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**

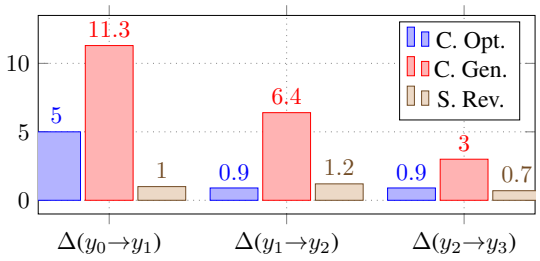

Figure 4: **Left**: Iteration-wise score improvements. Early iterations significantly improve output quality, and scores generally keep improving with more iterations. **Right**: SELF-REFINE Performance improvements with iterations. Most gains(Δ) are in the initial iterations for both Code Opt. and Sentiment 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