

# SELF-REFINE

Iterative Refinement with Self-Feedback

LLM 이 스스로 피드백하고 , 스스로 고쳐 쓴다

발표 : 심동혁

# LLM 의 첫 출력이 최선이 아닐 때가 많다 .

## 사람의 글쓰기

“Send me the data ASAP”

↓ 무례하다는 걸 스스로 깨닫고 수정

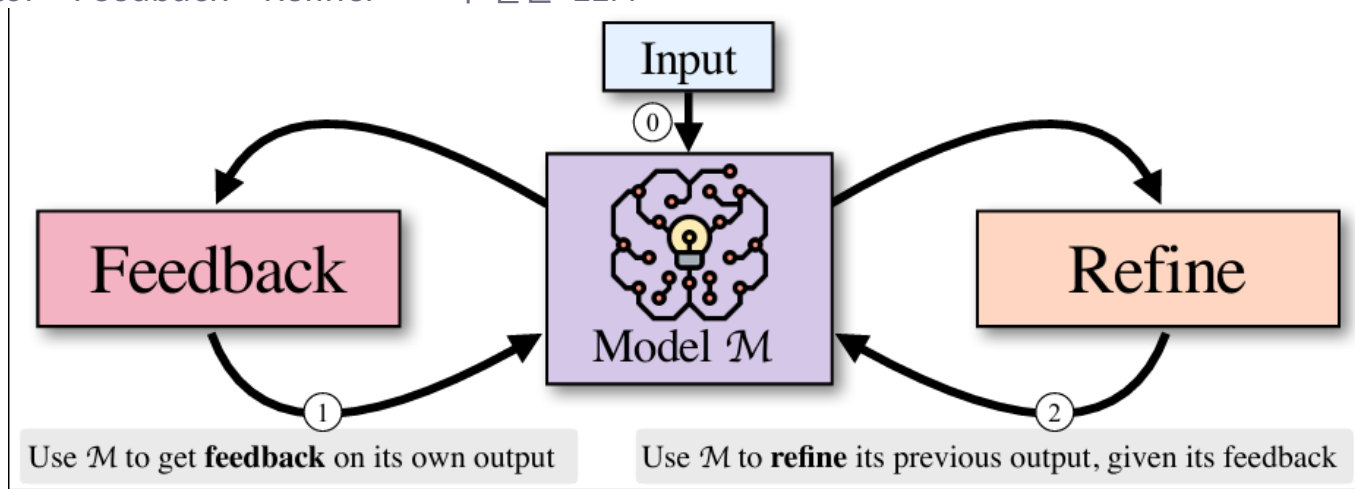
“Hi Ashley, could you please send me the data at your earliest convenience?”

## 기존 반복 개선 방법의 한계

- 도메인마다 별도의 개선 (refiner) 모델을 학습해야 함
- 보상 모델 · 사람의 주식 등 대규모 외부 감독 데이터 필요
- 비용이 크고 , 새 태스크에 바로 적용하기 어려움

# 핵심 아이디어 : 하나의 모델이 세 가지 역할

Generator · Feedback · Refiner = 모두 같은 LLM



## 학습 불필요

추가 학습 · 강화학습 없음 ( 사람이 만든 few-shot 예시만 사용 )

## LLM 1 개

생성 · 피드백 · 수정을 같은 모델이 수행

## 프롬프트 3 개

p\_gen, p\_fb, p\_refine (few-shot 예시 )

# 알고리즘 : 생성 → ( 피드백 → 수정 ) 반복

|| 는 이어붙이기 (concatenation). 모든 단계가 같은 모델 M 호출



초기 생성  $y_0 = M(p_{\text{gen}} \parallel x)$



피드백  $\text{fb}_t = M(p_{\text{fb}} \parallel x \parallel y_t)$



수정  $y_{t+1} = M(p_{\text{refine}} \parallel x \parallel y_0 \parallel \text{fb}_0 \parallel \dots \parallel y_t \parallel \text{fb}_t)$

② 에는 이전 출력과 피드백 이력 전체가 들어간다 → 같은 실수를 반복하지 않도록

## 멈춤 조건 $\text{stop}(\text{fb}, t)$

- 정해진 반복 횟수 도달 ( 주로 최대 4 회 , 태스크별 3~5 회 )
- 또는 피드백 안의 정지 신호 ( 예 : 점수 , “ 더 고칠 것 없음 ” )
- 최종 출력 = 마지막 수정본 ( 다면 평가 태스크는 최고 점수 출력 )

