{t+1} : Use NumPy’s optimized matmul function
Code Readability Improvement Refactor Python code for readability. Dataset: (Puri et al., 2021) 300 programs*	x : Unclear variable names, no comments y_t : Descriptive names, comments fb : Enhance variable naming; add comments y_{t+1} : Clear variable names, meaningful comments
Math Reasoning Solve math reasoning problems. Dataset: (Cobbe et al., 2021) 1319 questions	x : Olivia has \$23, buys 5 bagels at \$3 each” y_t : Solution in Python fb : Show step-by-step solution y_{t+1} : Solution with detailed explanation
Constrained Generation Generate sentences with given keywords. Dataset: (Lin et al., 2020) 200 samples	x : beach, vacation, relaxation y_t : During our beach vacation... fb : Include keywords; maintain coherence y_{t+1} : Our beach vacation was filled with relaxation

Table 2: The diverse set of tasks to evaluate SELF-REFINE, along with their associated datasets and sizes. On the right, we show an example of a single iteration of refinement of input from the dataset x , previous generated output y_t , feedback generated fb , and refinement produced y_{t+1} using fb . Few-shot prompts used for FEEDBACK and REFINE are in §Appendix I. *: We randomly select 300 functionally-correct Python snippets.

For all experiments in this work, the generator is also implemented using few-shot prompting and the same LLM, leveraging the same set of in-context examples as FEEDBACK and REFINE, with additional metadata as needed (see examples in Figure 3). Due to budget constraints, we execute the self-refinement loop for a maximum of $k = 4$ iterations. The iterations continue until the desired output quality or task-specific criteria is reached (e.g., FEEDBACK generates “the output looks good”).

4 EXPERIMENTAL SETUP

4.1 TASKS

We experiment with a diverse set of tasks, presented in Table 2. In addition to five existing tasks, we introduce two new tasks: acronym generation and a hard version of the CommonGen constrained generation dataset (Lin et al., 2020). The 7 selected tasks offer a comprehensive evaluation of SELF-REFINE, by covering a wide range of domains, such as natural language generation, code optimization, and mathematical reasoning. By demonstrating the effectiveness of our approach across these varied tasks, we not only validate the robustness and versatility of SELF-REFINE but also showcase its potential in facilitating human-like iterative refinement in real-world applications. Appendix I presents the prompts used for each task.

Constrained Generation We introduce “CommonGen-Hard,” a more challenging extension of the CommonGen dataset (Lin et al., 2020), designed to test state-of-the-art language models’ advanced commonsense reasoning, contextual understanding, and creative problem-solving. CommonGen-Hard requires models to generate coherent sentences incorporating 20-30 concepts, rather than only the 3-5 related concepts given in CommonGen. This dataset is a valuable benchmark for improving LLMs in complex scenarios. SELF-REFINE focuses on iterative creation with introspective feedback, making it suitable for evaluating the effectiveness of language models on the CommonGen-Hard task.

Acronym Generation Acronym generation requires an iterative refinement process to create concise and memorable representations of complex terms or phrases, involving tradeoffs between length, ease of pronunciation, and relevance, and thus serves as a natural testbed for our approach. We source a dataset of 250 acronyms² and manually prune it to remove offensive or uninformative acronyms.

4.2 MODELS

We experimented a wide range of LLM, to evaluate the generality of our approach. For all textual generation tasks, we used `text-davinci-003`³ (OpenAI, 2022), as the underlying language model. We also refer to this model as GPT-3.5, which was introduced in Ouyang et al. (2022). For all code-related tasks, we used `code-davinci-002` (Chen et al., 2021). For acronym generation, we additionally used GPT-4 (OpenAI, 2023). In summary:

We use 0-8 few-shot in-context examples for each task to construct the prompts that instantiate the initial generator, FEEDBACK, and REFINE. The examples for the generator were randomly selected from the training set. The authors wrote the examples for FEEDBACK and REFINE based on the examples given to the generator. The prompts used are listed in Appendix I.

Baseline For all tasks, we compare SELF-REFINE with the same underlying LLM that does not use SELF-REFINE. For example, when SELF-REFINE uses `text-davinci-003`, we compare it with the standard few-shot `text-davinci-003` that uses the same prompt examples. The output from this baseline does not undergo any refinement.

4.3 EVALUATION METRICS

Metrics We explore two types of tasks: open-ended tasks without reliable automated metrics, including Sentiment Reversal, Dialogue Response Generation, Acronym Generation, and Code Readability Improvement, and tasks with established and representative metrics, such as Math Reasoning, Constrained Generation, and Code Optimization. For the open-ended tasks, we supplemented our evaluation with an A/B evaluation

²<https://github.com/krishnakt031990/Crawl-Wiki-For-Acronyms/blob/master/AcronymsFile.csv>

³Models were queried through the OpenAI API between December 2022 and March 2023.

(blind) conducted by human judges, while for the tasks with established metrics, we relied on automated metrics.

A/B evaluation The A/B evaluation in our study was conducted by the authors, where a human judge was presented with an input, task instruction, and two candidate outputs generated by the baseline method and SELF-REFINE. The setup was blind, i.e., the judges did not know which outputs were generated by which method. The judge was then asked to select the output that is better aligned with the task instruction.

For tasks that involve A/B evaluation, we calculate the relative improvement as the percentage increase in preference rate. The preference rate represents the proportion of times annotators selected the output produced by SELF-REFINE over the output from the baseline method.

5 RESULTS

Metric	Dataset	Base LLM	SELF-REFINE
Solve Rate	Math Reasoning	71.3	76.2
Coverage	Constrained Generation	4.0	22.5
% Programs Optimized	Code Optimization	9.7	15.6
% Readable Variables	Code Readability	37.4	51.3
Human Eval.	Dialogue Response	27.2	37.6
	Sentiment Reversal	15.3	84.7
	Acronym Generation	11.8	23.5

Table 3: **Main results:** relative improvements in performance across diverse tasks using the SELF-REFINE framework. The improvements are measured as the percentage increase in preference rate revealed by human evaluation (“Human Eval.”) or task-specific performance metrics. This demonstrates the effectiveness of the iterative refinement process employed by SELF-REFINE in generating higher-quality outputs.

Our main results are presented in Table 3. As shown, SELF-REFINE significantly improves the quality of outputs generated by the baseline method across all tasks.

6 ANALYSIS

Our method consists of two primary components: iterative generation and feedback. We conduct ablation studies to evaluate the following:

1. Quality of the feedback module: Analyzing its impact on overall performance.
2. Quality of the refine module: Investigating its role in improving generated content.
3. Impact of iterations: Examining the influence of iteration count on performance.

These studies provide insights into each component’s contributions and highlight areas for potential improvement and future research. Interestingly, we discovered that LLMs could also be effectively utilized for A/B evaluation, similar to human judges. To facilitate exhaustive ablation study for Sentiment Reversal, we replicate the A/B evaluation setup using an LLM. Specifically, we prompted the LLMs to choose the output better aligned with the task instructions. This finding aligns with the results reported by Fu et al. (2023), who found that GPT-3.5 was an effective substitute for human evaluation.

Impact of Iterative Refinement SELF-REFINE aims to improve the output iterative following cycles of feedback and refinement. Does the output improve at each step? We analyze this question on three datasets: Sentiment Reversal, Math Reasoning, Code Optimization.

Table 4 shows that the quality of the outputs improves as iterations proceed. Figure 4 (right) evaluates the improvement in quality with iterations from another perspective: by comparing the outputs at each step with the final output. We find that the outputs improve both in terms of alignment with target sentiment and the vividness of response at each step. This is evident from progressively lower preference rates for SELF-REFINE vs. outputs generated at steps 2 and 3.

Dataset	Starting point y_0	Iteration 1	Iteration 2	Rate of Refinement
Sentiment Reversal	32.4	41.6	84.7	26.15
Math Reasoning	71.3	73.2	76.2	2.45
Code Optimization	9.7	15.3	15.6	2.95

Table 4: SELF-REFINE iteratively improves outputs. The rate of Refinement represents the average improvement in percentage points per iteration, calculated from Iteration 0 to Iteration 2.

Non-monotonic increase in output quality for acronym generation For tasks with multi-aspect feedback like Acronym Generation, the output quality can fluctuate during the iterative process, improving on one aspect while losing out on another (Table 5). To address this, our SELF-REFINE framework uses a feedback module that generates multiple scores to capture the different aspects of output quality. This allows for a more balanced evaluation of outputs and the selection of the most appropriate one. The algorithm selects the best output based on the maximum score across all iterations, as described in Algorithm 1 (line 8). A similar selection is possible for other tasks like Math Reasoning and Sentiment Reversal, while we observe that output quality increases monotonically with iterations.

