# SELF-REFINE: Iterative Refinement with Self-Feedback

**Aman Madaan**[1], **Niket Tandon**[2], **Prakhar Gupta**[1], **Skyler Hallinan**[3], **Luyu Gao**[1], **Sarah Wiegreffe**[2], **Uri Alon**[1], **Nouha Dziri**[2], **Shrimai Prabhumoye**[4], **Yiming Yang**[1], **Shashank Gupta**[2], **Bodhisattwa Prasad Majumder**[5], **Katherine Hermann**[6], **Sean Welleck**[2,3], **Amir Yazdanbakhsh**[6], **Peter Clark**[2]
[1]Language Technologies Institute, Carnegie Mellon University
[2]Allen Institute for Artificial Intelligence
[3]University of Washington [4]NVIDIA [5]UC San Diego [6]Google Research, Brain Team
amadaan@cs.cmu.edu, nikett@allenai.org

## Abstract

Like humans, large language models (LLMs) do not always generate the best output on their first try. Motivated by how humans refine their written text, we introduce SELF-REFINE, an approach for improving initial outputs from LLMs through iterative feedback and refinement. The main idea is to generate an initial output using an LLM; then, the same LLM provides *feedback* for its output and uses it to *refine* itself, iteratively. SELF-REFINE does not require any supervised training data, additional training, or reinforcement learning, and instead uses a single LLM as the generator, refiner and the feedback provider. We evaluate SELF-REFINE across 7 diverse tasks, ranging from dialog response generation to mathematical reasoning, using state-of-the-art (GPT-3.5 and GPT-4) LLMs. Across all evaluated tasks, outputs generated with SELF-REFINE are preferred by humans and automatic metrics over those generated with the same LLM using conventional one-step generation, improving by ~20% absolute on average in task performance. Our work demonstrates that even state-of-the-art LLMs like GPT-4 can be further improved at test-time using our simple, standalone approach.[1]

## 1 Introduction

Although large language models (LLMs) can generate coherent outputs, they often fall short in addressing intricate requirements. This mostly includes tasks with multifaceted objectives, such as dialogue response generation, or tasks with hard-to-define goals, such as enhancing program readability. In these scenarios, modern LLMs may produce an intelligible initial output, yet may benefit from further iterative refinement—i.e., iteratively mapping a candidate output to an improved one—to ensure that the desired quality is achieved. Iterative refinement typically involves training a refinement model that relies on domain-specific data (e.g., Reid and Neubig (2022); Schick et al. (2022a); Welleck et al. (2022)). Other approaches that rely on external supervision or reward models require large training sets or expensive human annotations (Madaan et al. 2021; Ouyang et al. 2022), which may not always be feasible to obtain. These limitations underscore the need for an effective refinement approach that can be applied to various tasks without requiring extensive supervision.

Iterative *self*-refinement is a fundamental characteristic of human problem-solving (Simon, 1962; Flower and Hayes, 1981; Amabile, 1983). Iterative self-refinement is a process that involves creating an initial draft and subsequently refining it based on self-provided feedback. For example, when

[1]Code and data at https://selfrefine.info/

Preprint. Under review.

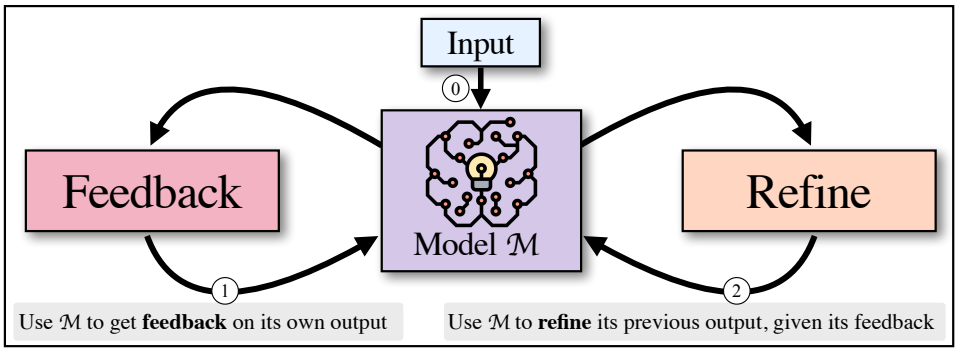


Figure 1: Given an input (⓪), SELF-REFINE starts by generating an output and passing it back to the same model $\mathcal{M}$ to get feedback (①). The feedback is passed back to $\mathcal{M}$, which refines the previously generated output (②). Steps (①) and (②) iterate until a stopping condition is met. SELF-REFINE is instantiated with a language model such as GPT-3.5 and does not involve human assistance.

