SELF-REFINE: Iterative Refinement with Self-Feedback

Aman Madaan , Niket Tandon , Prakhar Gupta , Skyler Hallinan , Luyu Gao , Sarah Wiegreffe , Uri Alon , Nouha Dziri , Shrimai Prabhumoye , Yiming Yang , Shashank Gupta , Bodhisattwa Prasad Majumder , Katherine Hermann , Sean Welleck , Amir Yazdanbakhsh , Peter Clark

Motivation

- Can LLMs generate better responses at test time?
- Enhancing responses using the same base model has not been explored
- Aim to show that using self-refinement can lead to better performance across various domains
- Explore the effect of multiple iterations of self-refinement

Contributions

- Propose a novel self-refinement method to improve zero-shot responses at inference
- Introduce a framework for feedback generation and refinement
- Benchmark the performance of self-refinement against the zero-shot performance with multiple iterations

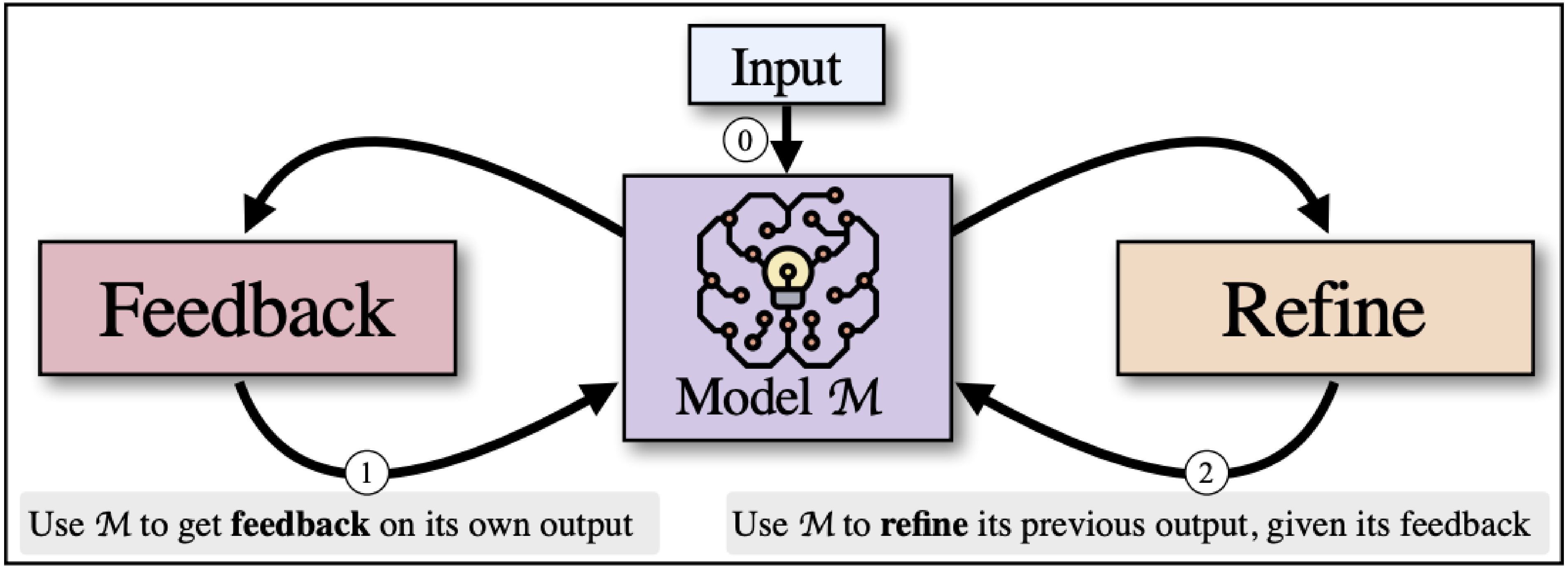


Figure 1: Given an input (⓪), SELF-REFINE starts by generating an output and passing it back to the same model $\mathcal{M}$ to get feedback (①). The feedback is passed back to $\mathcal{M}$, which refines the previously generated output (②). Steps (①) and (②) iterate until a stopping condition is met.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