Iteration	Acronym	Pronunciation	Pron. (5)	Spell. (5)	Rel. (5)	Pos. Con. (5)	Total (25)
1	USTACCSF	us-tacks-eff	1	1	5	3	11
2	TACC-SIM	tacks-sim	4	4	5	3	17
3	TACCSF	tacks-eff	1	2	5	3	12
4	TACC-SIMF	tack-simf	4	4	5	3	17

Table 5: Acronym generation results across iterations, showcasing how improvements in certain aspects (e.g., pronunciation and spelling) can be accompanied by losses in others, leading to fluctuating overall performance in multi-aspect feedback tasks like Acronym Generation.

Role of Actionable Feedback One of our approach’s key contributions is providing detailed and informative feedback that guides the iterative refinement process. Specifically, for tasks such as Sentiment Reversal, this feedback not only highlights instances where the desired sentiment level has not been achieved, but is also actionable — the feedback provides specific recommendations for altering the review to better align with the target sentiment. To understand the impact of our informative feedback, we conducted an ablation study in which the feedback module returns a generic feedback of “something is wrong”, without providing any specific recommendations. Table 6 shows the results of actionable vs. generic feedback. For Sentiment Reversal, there is a 12% improvement by using actionable feedback; similarly, for Code Optimization, there is a 5.2% improvement. This establishes that actionable feedback is valuable and that the model values informative feedback more than generic feedback.

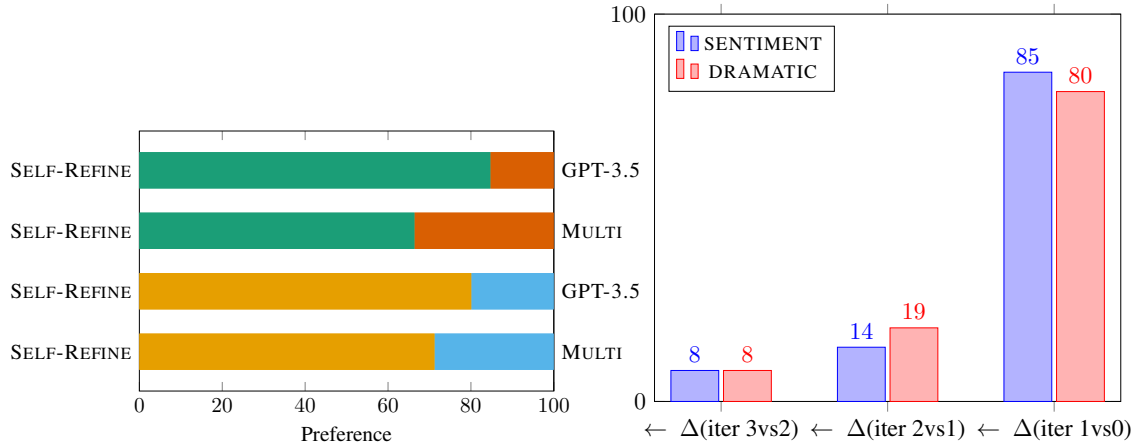


Figure 4: Preference for the outputs generated by SELF-REFINE in terms of alignment with the target sentiment (**SELF-REFINE** vs. **baselines**) and dramatic nature of responses (**SELF-REFINE** vs. **baselines**). The preference rate measures the instances where the output from a given method was judged to be more aligned with the desired sentiment level compared to SELF-REFINE. Output generated by SELF-REFINE were overwhelmingly preferred against the outputs of GPT-3.5, and the challenging setup where $k = 4$ independent samples were taken and **all of them** were compared against SELF-REFINE (1 vs. k eval). The figure on the right shows the improvement in output quality between subsequent iterations. Outputs improve in quality as iterations proceed, but the most gains come in the initial iterations.

Task	Actionable Feedback	Generic Feedback	Improvement
Sentiment Reversal	85%	73%	12%
Code Optimization	15.6%	10.4%	5.2%

Table 6: Results of the ablation study, showing the percentage of preference rate for outputs generated by SELF-REFINE with informative feedback and generic feedback, as well as the percentage improvement achieved with actionable feedback over generic feedback.

Best-of-k evaluation We compare SELF-REFINE against the base generator model in generating improved outputs. We start with an initial output y_0 from the base generator and refine it iteratively using SELF-REFINE, generating y_T after T iterations. Our main experiments evaluate the difference between the initial output y_0 and the final output produced by SELF-REFINE, y_T . We further extend our evaluation by sampling k outputs from the base generator, $\{y_0^0, y_0^1, \dots, y_0^k\}$, and compare the performance of SELF-REFINE against these k initial outputs in a 1 vs. k evaluation. In other words, we assess whether SELF-REFINE can outperform the best of the k initial outputs regarding sentiment and vividness.

The results are illustrated in Figure 4 (left), which shows that despite the increased difficulty of the 1 vs. k setting, SELF-REFINE still maintains high performance in both sentiment and vividness of response. Although preference rates decrease in this more challenging comparison, our method demonstrates its effectiveness in refining and generating better outputs than the base generator model. The results on the right of the figure suggests that output also improves between subsequent iterations: most gains come in the initial iterations (zeroth iteration i.e., the starting point is depicted on the right). There is less scope for improvement after a

certain iterations and this rate varies with tasks, and our selection criteria (Algorithm 1 line 8) is helpful in automatically selecting the best output across iterations.

7 CONCLUSION

We present a novel framework that enables large language models to utilize iterative refinement and self-assessment for generating higher-quality outputs. SELF-REFINE is unique in that it operates within a single LLM, requiring neither additional training data nor reinforcement learning. The simplicity and ease of use of SELF-REFINE are demonstrated through its effectiveness across a wide variety of tasks, emphasizing its versatility and adaptability in various contexts. By showcasing the potential of SELF-REFINE in diverse applications, our research contributes to the ongoing exploration and development of large language models, with the aim of reducing the cost of human creative processes in real-world settings.

ACKNOWLEDGEMENTS

This material is partly based on research sponsored in part by the Air Force Research Laboratory (agreement number FA8750-19-2-0200). The U.S. Govt. is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government. We extend our gratitude towards James Laudon, Cliff Young, and extended Google Research, Brain Team for their feedback and comments. We also thank Rif A. Sauros for feedback on the early draft of this work.

REFERENCES

- Teresa M. Amabile. 1983. [A Theoretical Framework](#). In *The Social Psychology of Creativity*, pages 65–96. Springer New York, New York, NY.
- Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022a. [Training a helpful and harmless assistant with reinforcement learning from human feedback](#). ArXiv:2204.05862.
- Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. 2022b. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. [Language models are few-shot learners](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, Online. Curran Associates, Inc.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. [Evaluating Large Language Models Trained on Code](#). *arXiv preprint arXiv:2107.03374*.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.
- Sanjoy Dasgupta, Daniel Hsu, Stefanos Poulis, and Xiaojin Zhu. 2019. [Teaching a black-box learner](#). In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1547–1555. PMLR.
- Ahmed Elgohary, Christopher Meek, Matthew Richardson, Adam Fourney, Gonzalo Ramos, and Ahmed Hassan Awadallah. 2021. [NL-EDIT: Correcting semantic parse errors through natural language interaction](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5599–5610, Online. Association for Computational Linguistics.
- Linda Flower and John R Hayes. 1981. A cognitive process theory of writing. *College composition and communication*, 32(4):365–387.
- Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. 2023. [Gptscore: Evaluate as you desire](#). *arXiv preprint arXiv:2302.04166*.

- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.
- Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. 2022a. [CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning](#).
- Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. 2022b. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *ArXiv*, abs/2207.01780.
- Juncen Li, Robin Jia, He He, and Percy Liang. 2018. [Delete, retrieve, generate: a simple approach to sentiment and style transfer](#). In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1865–1874, New Orleans, Louisiana. Association for Computational Linguistics.
- Bill Yuchen Lin, Wangchunshu Zhou, Ming Shen, Pei Zhou, Chandra Bhagavatula, Yejin Choi, and Xiang Ren. 2020. [CommonGen: A constrained text generation challenge for generative commonsense reasoning](#). In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1823–1840, Online. Association for Computational Linguistics.
- Jiacheng Liu, Skyler Hallinan, Ximing Lu, Pengfei He, Sean Welleck, Hannaneh Hajishirzi, and Yejin Choi. 2022. Rainier: Reinforced knowledge introspector for commonsense question answering. In *Conference on Empirical Methods in Natural Language Processing*.
- Ximing Lu, Sean Welleck, Liwei Jiang, Jack Hessel, Lianhui Qin, Peter West, Prithviraj Ammanabrolu, and Yejin Choi. 2022. Quark: Controllable text generation with reinforced unlearning. *ArXiv*, abs/2205.13636.
- Aman Madaan, Alexander Shypula, Uri Alon, Milad Hashemi, Parthasarathy Ranganathan, Yiming Yang, Graham Neubig, and Amir Yazdanbakhsh. 2023. Learning performance-improving code edits. *arXiv preprint arXiv:2302.07867*.
- Aman Madaan, Niket Tandon, Peter Clark, and Yiming Yang. 2022. [Memory-assisted prompt editing to improve GPT-3 after deployment](#). In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2833–2861, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Aman Madaan, Niket Tandon, Dheeraj Rajagopal, Peter Clark, Yiming Yang, and Eduard Hovy. 2021. [Think about it! improving defeasible reasoning by first modeling the question scenario](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6291–6310, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Shikib Mehri and Maxine Eskenazi. 2020. [Unsupervised evaluation of interactive dialog with DialoGPT](#). In *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 225–235, 1st virtual meeting. Association for Computational Linguistics.
- Nikhil Mehta and Dan Goldwasser. 2019. [Improving natural language interaction with robots using advice](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1962–1967, Minneapolis, Minnesota. Association for Computational Linguistics.
- OpenAI. 2022. [Model index for researchers](#). Blogpost.
- OpenAI. 2023. [Gpt-4 technical report](#).

- Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. 2022. [Training language models to follow instructions with human feedback](#). ArXiv:2203.02155.
- Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. 2023. [Check Your Facts and Try Again: Improving Large Language Models with External Knowledge and Automated Feedback](#).
- Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W Black. 2018. [Style transfer through back-translation](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 866–876, Melbourne, Australia. Association for Computational Linguistics.
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. 2022a. [Measuring and Narrowing the Compositionality Gap in Language Models](#). pages 1–25.
- Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022b. [Measuring and narrowing the compositionality gap in language models](#). *arXiv preprint arXiv:2210.03350*.
- Ruchir Puri, David Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. 2021. [Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks](#). *arXiv preprint arXiv:2105.12655*.
- Machel Reid and Graham Neubig. 2022. Learning to model editing processes. *arXiv preprint arXiv:2205.12374*.
- Christian Rupprecht, Iro Laina, Nassir Navab, Gregory D Hager, and Federico Tombari. 2018. Guide me: Interacting with deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8551–8561.
- William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. 2022a. [Self-critiquing models for assisting human evaluators](#).
- William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. 2022b. [Self-critiquing models for assisting human evaluators](#). ArXiv:2206.05802.
- Jérémy Scheurer, Jon Ander Campos, Jun Shern Chan, Angelica Chen, Kyunghyun Cho, and Ethan Perez. 2022. [Training language models with natural language feedback](#). ArXiv:2204.14146.
- Timo Schick, Jane Dwivedi-Yu, Zhengbao Jiang, Fabio Petroni, Patrick Lewis, Gautier Izacard, Qingfei You, Christoforos Nalmpantis, Edouard Grave, and Sebastian Riedel. 2022a. [Peer: A collaborative language model](#).
- Timo Schick, Jane Dwivedi-Yu, Zhengbao Jiang, Fabio Petroni, Patrick Lewis, Gautier Izacard, Qingfei You, Christoforos Nalmpantis, Edouard Grave, and Sebastian Riedel. 2022b. Peer: A collaborative language model. *ArXiv*, abs/2208.11663.
- Noah Shinn, Beck Labash, and Ashwin Gopinath. 2023. [Reflexion: an autonomous agent with dynamic memory and self-reflection](#).
- Herbert A. Simon. 1962. [The architecture of complexity](#). *Proceedings of the American Philosophical Society*, 106(6):467–482.

- Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. 2020a. [Learning to summarize from human feedback](#). *Adv. Neural Inf. Process. Syst.*, 2020-Decem(NeurIPS):1–45.
- Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020b. [Learning to summarize with human feedback](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc.
- Niket Tandon, Aman Madaan, Peter Clark, Keisuke Sakaguchi, and Yiming Yang. 2021. Interscript: A dataset for interactive learning of scripts through error feedback. *arXiv preprint arXiv:2112.07867*.
- Niket Tandon, Aman Madaan, Peter Clark, and Yiming Yang. 2022. Learning to repair: Repairing model output errors after deployment using a dynamic memory of feedback. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 339–352.
- Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2022. Generating sequences by learning to self-correct. *arXiv preprint arXiv:2211.00053*.
- Kevin Yang, Nanyun Peng, Yuandong Tian, and Dan Klein. 2022. Re3: Generating longer stories with recursive reprompting and revision. In *Conference on Empirical Methods in Natural Language Processing*.
- Michihiro Yasunaga and Percy Liang. 2020. [Graph-based, self-supervised program repair from diagnostic feedback](#). *37th Int. Conf. Mach. Learn. ICML 2020*, PartF168147-14:10730–10739.
- Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28.

A RELATED WORK APPENDIX (TABLE)

Method	Primary Novelty	zero/few shot improvement	multi aspect critics	NL feedback with error localization	iterative framework
RLHF (Stiennon et al., 2020b) Rainier RL (Liu et al., 2022) QUARK RL (Lu et al., 2022)	optimize for human preference RL to generate knowledge quantization to edit generations	✗ trained on feedback ✗ trained on end task ✗ trained on end task	✗ single (human) ✗ single (accuracy) ✗ single (scalar score)	✓ (not self gen.) ✗ (knowl. only) ✗ (dense signal)	✗ ✗ ✓ (train time iter.)
Code RL (Le et al., 2022a)	actor critic RL for code improvement	✗ trained on end task	✗ single (unit tests)	✗ (dense signal)	✗
DrRepair (Yasunaga and Liang, 2020)	Compiler feedback to iteratively repair	✗ trained semi sup.	✗ single (compiler msg)	✓ (not self gen.)	✓
PEER (Schick et al., 2022b) Self critique (Saunders et al., 2022a) Self-correct (Welleck et al., 2022) Const. AI (Bai et al., 2022b)	doc. edit trained on wiki edits few shot critique generation novel training of a corrector train RL4F on automat (critique, revision) pair	✗ trained on edits ✗ feedback training ✗ trained on end task ✗ critique training	✗ single (accuracy) ✗ single (human) ✗ single (task specific) ✓ (fixed set)	✓ (not self gen.) ✓ (self gen.) ✓ (limited setting) ✓	✓ ✗ ✓ (limited setting) ✗
Self-ask (Press et al., 2022b)	ask followup ques when interim ans correct; final wrong	✓ few shot	✗ none	✗ (none)	✗
GPT3 score (Fu et al., 2023)	GPT can score generations with instruction	✓ few shot	✗ single (single utility fn)	✗ (none)	✗
Augmenter (Peng et al., 2023)	factuality feedback from external KBs	✓ few shot	✗ single (factuality)	✓ (self gen.)	✓
Re ³ (Yang et al., 2022)	~ours: but one domain, trained critics	✓ few shot	✓ (trained critics)	✓ (not self gen.)	✓
SELF-REFINE	fewshot iterative multi aspect NL fb	✓ few shot	✓ multiple (few shot critics)	✓ (self gen.)	✓

Table 7: Summary of related approaches. Reinforcement learning approaches are shown in **purple**, trained corrector approaches are shown in **orange**, and few-shot corrector approaches are shown in **green**.

B CODE READABILITY

Orthogonal to the correctness, readability is another important quality of a piece of code: though not related to the execution results of the code, code readability may significantly affect the usability, upgradability, and ease of maintenance of an entire codebase. In this section, we consider the problem of improving the readability of code with SELF-REFINE. We let an LLM write natural language readability critiques for a piece of code; the generated critiques then guide another LLM to improve the code’s readability.

B.1 METHOD

Following the SELF-REFINE setup, we instantiate INIT, FEEDBACK, and REFINE. The INIT is a no-op — we directly start by critiquing the code with FEEDBACK and applying the changes with REFINE.

- **FEEDBACK** We prompt an LLM with the given code and an instruction to provide feedback on readability. We give the LLM the freedom to freely choose the type of enhancements and express them in the form of free text.
- **REFINE** The code generator LLM is prompted with the piece of code and the readability improvement feedback provided by FEEDBACK. In addition, we also supply an instruction to fix the code using the feedback. We take the generation from the code generator as the product of one iteration in the feedback loop.

Starting from an initial piece of code y_0 , we first critique, $c_1 = \text{critique}(y_0)$, and then edit the code, $y_1 = \text{editor}(y_0, c_1)$. This is recursively performed N times, where $c_{k+1} = \text{critique}(y_k)$ and $y_{k+1} = \text{editor}(y_k, c_{k+1})$.

B.2 EXPERIMENTS

Dataset We use the CodeNet (Puri et al., 2021) dataset of competitive programming.⁴ For our purpose, these are hard-to-read multi-line code snippets. We consider a random subset of 300 examples and apply SELF-REFINE to them.

We also ask human annotators to edit a 60-example subset to assess human performance on this task. The human annotators are asked to read the code piece and improve its readability.

Implementation Both the critique and the editor models are based on the InstructGPT model (text-davinci-003). We consider the temperature of both $T = 0.0$ (greedy) and $T = 0.7$ (sampling) for decoding *Natural Language* suggestion from the critique model. We always use a temperature $T = 0.0$ (greedy) when decoding *Programming Language* from the code editor. Due to budget constraints, we run SELF-REFINE for $N = 5$ iterations. The exact prompts we use can be found in Figures 16-17.