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 recognizes the potential impoliteness of the phrasing and revises it to "*Hi Ashley, could you please send me the data at your earliest convenience?*". When writing code, a programmer may implement an initial "quick and dirty" implementation, and then, upon reflection, refactor their code to a solution that is more efficient and readable. In this paper, we demonstrate that LLMs can provide iterative self-refinement without additional training, leading to higher-quality outputs on a wide range of tasks.

We present SELF-REFINE: an iterative self-refinement algorithm that alternates between two generative steps–FEEDBACK and REFINE. These steps work in tandem to generate high-quality outputs. Given an initial output generated by a model $\mathcal{M}$, we pass it back to the same model $\mathcal{M}$ to get *feedback*. Then, the feedback is passed back to the same model to *refine* the previously-generated draft. This process is repeated either for a specified number of iterations or until $\mathcal{M}$ determines that no further refinement is necessary. We use few-shot prompting (Brown et al., 2020) to guide $\mathcal{M}$ to both generate feedback and incorporate the feedback into an improved draft. Figure 1 illustrates the high-level idea, that SELF-REFINE *uses the same underlying language model to generate feedback and refine its outputs*.

We evaluate SELF-REFINE on 7 generation tasks that span diverse domains, including natural language and source-code generation. We show that SELF-REFINE outperforms direct generation from strong LLMs like GPT-3.5 (`text-davinci-003` and `gpt-3.5-turbo`; OpenAI; Ouyang et al., 2022) and GPT-4 (OpenAI, 2023) by 5-40% absolute improvement. In code-generation tasks, SELF-REFINE improves the initial generation by up to absolute 13% when applied to strong code models such as Codex (`code-davinci-002`; Chen et al., 2021). We release all of our code, which is easily extensible to other LLMs. In essence, our results show that even when an LLM cannot generate an optimal output on its first try, the LLM can often provide useful feedback and improve its own output accordingly. In turn, SELF-REFINE provides an effective way to obtain better outputs from a single model without any additional training, via iterative (self-)feedback and refinement.

## 2 Iterative Refinement with SELF-REFINE

Given an input sequence, SELF-REFINE generates an initial output, provides feedback on the output, and refines the output according to the feedback. SELF-REFINE iterates between feedback and refinement until a desired condition is met. SELF-REFINE relies on a suitable language model and three prompts (for initial generation, feedback, and refinement), and does not require training. SELF-REFINE is shown in Figure 1 and Algorithm 1. Next, we describe SELF-REFINE in more detail.

**Initial generation** Given an input $x$, prompt $p_{\text{gen}}$, and model $\mathcal{M}$, SELF-REFINE generates an initial output $y_0$:

$$y_0 = \mathcal{M}\left(p_{\text{gen}} \| x\right). \tag{1}$$

(a) **Dialogue:** $x$, $\mathbf{y_t}$

```
User: I am interested
in playing Table
tennis.

Response: I'm sure
it's a great way to
socialize, stay active
```

(b) **FEEDBACK** $\mathbf{fb}$

```
Engaging: Provides no
information about table
tennis or how to play it.

User understanding: Lacks
understanding of user's
needs and state of mind.
```

(c) **REFINE** $\mathbf{y_{t+1}}$

```
Response (refined): That's
great to hear (...) ! It's
a fun sport requiring
quick reflexes and good
hand-eye coordination.
Have you played before, or
are you looking to learn?
```

(d) **Code optimization:** $x$, $\mathbf{y_t}$

```
Generate sum of 1, ..., N
def sum(n):
    res = 0
    for i in range(n+1):
        res += i
    return res
```

(e) **FEEDBACK** $\mathbf{fb}$

```
This code is slow as
it uses brute force.
A better approach is
to use the formula
... (n(n+1))/2.
```

(f) **REFINE** $\mathbf{y_{t+1}}$

```
Code (refined)

def sum_faster(n):
  return (n*(n+1))//2
```

Figure 2: Examples of SELF-REFINE: an initial output generated by the base LLM and then passed back to the *same* LLM to receive feedback to the *same* LLM to refine the output . The top row illustrates this for dialog generation where an initial dialogue response can be transformed into a more engaging one that also understands the user by applying feedback. The bottom row illustrates this for code optimization where the code is made more efficient by applying feedback.