---

**Algorithm 1** SELF-REFINE algorithm

---

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

- 1:  $y_0 = \mathcal{M}(p_{\text{gen}} \| x)$  ▷ Initial generation (Eqn. 1)
- 2: **for** iteration  $t \in 0, 1, \dots$  **do**
- 3:      $fb_t = \mathcal{M}(p_{\text{fb}} \| x \| y_t)$  ▷ Feedback (Eqn. 2)
- 4:     **if**  $\text{stop}(fb_t, t)$  **then** ▷ Stop condition
- 5:         **break**
- 6:     **else**
- 7:          $y_{t+1} = \mathcal{M}(p_{\text{refine}} \| x \| y_0 \| fb_0 \| \dots \| y_t \| fb_t)$  ▷ Refine (Eqn. 4)
- 8:     **end if**
- 9: **end for**
- 10: **return**  $y_t$

---

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

# 예시로 보기 : 코드 최적화

1 부터 N 까지의 합

초기 출력  $y_0$

```
def sum(n):  
    res = 0  
    for i in range(n+1):  
        res += i  
    return res
```

피드백  $fb_0$

이 코드는 for 루프로 하나씩 더하는  
brute force 라 느리다 . 공식  
 $n(n+1)/2$  를 쓰는 게 낫다 .

수정 출력  $y_1$

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

## 좋은 피드백의 두 가지 조건

### Actionable ( 실행 가능 )

무엇을 하면 좋아지는지 구체적 행동 제시 → 공식을 사용

### Specific ( 구체적 )

고쳐야 할 위치를 정확히 지목 → for 루프

# 실험 설정 : 7 개의 서로 다른 태스크

자연어 생성부터 코드 , 수학까지 — 비교 대상은 “같은 모델의 1 회 생성”

## 감성 반전 (Sentiment reversal)

리뷰 1,000 개 · 선호도

## 대화 응답 생성 (Dialogue Response)

대화 372 개 · 선호도

## 코드 최적화 (Code Optimization)

프로그램 1,000 개 · 최적화 비율

## 코드 가독성 개선 (Code Readability)

프로그램 300 개 · 선호도 / 변수명

## 수학 추론 (Math Reasoning)

1,319 문제 (Gsm8k) · 정답률

## 약어 생성 (Acronym Generation)

250 개 · 선호도

## 제약 문장 생성 (Constrained Generation)

200 개 · 키워드 포함률

## 기반 모델

GPT-3.5 (text-davinci-003)

ChatGPT (gpt-3.5-turbo)

GPT-4

## 평가 지표

- 절대 지표 : 정답률 , 최적화 비율 , 키워드 포함률
- 선호도 : 같은 모델의 초안 vs 수정본 선택비율 ( 무승부 별도 )
- Table 1 의 선호도는 GPT-4 가 평가 ( 사람 평가는 부록 )

## 결과 : 대부분의 태스크에서 Self-Refine 이 우세

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

# 왜 수학 추론은 거의 안 올랐나 ?

고치는 능력이 아니라, “틀렸다는 걸 알아차리는 능력” 의 문제

## 94%

ChatGPT 가 수학 풀이에 준 피드백 중 “everything looks good” 의 비율

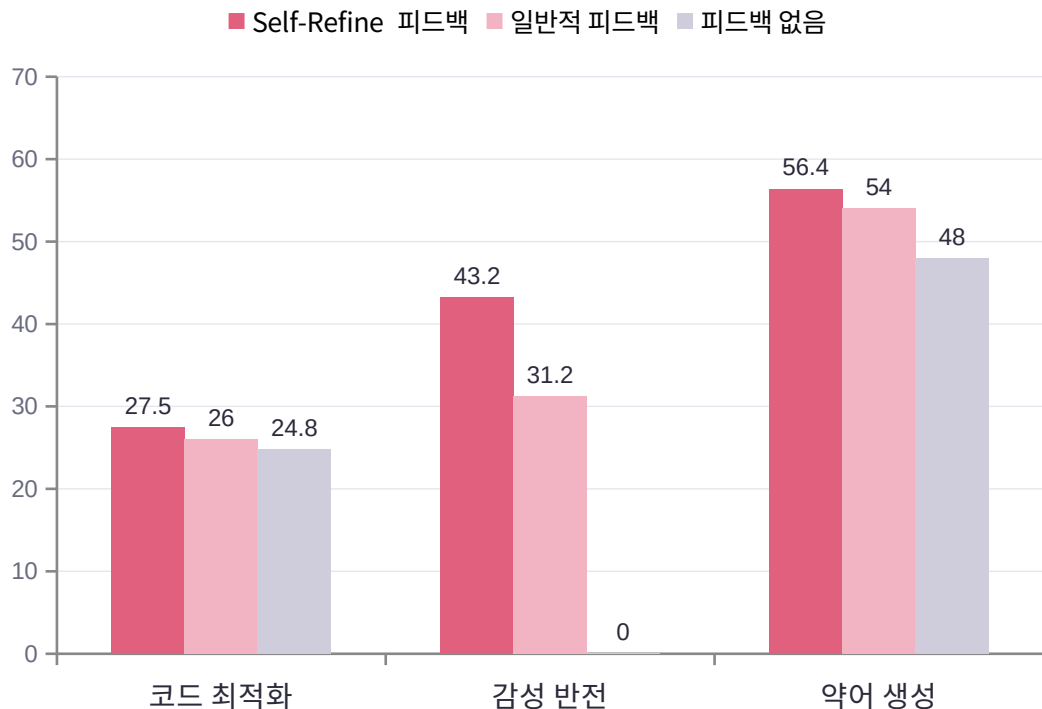
그렇듯한 풀이 과정에 속아 오류를 못 찾음 — 오류는 한 줄, 연산 하나에 숨어 있는 경우가 많다