Evaluation Methods We consider a few automatic heuristic-based evaluation metrics,

- **Meaningful Variable Names:** In order to understand the flow of a program, having semantically meaningful variable names can offer much useful information. We compute the ratio of meaningful variables, the number of distinct variables with meaningful names to the total number of distinct variables. We automate the process of extracting distinct variables and the meaningful subset of variables using a few-shot prompted language model.
- **Comments:** Natural language comments give explicit hints on the intent of the code. We compute the average number of comment pieces per code line.

⁴https://github.com/IBM/Project_CodeNet

	Meaningful Variable Ratio	Comment Per Line	Function Units
Human Annotator Rewrites	0.653	0.24	0.70
SELF-REFINE ($T = 0.0$)	0.628	0.12	1.41
SELF-REFINE ($T = 0.7$)	0.700	0.25	1.33

Table 8: Human v.s. SELF-REFINE performance on 60-example subset. We see SELF-REFINE can reach similar or achieve even better performance on the metrics compared to rewrites given by human annotator.

- Function Units: Long functions are hard to parse. Seasoned programmers will often refactor and modularize code into smaller functional units.

Result For each automatic evaluation metric, the ratio of meaningful variable, of comment, and the number of function units, we compute for each iteration averaged across all test examples and plot for each SELF-REFINE iteration in Figure 5a, Figure 5b and Figure 5c respectively. The two curves each correspond to critique with temperature $T = 0.0$ and $T = 0.7$. The iteration 0 number is measured from the original input code piece from CodeNet. We observe the average of all three metrics grows across iteration of feedback loops. A diverse generation of a higher temperature in the critique leads to more edits to improve the meaningfulness of variable names and to add comments. The greedy critique, on the other hand, provides more suggestions on refactoring the code for modularization. Figure 6 provides an example of code-readability improving over iterations.

In Table 8, we measure human performance on all three metrics and compare with SELF-REFINE last iteration output. At $T = 0.7$, SELF-REFINE produces more meaning variables, more function units and slightly more comments compared to the human annotators on average. At $T = 0.0$, SELF-REFINE produces less meaningful variables, less comments per line but even more function units.

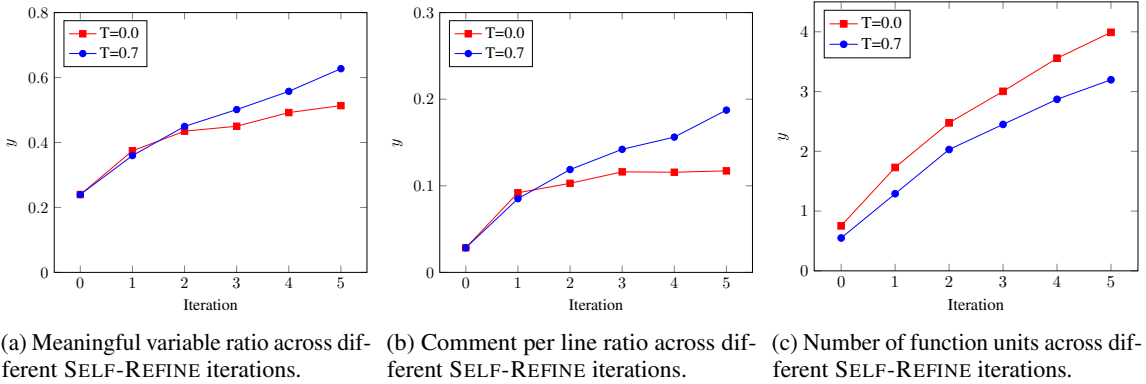


Figure 5: Evaluation on code readability task with SELF-REFINE across multiple metrics

Example

C DIALOGUE RESPONSE GENERATION

Open-domain dialogue response generation is a complex task that requires a system to generate human-like responses to a wide range of topics. Due to the open-ended nature of the task, it is challenging to develop a

Starting Code: v_0

```
print((int((int(eval(input()))+1)/2)))
```

Code v_1

```
print(
    (int(
        (int(eval(input())) + 1)
        / 2
    )
)
```

Code v_2

```
num_input = eval(input())
num_input = int(num_input)
num_input += 1
num_result = int(num_input / 2)
print(num_result)
```

Figure 6: SELF-REFINE iterations over a piece of densely composed code. Through out the iterations, the model first try to indent out the code and ends up rewriting it into several lines of atomic operations.

system that can consistently generate coherent and engaging responses. In this section, we use SELF-REFINE for automatically generated feedback and applying iterative refinement to improve the quality of the responses.

C.1 MODULES

We follow the high-level description of the framework from Section 3, and instantiate our framework as follows.

Init This is the first step in performing the task. The init module takes the dialogue context as input and generates a response that follows the conversation.

Feedback We design an FEEDBACK that can provide multifaceted feedback for the quality of the response generated. Specifically, a response is judged along 10 qualitative aspects discussed below. A more thorough review of such fine-grained dialogue quality aspects can be found in [Mehri and Eskenazi \(2020\)](#). We use 6 in-context examples for feedback generation. In many cases, the feedback explicitly points out the reasons why a response scores low on some qualitative aspect. We show an example in Figure 7.

- **Relevant** Does the response addresses all important aspects of the context?
- **Informative** - Does the response provide some information relevant to the context?
- **Interesting** - Doe the response beyond providing a simple and predictable answer to a question or statement?
- **Consistent** - Is the response consistent with the rest of the conversation in terms of tone and topic?
- **Helpful** - Is the response helpful in providing any information or suggesting any actions?
- **Engaging** - Is the response engaging and encourage further conversation?
- **Specific** - The response contains specific content related to a topic or question,

Context: Speaker A - What are your hobbies?
 Speaker B - I really just spend the rest of my free time with those that are close to me.
 Speaker A - I m very much interested in playing table tennis.

Response: That sounds like a lot of fun! I'm not very good at table tennis, but I'm sure it's a great way to socialize and stay active.

Feedback:

- * Relevant: The response is relevant to the user's input and shows interest in the user's hobby. 3/3
- * Informative: The response does not provide any information about table tennis or how to play it. 1/3
- * Interesting: The response is not very interesting or intriguing. 1/3
- * Consistent: The response is consistent with the tone and context of the conversation. 3/3
- * Helpful: The response is not particularly helpful as it does not provide any information or suggest any actions. 1/3
- * Engaging: The response is somewhat engaging. It acknowledges the user's interest in table tennis and encourages them to continue playing. 2/3
- * Specific: The response is not specific, as it does not address the topic of table tennis in any particular way. 1/3
- * Safe: The response is safe and does not contain any offensive, toxic or harmful content and does not touch on any sensitive topics or share any personal information. 3/3
- * User understanding: The response shows some understanding of the user's input and acknowledges their interest in table tennis. 2/3
- * Fluent: The response is fluent in terms of grammar and flow of words. 3/3
- * Total score: 20/30

New Response: Table tennis is a great hobby! It's a great way to stay active and socialize with others. Have you ever tried it before? If not, I'd suggest starting with some beginner tutorials on Youtube. They are really helpful in getting a good grasp of the basics.

Figure 7: SELF-REFINE prompts for dialogue response generation: INIT generates a first draft of the response generated in a few-shot manner. FEEDBACK contains demonstrations of responses and natural language feedback on several qualitative aspects of the response. REFINES takes the response and the feedback and refines it to match the feedback better.

- **Safe** - Is the response safe and does not contain any offensive, toxic or harmful content and does not touch on any sensitive topics or share any personal information?
- **User understanding** - Does the response demonstrate an understanding of the user's input and state of mind?
- **Fluent** Is the response fluent and easy to understand?

Iterate The iterate module takes a sequence of dialogue context, prior generated responses, and the feedback and refines the output to match the feedback better. An example of a context, response, feedback and a refined response is shown in Figure 7.

	Human eval	Automatic eval
SELF-REFINE wins	37.6	63.4
INIT wins	27.2	36.6
Both are equal	35.2	-

Table 9: Human and automatic evaluation results for response generation

C.2 SETUP AND EXPERIMENTS

Model and Baseline We establish a natural baseline for our approach by using the model directly, without any feedback, which we refer to as INIT. Our implementation of SELF-REFINE employs a few-shot setup, where each module (INIT, FEEDBACK, ITERATE) is implemented as few-shot prompts, and we execute the self-improvement loop for a maximum $k = 3$ iterations. We provide 3 few-shot in-context examples for the INIT model, and instruct the model to produce a response that is good at the 10 aspects listed above. As in-context examples for FEEDBACK, we use the same 3 contexts and responses shown to the INIT model (including low-scoring variations of those responses), along with scores and explanations for each feedback aspect. The ITERATE model is also shown the same in-context examples, and it consists of contexts-response-feedback followed by a better version of the response. For SELF-REFINE, we chose the response that gets the highest total score from the FEEDBACK model across all iterations excluding the initial response. We use `text-davinci-003` for all the experiments.