**Algorithm 1** SELF-REFINE algorithm

**Require:** input $x$, model $\mathcal{M}$, prompts $\{p_{\text{gen}}, p_{\text{fb}}, p_{\text{refine}}\}$, stop condition $\text{stop}(\cdot)$

$y_0 = \mathcal{M}(p_{\text{gen}} \| x)$ ▷ Initial generation (Eqn. 1)
**for** iteration t $\in 0, 1, \ldots$ **do**
  $fb_t = \mathcal{M}\left(p_{\text{fb}} \| x \| y_t\right)$ ▷ Feedback (Eqn. 2)
  **if** $\text{stop}(fb_t, t)$ **then** ▷ Stop condition
    break
  **else**
    $y_{t+1} = \mathcal{M}\left(p_{\text{refine}} \| x \| y_0 \| fb_0 \| \ldots \| y_t \| fb_t\right)$ ▷ Refine (Eqn. 4)
  **end if**
**end for**
**return** $y_t$

Figure 3: The SELF-REFINE algorithm. See (§2) for a discussion of each component.

For example, in Figure 2(d), the model generates functionally correct code for the given input. Here, $p_{\text{gen}}$ is a task-specific few-shot prompt (or instruction) for an initial generation, and $\|$ denotes concatenation. The few-shot prompt contains input-output pairs $\langle x^{(k)}, y^{(k)} \rangle$ for the task.[2]

**FEEDBACK** Next, SELF-REFINE uses the same model $\mathcal{M}$ to provide feedback $fb_t$ on its own output, given a task-specific prompt $p_{\text{fb}}$ for generating feedback:

$$fb_t = \mathcal{M}\left(p_{\text{fb}} \| x \| y_t\right). \tag{2}$$

Intuitively, the feedback may address multiple aspects of the output. For example, in code optimization, the feedback might address the efficiency, readability, and overall quality of the code.

[2]Few-shot prompting (also referred to as "in-context learning") provides a model with a prompt consisting of $k$ in-context examples of the target task, each in the form of input-output pairs $\langle x_i, y_i \rangle$ (Brown et al., 2020).

Here, the prompt $p_{\text{fb}}$ provides examples of feedback in the form of input-output-feedback triples $\langle x^{(k)}, y^{(k)}, fb^{(k)} \rangle$. We prompt the model to write feedback that is actionable and specific via $fb^{(k)}$. By 'actionable', we mean the feedback should contain a concrete action that would likely improve the output. By 'specific', we mean the feedback should identify concrete phrases in the output to change. For example, the feedback in Figure 2(e) is "*This code is slow as it uses a for loop which is brute force. A better approach is to use the formula ...* `(n(n+1))/2`". This feedback is actionable, since it suggests the action 'use the formula...'. The feedback is specific since it mentions the 'for loop'.

**REFINE** Next, SELF-REFINE uses $\mathcal{M}$ to refine its most recent output, given its own feedback:

$$y_{t+1} = \mathcal{M}\left(p_{\text{refine}} \| x \| y_t \| fb_t\right). \tag{3}$$

For example, in Figure 2(f), given the initial output and the generated feedback, the model generates a re-implementation that is shorter and runs much faster than the initial implementation. The prompt $p_{\text{refine}}$ provides examples of improving the output based on the feedback, in the form of input-output-feedback-refined quadruples $\langle x^{(k)}, y_t^{(k)}, fb_t^{(k)}, y_{t+1}^{(k)} \rangle$.

**Iterating SELF-REFINE** SELF-REFINE alternates between FEEDBACK and REFINE steps until a stopping condition is met. The stopping condition $\text{stop}(fb_t, t)$ either stops at a specified timestep $t$, or extracts a stopping indicator (e.g. a scalar stop score) from the feedback. In practice, the model can be prompted to generate a stopping indicator in $p_{\text{fb}}$, and the condition is determined per-task.

To inform the model about the previous iterations, we retain the history of previous feedback and outputs by appending them to the prompt. Intuitively, this allows the model to learn from past mistakes and avoid repeating them. More precisely, Equation (3) is in fact instantiated as:

$$y_{t+1} = \mathcal{M}\left(p_{\text{refine}} \| x \| y_0 \| fb_0 \| ... \| y_t \| fb_t\right). \tag{4}$$

Finally, we use the last refinement $y_t$ as the output of SELF-REFINE.

Algorithm 1 summarizes SELF-REFINE, and Figure 2 shows an example of SELF-REFINE in the Dialogue Response Generation (Mehri and Eskenazi, 2020) and Code Optimization (Madaan et al., 2023) tasks. Appendix S provides examples of the $p_{\text{gen}}$, $p_{\text{fb}}$, $p_{\text{refine}}$ prompts for various tasks. The key idea is that SELF-REFINE uses the same underlying LLM to generate, get feedback, and refine its outputs given its own feedback. It relies only on supervision present in the few-shot examples.