외부 신호 (Oracle) 로 “틀렸다” 만 알려주면 ?

| 모델      | Self-Refine | +Oracle |
|---------|-------------|---------|
| GPT-3.5 | 64.1        | 68.9    |
| ChatGPT | 74.8        | 76.2    |
| GPT-4   | 93.1        | 93.8    |

# 분석 ① 피드백의 질이 성능을 좌우한다

Self-Refine feedback vs Generic feedback vs No feedback



## 구체적 피드백

“for 루프 안의 반복 계산을 피하라”

## 일반적 피드백

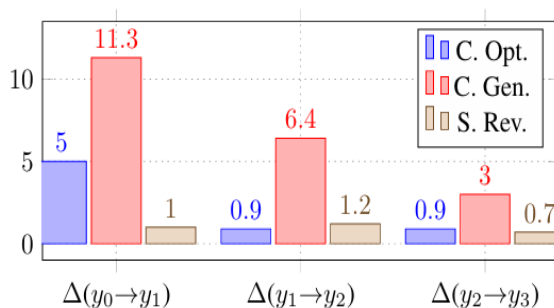
“코드 효율을 개선하라”

## 결론

반복해서 다시 쓰는 것 자체보다는 무엇을 고칠지 알려주는 피드백이 핵심

## 분석 ② 반복할수록 좋아지지만 , 효과는 점점 줄어든다

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



### 수확 체감 (diminishing returns)

첫 반복에서 가장 크게 오름 . 제약 생성  
 $+11.3 \rightarrow +6.4 \rightarrow +3.0 \rightarrow 1\sim 2$  회로도  
 효과 대부분

| 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         |

### 단조증가는 아니다 .

약어 생성처럼 여러 기준을 동시에 봐야  
 하는 태스크는 한 기준이 좋아지면 다른  
 기준이 나빠지기도 해서 점수가  
 오르락내리락 한다 . 그래서 피드백에서  
 항목별 점수를 매기고 가장 높은 출력을  
 최종적으로 고릅니다 .

## 분석 ③ 추가적인 질문들

그냥 여러 번 뽑으면 ?

**k=4**

ChatGPT 로 4 개를 뽑아 그중 최고와 비교해도 Self-Refine 이 더 좋은 성능 → 효과는 “피드백 기반 수정” 덕분 .

약한 모델에서도 될까 ?

**X**

Vicuna-13B 는 피드백 형식을 못 따르고 , 좋은 피드백을 줘도 같은 답 반복 · 환각 대화 → 강한 모델이 전제

실패는 어디서 ?

**94%**

코드 · 수학 실패 사례 35 개 중 피드백 문제가 94% 이다 -> 틀린 수정 제안 (actionable) 61%, 위치 오인 (specific) 33%, 수정 단계 문제는 6%

# 한계

## 논문이 밝힌 한계

- 강한 few-shot· 지시 수행 능력을 가진 모델이 전제
- 비공개 모델만 사용
- 영어 데이터셋만 실험
- 유해한 방향으로도 개선될 수 있음

# 정리

- 1 학습 없이 프롬프트만으로 , 하나의 LLM 이 생성 → 피드백 → 수정을 반복해 출력을 개선한다
- 2 7 개 중 대부분의 태스크에서 향상 . 선호도 태스크에서 특히 크지만 , 평가 방식을 감안해 해석해야 한다
- 3 성능의 열쇠는 피드백의 질 . 스스로 오류를 못 찾는 영역 ( 수학 ) 에서는 외부 신호가 필요하다