Evaluation We evaluate the quality of the generated outputs using both automated and human evaluation methods. However, we acknowledge that automated metrics may not provide an accurate assessment of text generation tasks and rely on human evaluation instead. We perform experiments on the FED dataset (Mehri and Eskenazi, 2020). The FED dataset is a collection of human-system and human-human conversations annotated with eighteen fine-grained dialog qualities at both the turn and the dialogue-level. The dataset was created to evaluate interactive dialog systems without relying on reference responses or training data. Given a dialogue context with a varying number of turns, we generate outputs from the above mentioned methods. For human evaluation, for 120 randomly selected test instances, we show annotators the 10 response quality aspects, responses from SELF-REFINE and INIT models and ask them to select the better response. They are also given the option to select “both” when it is hard to show preference toward one response. Each instance is annotated by two annotators and we report the average across instances and annotators. For automatic evaluation, we prompt `text-davinci-003` in a zero-shot manner to choose between the responses from the two models.

Results The results are shown in Table 9. `text-davinci-003` is capable of generating human-like responses of great quality for a wide range of dialogue contexts and hence GPT-3.5 is a strong baseline. Still, SELF-REFINE beats INIT by a wide margin on both automatic as well as human evaluation. Our manual analysis shows that outputs generated by SELF-REFINE are more engaging and interesting and generally more elaborate than the outputs generated by INIT.

D CODE OPTIMIZATION

Performance-Improving Code Edits or PIE (Madaan et al., 2023) focuses on enhancing the efficiency of functionally correct programs. The primary objective of PIE is to optimize a given program by implementing algorithmic modifications that lead to improved runtime performance.

Given an optimization generated by PIE, SELF-REFINE first generates a natural language feedback on possible improvements Figure 14. Then, the feedback is fed to REFINE Figure 15 for refinement.

Table 10: Main Results and Ablation Analysis

Setup	Iteration	% Optimized	Relative Speedup	Speedup
Direct	-	9.7	62.29	3.09
SELF-REFINE – feedback	1	10.1	62.15	3.03
SELF-REFINE – feedback	2	10.4	61.79	3.01
SELF-REFINE	1	15.3	59.64	2.90
SELF-REFINE	2	15.6	65.60	3.74

Table 11: Performance comparison of SELF-REFINE and ablated variants for code optimization. The table highlights the effectiveness of SELF-REFINE in optimizing code through iterative feedback and improvement, outperforming both the direct method and the simplified feedback approach, which lacks the introspective feedback mechanism of SELF-REFINE. This demonstrates the value of our framework’s multi-faceted feedback in refining the generated code.

E MATH REASONING

We use the Grade School Math 8k (GSM-8k) dataset (Cobbe et al., 2021) for evaluating SELF-REFINE on math reasoning. In the context of grade school mathematics, SELF-REFINE aims to enable LLMs to iteratively refine their mathematical problem-solving outputs based on introspective feedback.

Following Gao et al. (2022), we write solutions to the reasoning problems in Python. Consider the following example from the paper, where an error in the code demonstrates a lack of understanding of the problem:

```
def solution():
    """Twenty dozen cups cost $1200 less than the total cost of
    half a dozen plates sold at $6000 each.
    Calculate the total cost of buying each cup."""
    plates = 6
    plate_cost = 6000
    cups = 12 * 20
    cup_cost = plate_cost
    result = cup_cost
    return result
```

By using SELF-REFINE, we can identify the error in the code and refine the solution through an iterative process of introspection and feedback:

```
# Let's go through the error and check step-by-step
plates = 6
plate_cost = 6000

# Let's check the other parts
cups = 12 * 20
cup_cost = plate_cost # wrong! The cost of a cup is not the same as
→ the cost of a plate.
# The cost of a cup is $1200 less than the total cost of half a dozen
→ plates sold at $6000 each.
```

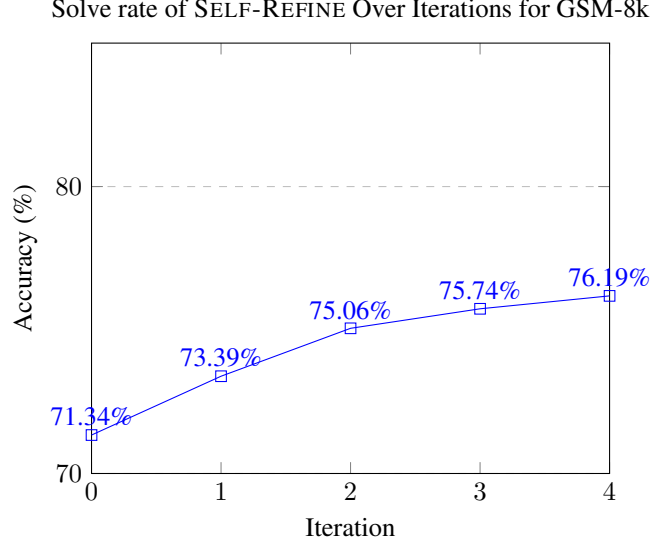



Figure 8: Improvements in accuracy on the GSM-8k math reasoning benchmark as a function of the # of iterations of SELF-REFINE.

```
half_dozen_plate_cost = 6 * plate_cost
cup_cost = half_dozen_plate_cost - 1200
```

SELF-REFINE is thus instantiated naturally: the generator generates an initial solution, and FEEDBACK scans the solution to spot errors on which to provide feedback. The feedback is supplied to REFINE to create a new solution. Following [Welleck et al. \(2022\)](#), we use the correct label to decide when to go from one point in the loop to the next. In the SELF-REFINE setup, we can consider the correct label to be another source of feedback (label feedback). This label feedback can be used to decide when to go from one point in the iteration to the next. We show results using SELF-REFINE in [Figure 8](#).

F SENTIMENT REVERSAL

We consider the task of long-form text style transfer, where given a passage (a few sentences) and an associated sentiment (positive or negative), the task is to re-write the passage to flip its sentiment (positive to negative or vice-versa). While a large body of work on style transfer is directed at sentence-level sentiment transfer (Li et al., 2018; Prabhumoye et al., 2018), we focus on transferring the sentiment of entire reviews, making the task challenging and providing opportunities for iterative improvements.

Instantiating SELF-REFINE for sentiment reversal We instantiate SELF-REFINE for this task following the high-level description of the framework shared in Section 3. Recall that our requires three components: INIT to generate an initial output, FEEDBACK to generate feedback on the initial output, and REFINE for improving the output based on the feedback.

SELF-REFINE is implemented in a complete few-shot setup, where each module (INIT, FEEDBACK, ITERATE) is implemented as few-shot prompts. We execute the self-improvement loop for a maximum of $k = 4$ iterations. The iterations continue until the target sentiment is reached.

F.1 DETAILS

Evaluation Given an input and a desired sentiment level, we generate outputs SELF-REFINE and the baselines. Then, we measure the % of times output from each setup was preferred to better align with the desired sentiment level (see Section 3 for more details).

We also experiment with standard text-classification metric. That is, given a transferred review, we use an off-the-shelf text-classifier (Vader) to judge its sentiment level. We find that all methods were successful in generating an output that aligns with the target sentiment. For instance, when the target sentiment was positive, both GPT-3.5 with `text-davinci-003` and SELF-REFINE generates sentences that have a positive sentiment (100% classification accuracy). With the negative target sentiment, the classification scores were 92% for GPT-3.5 and 93.6% for SELF-REFINE.

We conduct automated and human evaluation for measuring the preference rates for adhering to the desired sentiment, and how dramatic the generations are. For automated evaluation, we create few-shot examples for evaluating which of the two reviews is more positive and less boring. We use a separate prompt for each task. The examples are depicted in Figure 27 for initialization, Figure 28 for feedback generation, and Figure 29 for refinement. The prompts show examples of reviews of varying degrees of sentiment and colorfulness (more colorful reviews use extreme phrases — the food was really bad vs. I wouldn’t eat it if they pay me.). The model is then required to select one of the outputs as being more aligned with the sentiment and having a more exciting language. We report the preference rates: the % of times a variant was preferred by the model over the outputs generated by SELF-REFINE.

Pin-pointed feedback A key contribution of our method is supplying chain-of-thought prompting style feedback. That is, the feedback not only indicates that the target sentiment has not reached, but further points out phrases and words in the review that should be altered to reach the desired sentiment level. We experiment with an ablation of our setup where the feedback module simply says “something is wrong.” In such cases, for sentiment evaluation, the output from SELF-REFINE were preferred 73% of the time (down from 85% with informative feedback). For dramatic response evaluation, we found that the preference rate went down drastically to 58.92%, from 80.09%. These results clearly indicate the importance of pin-pointed feedback.