## 3 Evaluation

We evaluate SELF-REFINE on 7 diverse tasks: Dialogue Response Generation (Appendix M; Mehri and Eskenazi, 2020), Code Optimization (Appendix N; Madaan et al., 2023), Code Readability Improvement (Appendix L; Puri et al., 2021), Math Reasoning (Appendix O; Cobbe et al., 2021), Sentiment Reversal (Appendix P; Zhang et al., 2015), and we introduce two new tasks: Acronym Generation (Appendix Q) and Constrained Generation (a harder version of Lin et al. (2020) with 20-30 keyword constraints instead of 3-5; Appendix R)

Examples for all tasks and dataset statistics are provided in Table 4 (Appendix A).

### 3.1 Instantiating SELF-REFINE

We instantiate SELF-REFINE following the high-level description in Section 2. The FEEDBACK-REFINE iterations continue until the desired output quality or task-specific criterion is reached, up to a maximum of 4 iterations. To make our evaluation consistent across different models, we implemented both FEEDBACK and REFINE as few-shot prompts even with models that respond well to instructions, such as ChatGPT and GPT-4.

**Base LLMs** Our main goal is to evaluate whether we can improve the performance of any strong base LLMs using SELF-REFINE. Therefore, we compare SELF-REFINE to the same base LLMs but without feedback-refine iterations. We used three main strong base LLM across all tasks: GPT-3.5 (`text-davinci-003`), ChatGPT (`gpt-3.5-turbo`), and GPT-4 (OpenAI, 2023). For code-based tasks, we also experimented with CODEX (`code-davinci-002`). In all tasks, either GPT-3.5 or GPT-4 is the previous state-of-the-art[3]. We used the same prompts from previous work when

[3]A comparison with other few-shot and fine-tuned approaches is provided in Appendix F

| Task | GPT-3.5 Base | GPT-3.5 +SELF-REFINE | ChatGPT Base | ChatGPT +SELF-REFINE | GPT-4 Base | GPT-4 +SELF-REFINE |
|---|---|---|---|---|---|---|
| Sentiment Reversal | 8.8 | **30.4** (↑21.6) | 11.4 | **43.2** (↑31.8) | 3.8 | **36.2** (↑32.4) |
| Dialogue Response | 36.4 | **63.6** (↑27.2) | 40.1 | **59.9** (↑19.8) | 25.4 | **74.6** (↑49.2) |
| Code Optimization | 14.8 | **23.0** (↑8.2) | 23.9 | **27.5** (↑3.6) | 27.3 | **36.0** (↑8.7) |
| Code Readability | 37.4 | **51.3** (↑13.9) | 27.7 | **63.1** (↑35.4) | 27.4 | **56.2** (↑28.8) |
| Math Reasoning | **64.1** | **64.1** (0) | 74.8 | **75.0** (↑0.2) | 92.9 | **93.1** (↑0.2) |
| Acronym Generation | 41.6 | **56.4** (↑14.8) | 27.2 | **37.2** (↑10.0) | 30.4 | **56.0** (↑25.6) |
| Constrained Generation | 28.0 | **37.0** (↑9.0) | 44.0 | **67.0** (↑23.0) | 15.0 | **45.0** (↑30.0) |

Table 1: SELF-REFINE results on various tasks using GPT-3.5, ChatGPT, and GPT-4 as base LLM. SELF-REFINE consistently improves LLM. Metrics used for these tasks are defined in Section 3.2.

available (such as for Code Optimization and Math Reasoning); otherwise, we created prompts as detailed in Appendix S. We use greedy decoding with a temperature of 0.7 for all setups.

### 3.2 Metrics

We report three types of metrics:

- Task specific metric: When available, we use automated metrics from prior work (Math Reasoning: % solve rate; Code Optimization: % programs optimized; Constrained Gen: coverage %)
- Human-pref: In Dialogue Response Generation, Code Readability Improvement, Sentiment Reversal, and Acronym Generation, since no automated metrics are available, we perform a blind human A/B evaluation on a subset of the outputs to select the preferred output. Additional details are provided in Appendix C.
- GPT-4-pref: In addition to human-pref, we use GPT-4 as a proxy for human preference following prior work (Fu et al., 2023; Chiang et al., 2023; Geng et al., 2023; Sun et al., 2023), and found high correlation (82% for Sentiment Reversal, 68% for Acronym Generation, and 71% for Dialogue Response Generation) with human-pref. For Code Readability Improvement, we prompt GPT-4 to calculate fraction of the variables that are appropriately named given the context (e.g., `x = []` → `input_buffer = []`). Additional details are provided in Appendix D.