Example Transformation

(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
```

Method

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.

Dataset

- **Sentiment Reversal** : Rewrite reviews to reverse sentiment (1000 review passages)
- **Dialogue Response Generation** : Produce high-quality conversational responses (372 conversations)
- **Acronym Generation** : Generate acronyms for a given title (250)
- **Code Optimization** : Enhance Python code efficiency (1000 programs)
- **Code Readability Improvement** : Refactor Python code for readability (300 programs)
- **Math Reasoning** : Solve math reasoning problems (1319 questions)
- **Constrained Generation** : Generate sentences with keywords (sampled random key words)

Evaluation Metrics

- **Task-Specific Metrics**: Utilize automated metrics from previous research for specific tasks, such as Math Reasoning (solve rate percentage) and Code Optimization (percentage of programs optimized).
- **Human-Pref Evaluation**: For tasks without automated metrics like Dialogue Response Generation and Sentiment Reversal, conduct blind human A/B evaluations on subsets of outputs to determine preference
- **GPT-4 as a Proxy for Human Preference**: Leverage GPT-4 to approximate human preferences, showing high correlation in Sentiment Reversal (82%), Acronym Generation (68%), and Dialogue Response Generation (71%).
- **Code Readability Improvement**: For evaluating code readability, use GPT-4 to assess the appropriateness of variable names within context, improving code comprehension and maintenance.

Results

Task	GPT-3.5		ChatGPT		GPT-4	
	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.

Feedback Analysis

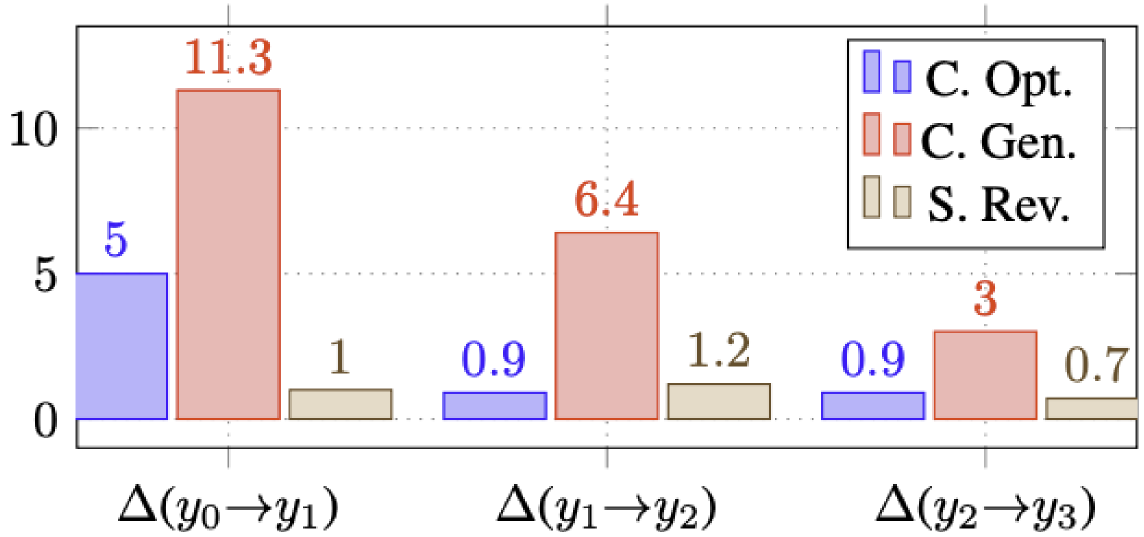
Task	SELF-REFINE feedback	Generic feedback	No feedback
Code Optimization	**27.5**	26.0	24.8
Sentiment Reversal	**43.2**	31.2	0
Acronym Generation	**56.4**	54.0	48.0

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.

- Specific, actionable feedback yields superior results compared to generic or no feedback
- Even generic feedback offers some guidance, but targeted, constructive feedback achieves the best outcomes
- The quality of feedback plays a crucial role in enhancing the performance of SELF-REFINE tasks

Effect of Multiple Iterations

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**



Analysis

- Almost no improvement on Math Reasoning due to inability of the model to identify errors
- Self-refine tends to work better on bigger models as shown by GPT-4 having higher improvement compared to GPT-3.5

Effect of Planning Annotations

- Model fine-tuned on $D_{planning}$ using plans from $D_{modular}$ only show minor improvement
- Generated plans are imprecise or incorrect demonstrating planning as a bottleneck
- Fine-Tuned Model showed increased performance when ground truth plans ($\boldsymbol{D^{GT}}$) were generated from the test set and not $\boldsymbol{D_{modular}}$
- The Fine-Tuned model was not capable of synthesizing new plans but followed generated plans correctly

Drawbacks and limitations

- Dependence on LLMs as an oracle for functional equivalence post transformation
- Generating natural language plans had a lot of inconsistencies and remains a bottleneck
- The three-step transformation process introduces significant computational overhead, making the approach less scalable or efficient for large datasets
- Training code generators exclusively on cleaned and modularized datasets might lead to models that perform well on similarly structured code but struggle with more diverse or less structured datasets
- Enhancing planning in code generation through planning annotations saw limited improvements due to the inherent complexity of algorithmic reasoning

Q & A