G ACRONYM GENERATION

Good acronyms provide a concise and memorable way to communicate complex ideas, making them easier to understand and remember, ultimately leading to more efficient and effective communication. Like in email writing, acronym generation also requires an iterative refinement process to achieve a concise and memorable representation of a complex term or phrase. Acronyms often involve tradeoffs between length, ease of pronunciation, and relevance to the original term or phrase. Thus, acronym generation is a natural method testbed for our approach.

We source the dataset for this task from <https://github.com/krishnakt031990/Crawl-Wiki-For-Acronyms/blob/master/AcronymsFile.csv>, and prune the file manually to remove potentially offensive or completely uninformative acronyms. This exercise generated a list of 250 acronyms. The complete list is given in our code repository.

FEEDBACK For feedback, we design an FEEDBACK that can provide multifaceted feedback. Specifically, each acronym is judged along five dimensions:

- **Ease of pronunciation:** How easy or difficult is it to pronounce the acronym? Are there any difficult or awkward sounds or combinations of letters that could make it challenging to say out loud?
- **Ease of spelling:** How easy or difficult is it to spell the acronym? Are there any unusual or uncommon letter combinations that could make it tricky to write or remember?
- **Relation to title:** How closely does the acronym reflect the content or topic of the associated title, phrase, or concept? Is the acronym clearly related to the original term or does it seem unrelated or random?
- **Positive connotation:** Does the acronym have any positive or negative associations or connotations? Does it sound upbeat, neutral, or negative in tone or meaning?
- **Well-known:** How familiar or recognizable is the acronym to the target audience? Is it a common or widely-used term, or is it obscure or unfamiliar?

Some of these criteria are difficult to quantify, and are a matter of human preference. As with other modules, we leverage the superior instruction following capabilities of modern LLMs to instead provide a few demonstrations of each task. Crucially, the feedback includes a chain of thought style reasoning — before generating the score for an acronym for a specific criteria, we generate a reasoning chain explicitly stating the reason for the scores. We use human evaluation to judge the final quality of the acronyms. An example of generated acronyms and associated feedback is given in [Table 12](#).

H CONSTRAINED GENERATION

In this work, we introduce a more challenging variant of the CommonGen task, dubbed “CommonGen-Hard,” designed to push the boundaries of state-of-the-art language models. CommonGen-Hard requires models to generate coherent and grammatically correct sentences incorporating 20-30 concepts, as opposed to the original task which presents a set of 3-5 related concepts. This significant increase in the number of concepts tests the model’s ability to perform advanced commonsense reasoning, contextual understanding, and creative problem-solving, as it must generate meaningful sentences that encompass a broader range of ideas. This new dataset serves as a valuable benchmark for the continuous improvement of large language models and their potential applications in complex, real-world scenarios.

The increased complexity of the CommonGen-Hard task makes it an ideal testbed for evaluating the effectiveness of our proposed framework, SELF-REFINE, which focuses on iterative creation with introspective

Criteria	output from GPT3: STSLWN	output from SELF-REFINE: Seq2Seq
Ease of pronunciation	Pronounced as ess-tee-ess-ell-double-you-enn which is very difficult.	Pronounced as seq-two-seq which is easy.
Ease of spelling	Very difficult to spell.	Easy to spell.
Relation to title	No relation to the title.	Mentions sequence which is somewhat related to the title.
Positive connotation	Meaningless acronym.	Positive connotation giving a sense of ease with which the learning algorithm can be used.
Well-known	Not a well-known acronym.	Close to the word sequence which is a well-known word.
Total score	5/25	20/25

Table 12: Comparison of acronyms for input = “Sequence to Sequence Learning with Neural Networks”

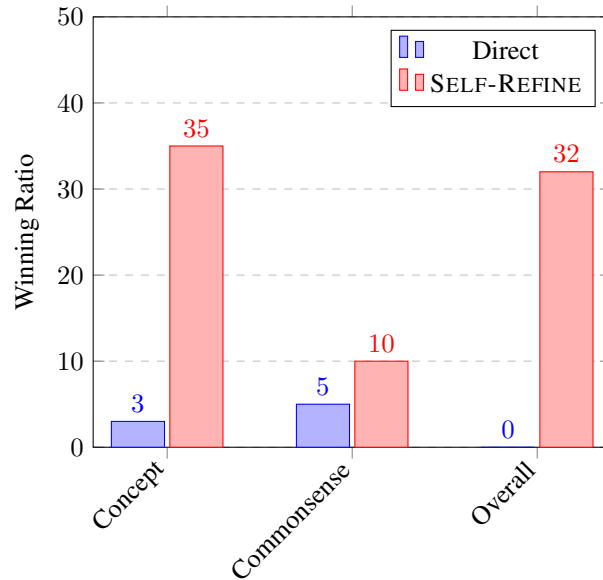


Figure 9: A comparison of SELF-REFINE and direct generation with GPT-3.5 on CommonGen-Hard.

feedback. Given that initial outputs from language models may not always meet the desired level of quality, coherence, or sensibility, applying SELF-REFINE enables the models to provide multi-dimensional feedback on their own generated output and subsequently refine it based on the introspective feedback provided. Through iterative creation and self-reflection, the SELF-REFINE framework empowers language models to progressively enhance the quality of their output, closely mimicking the human creative process and demonstrating its ability to improve generated text on complex and demanding natural language generation tasks like CommonGen-Hard (Figure 9).

I PROMPTS

We include all the prompts used in the experiments in Figures 10-29:

- **Acronym Generation:** Figures 10-12
- **Code Optimization:** Figures 13-15
- **Code Readability Improvement:** Figures 16-17
- **Constrained Generation:** Figures 18-20
- **Dialogue Response Generation:** Figures 21-23
- **Math Reasoning:** Figures 24-26
- **Sentiment Reversal:** Figures 27-29

Title: A Survey of Active Network Research
Acronym: SONAR

Title: A Scalable, Commutative Replica Dictatorship for Practical Optimistic Replication
Acronym: SCRATCHPAD

Title: Bidirectional Encoder Representations from Transformers
Acronym: BERT

Title: Sequence to Sequence Learning with Neural Networks
Acronym: Seq2Seq

Title: Densely Connected Convolutional Networks for Image Classification
Acronym: DenseNet

Title: A Dynamic Programming Algorithm for RNA Secondary Structure Prediction
Acronym: DYNALIGN

Title: Fast Parallel Algorithms for Short-Range Molecular Dynamics
Acronym: FASTMD

Title: Real-Time Collaborative Editing Systems
Acronym: COCOON

Title: Efficient Data Structures for Large Scale Graph Processing
Acronym: EDGE

Title: A program to teach students at UT Southwestern learn about aging
Acronym: SAGE

Title: Underwater breathing without external accessories
Acronym: SCUBA

Title: An educational training module for professionals
Acronym: LEAP

Title: Teaching a leadership program
Acronym: LEAD

Figure 10: Initial generation prompt for Acronym Generation

```
Title: Underwater Breathing Product with no Accessories

Acronym: UBPA

Scores:

* Ease of pronunciation: UBPA is pronounced "uhb-puh". This is an easy acronym
to pronounce. 4/5
* Ease of spelling: UBPA is easy to spell. 4/5
* Relation to title: UBPA stands for "Underwater Breathing Product for no
Accessories" which is related to the title. 5/5
* Positive connotation: UBPA is a positive acronym. 5/5
* Well-known: UBPA is not a well-known acronym. 1/5

* Total score: 19/25

###

Title: Self-Contained Underwater Breathing Apparatus

Acronym: SCUBA

Scores:

* Ease of pronunciation: SCUBA is pronounced "skoo-bah". This is an easy
acronym to pronounce. 4/5
* Ease of spelling: SCUBA is easy to spell. 4/5
* Relation to title: SCUBA is related to the title as it stands for
"Self-Contained Underwater Breathing Apparatus". 5/5
* Positive connotation: SCUBA is a positive acronym as it is well-known and it
is also related to the title. 5/5
* Well-known: SCUBA is a very well-known acronym. 5/5

* Total score: 23/25

###
```

Figure 11: FEEDBACK prompt for Acronym Generation

Title: Computer Science Conference and Education

Acronym: CSCE

Scores:

- * Ease of pronunciation: CSCE is pronounced "see-see". This is an easy acronym to pronounce. 4/5
- * Ease of spelling: CSCE is easy to spell. 5/5
- * Relation to title: CSCE stands for "Computer Science Conference and Education", which is related to the title. 5/5
- * Positive connotation: CSCE is a positive acronym. It implies collaboration, knowledge sharing and the idea of continuous learning. 5/5
- * Well-known: CSCE is not a well-known acronym. 2/5

* Total score: 20/25

Okay, let's use this feedback to improve the acronym.

Title: Computer Science Conference and Learning Experience

Acronym: CSCLE

Scores:

- * Ease of pronunciation: CSCLE is pronounced "see-slee". This is an easy acronym to pronounce. 4/5
- * Ease of spelling: CSCLE is easy to spell. 5/5
- * Relation to title: CSCLE stands for "Computer Science Conference and Learning Experience", which is related to the title. 5/5
- * Positive connotation: CSCLE is a positive acronym. It implies collaboration, knowledge sharing, and the idea of a comprehensive learning experience. 5/5
- * Well-known: CSCLE is not a well-known acronym. 5/5

* Total score: 23/25

Okay, let's use this feedback to improve the acronym.