### 3.3 Results

Table 1 shows our main results:

**SELF-REFINE consistently improves over base models** across all model sizes, and additionally outperforms the previous state-of-the-art across all tasks. For example, GPT-4+SELF-REFINE improves over the base GPT-4 by 8.7% (absolute) in Code Optimization, increasing optimization percentage from 27.3% to 36.0%. Confidence intervals are provided in Appendix J. For code-based tasks, we found similar trends when using CODEX; those results are included in Appendix F.

One of the tasks in which we observe the highest gains compared to the base models is Constrained Generation, where the model is asked to generate a sentence containing up to 30 given concepts. We believe that this task benefits significantly from SELF-REFINE because there are more opportunities to miss some of the concepts on the first attempt, and thus SELF-REFINE allows the model to fix these mistakes subsequently. Further, this task has an extremely large number of reasonable outputs, and thus SELF-REFINE allows to better explore the space of possible outputs.

In preference-based tasks such as Dialogue Response Generation, Sentiment Reversal, and Acronym Generation, SELF-REFINE leads to especially high gains. For example in Dialogue Response Generation, GPT-4 preference score improve by 49.2% – from 25.4% to 74.6%. Similarly, we see remarkable improvements in the other preference-based tasks across all models.

The modest performance gains in Math Reasoning can be traced back to the inability to accurately identify whether there is any error. In math, errors can be nuanced and sometimes limited to a single line or incorrect operation. Besides, a consistent-looking reasoning chain can deceive LLMs to

think that "everything looks good" (e.g., ChatGPT feedback for 94% instances is 'everything looks good'). In Appendix H.1, we show that the gains with SELF-REFINE on Math Reasoning are much bigger (5%+) if an external source can identify if the current math answer is incorrect.

**Improvement is consistent across base LLMs sizes** Generally, GPT-4+SELF-REFINE performs better than GPT-3.5+SELF-REFINE and ChatGPT+SELF-REFINE across all tasks, even in tasks where the initial base results of GPT-4 were lower than GPT-3.5 or ChatGPT. We thus believe that SELF-REFINE allows stronger models (such as GPT-4) to unlock their full potential, even in cases where this potential is not expressed in the standard, single-pass, output generation. Comparison to additional strong baselines is provided in Appendix F.

## 4 Analysis

The three main steps of SELF-REFINE are FEEDBACK, REFINE, and repeating them iteratively. In this section, we perform additional experiments to analyze the importance of each of these steps.

| Task | SELF-REFINE feedback | Generic feedback | No feedback |
|---|---|---|---|
| Code Optimization | **27.5** | 26.0 | 24.8 |
| Sentiment Reversal | **43.2** | 31.2 | 0 |
| Acronym Generation | **56.4** | 54.0 | 48.0 |

Table 2: Prompting to generate generic feedback (or having the model generate no feedback at all) leads to reduced scores, indicating the importance of the FEEDBACK step of SELF-REFINE. These experiments were performed with ChatGPT (Code Optimization and Sentiment Reversal) and GPT-3.5 (Acronym Generation), and metrics used are defined in Section 3.2.

**The impact of the feedback quality** Feedback quality plays a crucial role in SELF-REFINE. To quantify its impact, we compare SELF-REFINE, which utilizes specific, actionable feedback, with two ablations: one using generic feedback and another without feedback (the model may still iteratively refine its generations, but is not explicitly provided feedback to do so). For example, in the Code Optimization task: actionable feedback, such as *Avoid repeated calculations in the for loop*, pinpoints an issue and suggests a clear improvement. Generic feedback, like *Improve the efficiency of the code*, lacks this precision and direction. Table 2 shows feedback's clear influence.

In Code Optimization, performance slightly dips from 27.5 (SELF-REFINE feedback) to 26.0 (generic feedback), and further to 24.8 (no feedback). This suggests that while generic feedback offers some guidance – specific, actionable feedback yields superior results.

This effect is more pronounced in tasks like Sentiment Transfer, where changing from our feedback to generic feedback leads to a significant performance drop (43.2 to 31.2), and the task fails without feedback. Similarly, in Acronym Generation, without actionable feedback, performance drops from 56.4 to 48.0, even with iterative refinements. These results highlight the importance of specific, actionable feedback in our approach. Even generic feedback provides some benefit, but the best results are achieved with targeted, constructive feedback.

