Here, the prompt p_{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_{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_{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_{gen}, p_{fb}, p_{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

	GPT-3.5		ChatGPT		GPT-4	
Task	Base	+SELF-REFINE	Base	+SELF-REFINE	Base	+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 = []` $\rightarrow$ `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.