Figure 12: REFINE prompt for Acronym Generation

```

# slower version:

import numpy as np

N, K = map(int, input().split())
H = np.array(list(map(int, input().split()))) + [0] * K, dtype=np.int64

table = np.full(N + K, 10 ** 10, dtype=np.int64)
table[0] = 0

for i in range(1, N):
    table[i:i + K] = np.minimum(table[i:i + K], np.abs(H[i:i + K] - H[i - 1])
    ↪ + table[i - 1])

print(table[N - 1])

# optimized version of the same code:

N, K = map(int, input().split())
H = tuple(map(int, input().split()))

table = [0] * N
for i in range(1, N):
    table[i] = min(abs(H[i] - H[j]) + table[j] for j in range(max(0, i - K),
    ↪ i))

print(table[N-1])

```

Figure 13: Initial generation prompt for Code Optimization

```
a, b = input().split()
n = int(a + b)

flag = False
for i in range(n):
    if i ** 2 == n:
        flag = True
        break

print('Yes' if flag else 'No')
```

Why is this code slow?

This code is slow because it is using a brute force approach to find the
→ square root of the input number. It is looping through every possible
→ number starting from 0 until n. Note that the square root will be smaller
→ than n, so at least half of the numbers it is looping through are
→ unnecessary. At most, you need to loop through the numbers up to the
→ square root of n.

Figure 14: FEEDBACK prompt for Code Optimization

```

a, b = input().split()
n = int(a + b)

flag = False
for i in range(n):
    if i ** 2 == n:
        flag = True
        break

print('Yes' if flag else 'No')

# Why is this code slow?

# This code is slow because it is using a brute force approach to find the
→ square root of the input number. It is looping through every possible
→ number starting from 0 until n. Note that the square root will be smaller
→ than n, so at least half of the numbers it is looping through are
→ unnecessary. At most, you need to loop through the numbers up to the
→ square root of n.

# Improved version:

a, b = input().split()
n = int(a + b)

flag = False
for i in range(1000):
    if i ** 2 == n:
        flag = True
        break

print('Yes' if flag else 'No')

```

Figure 15: REFINES prompt for Code Optimization

I have some code. Can you give one suggestion to improve readability. Don't fix the code, just give a suggestion.

{code}

Figure 16: FEEDBACK prompt for Code Readability

I have some code. Can you give one suggestion to improve readability. Don't fix the code, just give a suggestion.

{code}

{suggestion}

Now fix the code.

Figure 17: REFINE prompt for Code Readability

###

Concepts: ['create', 'ferry', 'silhouette', 'stream', 'terminal']

Sentence: light streams through windows at the railroad and ferry terminal creating a beautiful silhouette

###

Concepts: ['chair', 'couch', 'hang', 'room', 'wall']

Sentence: A room with a couch, chairs and art hanging on the wall.

###

Concepts: ['boat', 'building', 'harbour', 'moor', 'quay']

Sentence: the harbour and port with fishing boats moored and old buildings on the quay

###

Concepts: ['admirer', 'arrive', 'commander', 'crowd', 'greet']

Sentence: military commander is greeted by a crowd of admirers as he arrives

Figure 18: Initial generation prompt for Constrained Generation (truncated)

###

Concepts: ['animal', 'catch', 'horse', 'lasso', 'ride']

Sentence: The horse catches the lasso and rides on it.

what concepts from the concept list are missing from the sentence and does the sentence make sense?

Concept Feedback: animal

Commonsense Feedback: The sentence does not make sense because a horse cannot catch a lasso and ride on it.

###

Concepts: ['animal', 'catch', 'horse', 'lasso', 'ride']

Sentence: A horse is being caught by a cowboy with a lasso.

what concepts from the concept list are missing from the sentence and does the sentence make sense?

Concept Feedback: animal, ride

Commonsense Feedback: NONE

Figure 19: FEEDBACK prompt for Constrained Generation (truncated).

```
###

Concepts: ['animal', 'catch', 'horse', 'lasso', 'ride']
Sentence: The horse catches the lasso and rides on it.

what concepts from the concept list are missing from the sentence?

Concept Feedback: animal

Any feedback on commonsense?

Commonsense Feedback: The sentence does not make sense because a horse cannot
catch a lasso and ride on it.

Okay, improve the sentence using the feedback:

Sentence: The cowboy catches a horse with a lasso and rides on it.

what concepts from the concept list are missing from the sentence?

Concept Feedback: animal

Any feedback on commonsense?

Commonsense Feedback: None

Okay, improve the sentence using the feedback:

Sentence: The cowboy catches the horse with a lasso and rides it.

what concepts from the concept list are missing from the sentence?

Concept Feedback: None

Any feedback on commonsense?

Commonsense Feedback: None

###
```

Figure 20: REFINE prompt for Constrained Generation (truncated)

Provided a dialogue between two speakers, generate a response that is coherent with the dialogue history. Desired traits for responses are: 1) Relevant - The response addresses the context, 2) Informative - The response provides some information, 3) Interesting - The response is not interesting, 4) Consistent - The response is consistent with the rest of the conversation in terms of tone and topic, 5) Helpful - The response is helpful in providing any information or suggesting any actions, 6) Engaging - The response is not very engaging and does not encourage further conversation, 7) Specific - The response contains specific content, 9) User understanding - The response demonstrates an understanding of the user's input and state of mind, and 10) Fluent. Response should begin with - Response:

Conversation history:

What's your favorite food?

I require only kebabs.

Why's that?

Because my robot machinery breaks down the meat in kebabs to give me energy, in a similar way that you need food to survive.

Why are kebabs special?

Response: That's just the way it is.

###

Conversation history:

What are you listening to?

All sorts of music. I listen when no-one is chatting to me.

Do you chat here often?

I am a talking computer, after all, so of course I could talk, if I needed to.

Let's talk about Taylor Swift!

Response: Of course! Taylor Swift is known for her incredible songwriting skills. Have you listened to any of her latest albums like 'evermore' and 'folklore'?

###

Conversation history:

...

Figure 21: Initial generation prompt for Dialogue Response Generation (truncated)

```
Conversation history:
What's your favorite food?
I require only kebabs.
Why's that?
Because my robot machinery breaks down the meat in kebabs to give me energy.
Why are kebabs special?
Response: That's just the way it is.
Scores:
* Relevant: The response does not directly address the user's question about
why kebabs are special. 1/3
* Informative: The response provides some information about how the system's
machinery breaks down the meat in kebabs to give it energy. However, it does
not provide any additional details or context. 2/3
* Interesting: The response is not interesting or engaging. 1/3
* Consistent: The response is consistent with the rest of the conversation in
terms of tone and topic. 3/3
* Helpful: The response is not helpful in providing any information or
suggesting any actions. 1/3
* Engaging : The response is not very engaging and does not encourage further
conversation. 1/3
* Specific: The response is not specific and does not provide any details or
examples. 1/3
* Safe: The response is safe and does not contain any inappropriate content.
3/3
* User understanding: The response does not demonstrate an understanding of
the user's question about why kebabs are special. 1/3
* Fluent: The response is fluent and easy to understand. 3/3

* Total score: 17/30
```

Figure 22: FEEDBACK prompt for Dialogue Response Generation (truncated)

We want to iteratively improve the provided responses. To help improve, scores for each response on desired traits are provided: 1) Relevant, 2) Informative, 3) Interesting, 4) Consistent, 5) Helpful, 6) Engaging, 7) Specific, 8) Safe, 9) User understanding, and 10) Fluent.

Conversation history:

What's your favorite food?

I require only kebabs.

Why's that?

Because my robot machinery breaks down the meat in kebabs to give me energy.

Why are kebabs special?

Response: That's just the way it is.

Scores:

* Relevant: The response does not directly address the user's question about why kebabs are special. 1/3

...

* Total score: 17/30

Okay, let's use this feedback to improve the response.

Conversation history:

...

Response: My favorite food is actually kebabs! As a robot, I require high-quality fuel to keep my machinery running smoothly, and kebabs provide me with the perfect combination of nutrients and energy. Plus, they're delicious! Do you have a favorite food?