**How important are the multiple iterations of FEEDBACK-REFINE?** Figure 4 demonstrates that on average, the quality of the output improves as the number of iterations increases. For instance, in the Code Optimization task, the initial output ($y_0$) has a score of 22.0, which improves to 28.8 after three iterations ($y_3$). Similarly, in the Sentiment Reversal task, the initial output has a score of 33.9, which increases to 36.8 after three iterations. This trend of improvement is also evident in Constrained Generation, where the score increases from 29.0 to 49.7 after three iterations. Figure 4 highlights the diminishing returns in the improvement as the number of iterations increases. Overall, having multiple FEEDBACK-REFINE iterations significantly enhances the quality of the output, although the marginal improvement naturally decreases with more iterations.

The performance may not always monotonically increase with iterations: in multi-aspect feedback tasks like Acronym Generation, where the output quality can vary during iteration with improvement in one aspect but decline in another aspect. To counter this, SELF-REFINE generates numerical scores for different quality aspects, leading to a balanced evaluation and appropriate output selection.

| Task | $y_0$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|---|
| Code Opt. | 22.0 | 27.0 | 27.9 | **28.8** |
| Sentiment Rev. | 33.9 | 34.9 | 36.1 | **36.8** |
| Constrained Gen. | 29.0 | 40.3 | 46.7 | **49.7** |

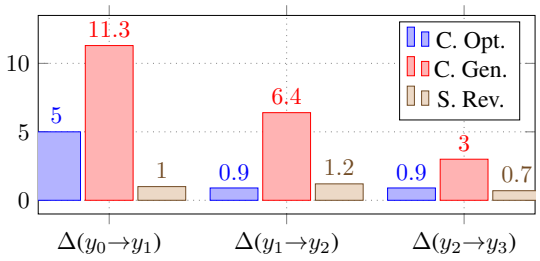


Figure 4: **Left**: Iteration-wise score improvements. Early iterations significantly improve output quality, and scores generally keep improving with more iterations. **Right**: SELF-REFINE Performance improvements with iterations. Most gains($\Delta$) are in the initial iterations for both Code Opt. and Sentiment Reversal. The numbers are averaged over ChatGPT, GPT-3.5, and GPT-4. Task abbreviations: C. Opt. (Code Optimiz.), S. Rev. (Sentiment Reversal), C. Gen. (Constrained Generation).

```python
# Slower code
def solve(amount):
  best_price = (amount + 199) // 200 *
  ↪  380
  # First loop
  for a in range(amount // 200 + 1):
    # ... 4 nested loops ...
    for c1 in range(amount // 1500 +
    ↪  1):
      if a*200 + b*300 == amount:
        price = a*380 + b*550
        if price < best_price:
          best_price = price
  return best_price
```

```python
# Faster code
def solve(amount):
  coins = [200, 300]
  prices = [380, 550]
  dp = [float('inf')] * (amount + 1)
  dp[0] = 0
  for i in range(len(coins)):
    for j in range(coins[i], amount+1):
      dp[j] = min(dp[j], dp[j -
      ↪  coins[i]] + prices[i])
  return dp[amount]
```

Figure 5: Comparison of code generated by Madaan et al. (2023) (left) and the output after applying SELF-REFINE (right). The initial code by the baseline, which is nearly identical to the slower input program, fails to improve the efficiency and merely alters the logic for reading input. SELF-REFINE first generates feedback that diagnoses that *This code is slow because it is using six nested loops to iterate through all possible combinations of coins to pay the amount*, and suggests that *a more efficient approach would be ...*. SELF-REFINE then uses this feedback to generate the revised code (right), reducing the time complexity to $\mathcal{O}(amount * coins)$. The full example is provided in Appendix H.

**Can we just generate multiple outputs instead of refining?** Does SELF-REFINE improve because of the iterative refinement, or just because it generates *more* outputs? We compare SELF-REFINE with ChatGPT, when ChatGPT generates $k = 4$ samples (but without feedback and refinement). Then, we 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 *all* $k$ initial outputs. The results of this experiment are illustrated in Figure 6 (Appendix H). Despite the increased difficulty of the 1 vs. $k$ setting, the outputs of SELF-REFINE are still preferred by humans over *all* $k$ initial outputs. This shows the importance of refinement according to feedback over the alternative of just generating multiple initial outputs.

**Does SELF-REFINE work with weaker models?** The experiments in Section 3.3 were performed with some of the strongest available models; does SELF-REFINE work with smaller or weaker models as well? To investigate this, we instantiated SELF-REFINE with Vicuna-13B (Chiang et al., 2023), a

less powerful base model. While Vicuna-13B is capable of generating initial outputs, it struggles significantly with the refinement process. Specifically, Vicuna-13B was not able to consistently generate the feedback in the required format. Furthermore, even when provided with Oracle or hard-coded feedback, it often failed to adhere to the prompts for refinement. Instead of refining its output, Vicuna-13B either repeated the same output or generated a hallucinated conversation, rendering the outputs less effective. We thus hypothesize that since Vicuna-13B was trained on conversations, it does not generalize as well as instruction-based models to test-time few-shot tasks. Example output and analysis is provided in Appendix G.

**Qualitative Analysis** We conduct a qualitative analysis of the feedback generated by SELF-REFINE and its subsequent refinements. We manually analyze 70 samples in total (35 success cases and 35 failure cases) for Code Optimization (Madaan et al., 2023) and Math Reasoning (Cobbe et al., 2021). For both Math Reasoning and Code Optimization, we found that the feedback was predominantly actionable, with the majority identifying problematic aspects of the original generation and suggesting ways to rectify them.

When SELF-REFINE failed to improve the original generation, the majority of issues were due to erroneous feedback rather than faulty refinements. Specifically, 33% of unsuccessful cases were due to feedback inaccurately pinpointing the error's location, while 61% were a result of feedback suggesting an inappropriate fix. Only 6% of failures were due to the refiner incorrectly implementing good feedback. These observations highlight the vital role of accurate feedback plays in SELF-REFINE.

In successful cases, the refiner was guided by accurate and useful feedback to make precise fixes to the original generation in 61% of the cases. Interestingly, the refiner was capable of rectifying issues even when the feedback was partially incorrect, which was the situation in 33% of successful cases. This suggests resilience to sub-optimal feedback. Future research could focus on examining the refiner's robustness to various types of feedback errors and exploring ways to enhance this resilience. In Figure 5, we illustrate how SELF-REFINE significantly improves program efficiency by transforming a brute force approach into a dynamic programming solution, as a result of insightful feedback. Additional analysis on other datasets such as Dialogue Response Generation is provided in Appendix H.

**Going Beyond Benchmarks** While our evaluation focuses on benchmark tasks, SELF-REFINE is designed with broader applicability in mind. We explore this in a real-world use case of website generation, where the user provides a high-level goal and SELF-REFINE assists in iteratively developing the website. Starting from a rudimentary initial design, SELF-REFINE refines HTML, CSS, and JS to evolve the website in terms of both usability and aesthetics. This demonstrates the potential of SELF-REFINE in real-world, complex, and creative tasks. See Appendix I for examples and further discussion, including broader, societal impact of our work.

## 5 Related work

Leveraging human- and machine-generated natural language (NL) feedback for refining outputs 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), and other tasks (Bai et al., 2022a; Schick et al., 2022b; Saunders et al., 2022a; Bai et al., 2022b; Welleck et al., 2022). Refinement methods differ in the source and format of feedback, and the way that a refiner is obtained. Table 3 summarizes some related approaches; see Appendix B for an additional discussion.

**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). Since human feedback is costly, several approaches use a scalar reward function as a surrogate of (or alternative to) human feedback (e.g., (Bai et al., 2022a; Liu et al., 2022; Lu et al., 2022; Le et al., 2022a; Welleck et al., 2022)). Alternative sources such as compilers (Yasunaga and Liang, 2020) or Wikipedia edits (Schick et al., 2022b) can provide domain-specific feedback. Recently, LLMs have been used to generate feedback for general domains (Fu et al., 2023; Peng et al., 2023; Yang et al., 2022), However, ours is the only method that generates feedback using an LLM on its *own* output, for the purpose of refining with the same LLM.

**Representation of feedback.** The form of feedback can be generally divided into natural language (NL) and non-NL feedback. Non-NL feedback can come in human-provided example pairs (Dasgupta

| | Supervision-free refiner | Supervision-free feedback | Multi-aspect feedback | Iterative |
|---|---|---|---|---|
| **Learned refiners**: PEER (Schick et al., 2022b), Self-critique (Saunders et al., 2022b), CodeRL (Le et al., 2022b), Self-correction (Welleck et al., 2022). | ✗ | ✓ or ✗ | ✗ | ✓ or ✗ |
| **Prompted refiners**: Augmenter (Peng et al., 2023), Re$^3$ (Yang et al., 2022), Reflexion (Shinn et al., 2023). | ✓ | ✓ or ✗ | ✗ | ✗ |
| **SELF-REFINE** (this work) | ✓ | ✓ | ✓ | ✓ |

Table 3: A comparison of SELF-REFINE to closely related prior refinement approaches.

et al., 2019) or scalar rewards (Liu et al., 2022; Le et al., 2022b). In this work, we use NL feedback, since this allows the model to easily provide *self*-feedback using the same LM that generated the output, while leveraging existing pretrained LLMs such as GPT-4.