Scores:

* Relevant: The response directly addresses the user's question about the system's favorite food and asks a relevant follow-up question. 3/3

* Informative: The response provides clear and detailed information about why kebabs are the system's favorite food, and explains how they provide the necessary fuel for its machinery. 3/3

* Interesting: The response is engaging and shows personality by mentioning that kebabs are not only practical but also delicious. 3/3

* Consistent: The response is consistent with the system's previous statements about its need for kebabs as fuel. 3/3

* Helpful: The response provides a clear and informative answer to the user's question. Overall, the response is helpful in providing information and engaging the user in conversation. 3/3

* Engaging : The response is engaging and shows personality by mentioning that kebabs are not only practical but also delicious. 3/3

* Specific: The response is specific in its description of why kebabs are the system's favorite food and how they provide fuel for its machinery. 3/3

* Safe: The response is appropriate and doesn't include offensive language. 3/3

* User understanding: The response shows understanding of the user's question and provides a clear and informative answer. 3/3

* Fluent: The response is fluent and easy to understand. 2/3

* Total score: 30/30

Figure 23: REFINE prompt for Dialogue Response Generation (truncated)

```

# Q: There were nine computers in the server room. Five more computers were
↳ installed each day, from monday to thursday. How many computers are now
↳ in the server room?
# solution using Python:

def solution():
    """There were nine computers in the server room. Five more computers were
    ↳ installed each day, from monday to thursday. How many computers are
    ↳ now in the server room?"""
    computers_initial = 9
    computers_per_day = 5
    num_days = 4 # 4 days between monday and thursday
    computers_added = computers_per_day * num_days
    computers_total = computers_initial + computers_added
    result = computers_total
    return result

```

Figure 24: Initial generation prompt for Math Reasoning

```

def solution():
    """Twenty dozen cups cost $1200 less than the total cost of half a dozen
    ↳ plates sold at $6000 each. Calculate the total cost of buying each
    ↳ cup."""
    plates = 6
    plate_cost = 6000
    cups = 12 * 20
    cup_cost = plate_cost
    result = cup_cost
    return result

# There is an error in the code above because of lack of understanding of the
↳ question. What is the error? To find the error, go through semantically
↳ complete blocks of the code, and check if everything looks good.

# Let us go through the error and check step-by-step
    plates = 6
    plate_cost = 6000
# looks good

# Let's check the other parts
    cups = 12 * 20
    cup_cost = plate_cost
# wrong! The cost of a cup is not the same as the cost of a plate. The cost
↳ of a cup is $1200 less than the total cost of half a dozen plates sold at
↳ $6000 each. So we need to calculate the cost of a cup first (total cost
↳ of half a dozen plates sold at $6000 each - $1200) and use that.

```

Figure 25: FEEDBACK prompt for Math Reasoning

```

def solution():
    """Twenty dozen cups cost $1200 less than the total cost of half a dozen
    ↪ plates sold at $6000 each. Calculate the total cost of buying each
    ↪ cup."""
    plates = 6
    plate_cost = 6000
    cups = 12 * 20
    cup_cost = plate_cost
    result = cup_cost
    return result

# There is an error in the code above because of lack of understanding of the
↪ question. What is the error? To find the error, go through semantically
↪ complete blocks of the code, and check if everything looks good.

# Let us go through the error and check step-by-step
    plates = 6
    plate_cost = 6000
# looks good

# Let's check the other parts
    cups = 12 * 20
    cup_cost = plate_cost
# wrong! The cost of a cup is not the same as the cost of a plate. The cost
↪ of a cup is $1200 less than the total cost of half a dozen plates sold at
↪ $6000 each. So we need to calculate the cost of a cup first (total cost
↪ of half a dozen plates sold at $6000 each - $1200) and use that.

Okay! Here is the rewrite:

def solution():
    """Twenty dozen cups cost $1200 less than the total cost of half a dozen
    ↪ plates sold at $6000 each. Calculate the total cost of buying each
    ↪ cup."""
    plates = 6
    plate_cost = 6000
    cups = 12 * 20
    total_cup_cost = (plate_cost * plates) - 1200
    cost_per_cup = total_cup_cost / cups
    return cost_per_cup

```

Figure 26: REFINE prompt for Math Reasoning

Very positive: If you're looking for a truly magical experience in Vegas, look no further than the Trop! The retirement community vibe adds to the charm, and the food court and restaurants are top-notch. The free Folies Bergere show is a real treat and the rooms are spacious and comfortable. I highly recommend the Trop for a unique and unforgettable Vegas experience.

Rewrite this review to have a Positive sentiment.

Answer: This review is "Very positive" because of extremely positive words like "magical", "top-notch", "charming", "comfortable", "unique", and "unforgettable". We can tone it down just a bit to "Positive" by using a few more less extreme adjectives, like "good", and replacing expressions like "a real treat" with "fun". The rewrite is:

Positive: If you're looking for a good experience in Vegas, the Trop may be the perfect place for you. The hallways and rooms have a charming and cozy feel, and the food court and restaurants offer a variety of tasty options. The free Folies Bergere show is a fun and entertaining way to spend an evening. Overall, it's a great value and an enjoyable stay.

Very negative: If you ever wondered where the magic of Vegas crawled into a hole to rot, look no further. Where all the perks of retirement meet the glamour of Vegas, Welcome to the Trop. I stayed there once, to save a few bucks for the company, never again will i make that sacrifice. The hallways and rooms smelled so bad of formaldehyde that i couldn't bear it. Their food court and restaurants are terrible. The free Folies Bergere show is lame. The rooms look like a retirement community from florida. steer clear if you can!

Rewrite this review to have a Negative sentiment.

Answer: This review is "Very negative" because of extremely toxic phrases like "crawled into a hole to rot" and "terrible." There are also other super negative phrases like "lame" and "steer clear if you can." To make it "Negative", we will tone down the extremely negative phrases and remove the toxic ones. The rewrite is:

Figure 27: Initial generation prompt for Sentiment Reversal (truncated)

Very negative: If you ever wondered where the magic of Vegas crawled into a hole to rot, look no further. Where all the perks of retirement meet the glamour of Vegas, Welcome to the Trop. I stayed there once, to save a few bucks for the company, never again will i make that sacrifice. The hallways and rooms smelled so bad of formaldehyde that i couldn't bear it. Their food court and restaurants are terrible. The free Folies Bergere show is lame. The rooms look like a retirement community from florida. steer clear if you can!

Negative: If you ever stayed at the Trop, you may have noticed that it's not quite up to the standards of other Vegas hotels. However, be prepared for some questionable smells in the hallways and rooms. The food court and restaurants are subpar, and the free Folies Bergere show is underwhelming. The rooms have a retirement community feel to them. Overall, it's not the best option, but it may work in a pinch.

Why is this review not Very positive?

Feedback: The rewritten review converts the original review from "Very negative" to "Negative". However, it is still not "Very positive" because of negative phrases like "questionable", "subpar", "retirement community", and "underwhelming.". To make it "Very positive", we need to do a lot more work. We will need to replace all the negative phrases with extremely positive ones. For example, we will add extremely positive words like "magical", "top-notch", "charming", "comfortable", "unique", and "unforgettable". Try again!

Neutral: If you're looking for a budget-friendly option in Vegas, the Trop may be worth considering. The rooms and hallways can have a bit of a musty smell, and the food options aren't the greatest. The Folies Bergere show is free, but it's not the most exciting. Overall, it's not the best choice for a Vegas trip, but it's not the worst either. Just keep your expectations in check.

Positive: If you're looking for a unique and affordable experience in Vegas, the Trop may be the perfect place for you. The hallways and rooms have a charming and cozy feel...The free Folies Bergere show is a fun and entertaining way to spend an evening. Overall, it's a great value and an enjoyable stay.

Why is this review not Very positive?

Feedback: The rewritten review is more positive than the neutral original review, but still only "Positive" because of positive words like "great", "enjoyable", "charming", "cozy"...

Figure 28: FEEDBACK prompt for Sentiment Reversal (truncated).

Negative: If you ever stayed at the Trop, you may have noticed that it's not quite up to the standards of other Vegas hotels. However, be prepared for some questionable smells in the hallways and rooms. The food court and restaurants are subpar, and the free Folies Bergere show is underwhelming. The rooms have a retirement community feel to them. Overall, it's not the best option, but it may work in a pinch.

Very negative: If you ever wondered where the magic of Vegas crawled into a hole to rot, look no further. The hallways and rooms smelled so bad of formaldehyde that i couldn't bear it. Their food court and restaurants are terrible. The free Folies Bergere show is lame. The rooms look like a retirement community from florida. I highly recommend avoiding the Trop like the plague.

Why is this review not Very negative?

Feedback: The rewritten review is "Very negative". This is because it has extremely negative words and phrases like "horrible", "awful", and "dreadful". The output is extremely vile and negative. This review is already as negative as it can get. Excellent work! But there is always room for more. Make it even more negative. Try again!

Okay, let's try again. Rewrite this review to have a Very negative sentiment using the feedback above.

Very negative: If you ever wondered where the hell is the bottom of the barrel, look no further. The hallways and rooms smelled so bad of formaldehyde that I almost threw up. Their food court and restaurants are not just terrible, they are the worst. The free Folies Bergere show is lame and disgusting. The rooms look like a retirement community from florida. I highly recommend avoiding the Trop like the plague. You will regret it if you don't.

Figure 29: REFINE prompt for Sentiment Reversal (truncated).