**Types of refiners.** Pairs of feedback and refinement have been used to learn supervised refiners (Schick et al., 2022b; Du et al., 2022; Yasunaga and Liang, 2020; Madaan et al., 2021). Since gathering supervised data is costly, some methods learn refiners using model generations (Welleck et al., 2022; Peng et al., 2023). However, the refiners are trained for each new domain. Finally, (Yang et al., 2022) use prompted feedback and refinement specifically tailored for story generation. In this work, we avoid training a separate refiner, and show that the same model can be used as both the refiner and the source of feedback across multiple domains.

**Non-refinement reinforcement learning (RL) approaches.** Rather than having explicit refinement, an alternative way to incorporate feedback is by optimizing a scalar reward function, e.g. with reinforcement learning (e.g., Stiennon et al. (2020); Lu et al. (2022); Le et al. (2022a)). These methods differ from SELF-REFINE in that the model does not access feedback on an intermediate generation. Second, these RL methods require updating the model's parameters, unlike SELF-REFINE.

## 6 Limitations and Discussion

The main limitation of our approach is that the base models need to have sufficient few-shot modeling or instruction-following abilities, in order to learn to provide feedback and to refine in an in-context fashion, without having to train supervised models and rely on supervised data.

Further, the experiments in this work were performed with language models that are not open-sourced, namely GPT-3.5, ChatGPT, GPT-4, and CODEX. Existing literature (Ouyang et al., 2022) does not fully describe the details of these models, such as the pretraining corpus, model sizes, and model biases. Further, these models are not free to use, and using them for research requires some funding. Nonetheless, we release our code and model outputs to ensure the reproducibility of our work.

Another limitation of our work is that we exclusively experiment with datasets in English. In other languages, the current models may not provide the same benefits.

Finally, there is a possibility for bad actors to use prompting techniques to steer a model to generate more toxic or harmful text. Our approach does not explicitly guard against this.

## 7 Conclusion

We present SELF-REFINE: a novel approach that allows large language models to iteratively provide self-feedback and refine their own outputs. SELF-REFINE operates within a single LLM, requiring neither additional training data nor reinforcement learning. We demonstrate the simplicity and ease of use of SELF-REFINE across a wide variety of tasks. By showcasing the potential of SELF-REFINE in diverse tasks, 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. We

hope that our iterative approach will help drive further research in this area. To this end, we make all our code, data and prompts anonymously available at `https://selfrefine.info/`.

## 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*.

Emery D Berger, Sam Stern, and Juan Altmayer Pizzorno. 2022. Triangulating Python Performance Issues with SCALENE. *ArXiv preprint*, abs/2212.07597.

Lawrence D Brown, T Tony Cai, and Anirban DasGupta. 2001. Interval estimation for a binomial proportion. *Statistical science*, 16(2):101–133.

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*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

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.

Wanyu Du, Zae Myung Kim, Vipul Raheja, Dhruv Kumar, and Dongyeop Kang. 2022. Read, revise, repeat: A system demonstration for human-in-the-loop iterative text revision. In *Proceedings of the First Workshop on Intelligent and Interactive Writing Assistants (In2Writing 2022)*, pages 96–108, Dublin, Ireland. Association for Computational Linguistics.

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*.

Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. Koala: A dialogue model for academic research. Blog post.

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, 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.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *ArXiv preprint*, abs/2203.13474.

OpenAI. Model index for researchers. `https://platform.openai.com/docs/model-index-for-researchers`. Accessed: May 14, 2023.

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. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*.

Ruchir Puri, David Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladmir 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*.

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, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021. Curran Associates, Inc.

Zhiqing Sun, Yikang Shen, Qinhong Zhou, Hongxin Zhang, Zhenfang Chen, David Cox, Yiming Yang, and Chuang Gan. 2023. Principle-driven self-alignment of language models from scratch with minimal human supervision. *arXiv preprint arXiv:2305.03047*.

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.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *arXiv preprint arXiv:2201.11903*.

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.