

COMP 7607 NLP: Assignment 1

Jiang Feiyu - 3035770800

Files included:

- `test_accuracy.py`
- `zeroshot.baseline.jsonl`
- `fewshot.baseline.jsonl`
- `TreePrompting.jsonl`
- `SelfRefine.jsonl`
- `MethodCombine.jsonl`

Objective

Explore the capabilities of the `Llama-3.1-8B-Instruct` model in **mathematics** domains. Using advanced methods to enhance performance beyond baselines.

Testing Dataset

Grade school math dataset: `GSM8K`

Model

`Meta-Llama-3.1-8B-Instruct`

Evaluation of Inference Cost

1. **wall-clock time** : Sleep time not included. Limited by the speed of API, sleep 3 secs for each iteration.

$$\text{\text{\text{Total Inference Time}}} = \text{\text{end time}} - \text{\text{start time}}$$
2. **average number of generated tokens per question**:
$$\text{\text{Average Tokens per Question}} = \begin{cases} \frac{\text{\text{Total Tokens}}}{\text{\text{idx}}} & \text{\text{if }} \text{\text{idx}} > 0 \\ 0 & \text{\text{if }} \text{\text{idx}} = 0 \end{cases}$$

Evaluation of Accuracy

$$\text{\text{accuracy}} = \frac{\text{\text{no. of correct}}}{\text{\text{no. of data}}} \times 100 \%$$

Format of output

The log file is in `jsonl` format. Each line represents a valid json object. The format of each line is:

```
{
  "input": question from dataset,
  "prompt": [the prompt to the LLM, which is a list],
  "output": the response generated from the LLM,
  "ans": the answer from the dataset,
```

```
"correct_answer_num": the correct number extracted from the answer
"response_num": the LLM generated number extracted from the LLM response
"is_correct": if the LLM generated answer is correct
}
```

Baseline

Zero-shot

```
temperature=0.1,
top_p=0.1,
```

Result:

```
Accuracy: 0.8271417740712661 (1091/1319)
Total inference time (seconds): 1139.1982908248901
Average token count per question: 141.22289613343443
```

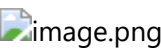
Few-shot

```
temperature=0.1,
top_p=0.1,
```

Result:

```
Accuracy: 0.8483699772554966 (1119/1319)
Total inference time (seconds): 1214.857105731964
Average token count per question: 89.1683093252464
```

Advanced Prompting Method 1: [Self-Refine](#)



Main Principles

The Self-Refine method aims to mimic human cognitive processes by iteratively refining the initial output through feedback. Below are the basic steps of this method:

- 1. **Initial Generation:**
 - Use the language model (M) to generate the initial output $y_{(0)}$.
- 2. **Feedback:**
 - Pass $y_{(0)}$ back to the language model (M) to generate feedback $f_{(b)}$.

3. Optimization:

- Based on the feedback $f_{\{b\}}$, use M to generate the optimized output $y_{\{1\}}$.

4. Iteration:

- Repeat the above steps until the set stopping criteria are met.

Key Advantages

- **No Supervised Training Data Needed:** This method does not rely on any additional supervised data, reducing the complexity of data preparation.
- **No Extra Training or Reinforcement Learning:** It can optimize without additional training, saving resources and time.

Strategy for math problem

The original paper notes that in mathematical reasoning tasks, the improvements of SELF-REFINE are minimal due to the complexity of erroneous judgments.

However, the authors highlight in [Appendix H.1](#) that by experimenting with [Oracle Feedback methods](#), they found that introducing external signals significantly improved the performance of large models in handling mathematical problems.

For instance, this adjustment notably enhanced performance in the Math Reasoning task, with GPT-3 improving by 4.8% and GPT-4 by 0.7%.

This also suggests that if there is an external mechanism to determine whether the current answer is incorrect, the improvements from SELF-REFINE would be even greater.

Strategy improved

1. Initially generate a weak answer using the same zero-shot prompt.
2. Input the weak answer and the question into the LLM to get improvement suggestions.
3. Use the hints and the question to prompt the LLM to generate a better answer, iterating this process up to three times.
4. During the iteration process, if the answer is judged to be correct, break the iteration and return it.

pseudocode

Generate the weak answer first

```
def generate_weak_ans(question):
    here we generate a answer same as the zero-shot
    return weak answer
```

Generate the hint according to the question and the LLM answer

```
def def generate_hint(question, weak_answer):
    query = f'''Question: {question}
```

Now we have a weak answer:

```
<weak answer>
{weak_answer}
<weak answer>
```

You should generate some hints to improve the answer.

Your creteria should include:

```
<criterion>
- The final answer must include "#### [value]" format (e.g., "#### 500").
- Every number used in the steps must be provided by the question. For example,
you should check if any number in the weak answer can not be found in the
question. The intermediate results calculated through the variables in the
question are not counted.
- The answer must directly correspond to the question asked. For example, if the
question asks the number of Apple, the answer should be answer the amout of Apple,
not banana.
- Each step should have a logical explanation justifying its inclusion (e.g.,
formulas).
<criterion>
```

```
<requirements>
- The hint should be less than 50 words.
- The hint should not give the answer directly, so avoid any calculation.
- You Should not return improved answer, or comments to the weak answer.
- The hint should be return in bullet points.
- Hint SHOULD NOT provide steps to solve the question.
<requirements>'''
```

```
generate hint
return hints
```

Using Hint to generate better answer

```
def get_better_ans(question, hints, dialogue):
    query = f'''Here is a Math Question:
<Question>
{question}
<Question>

<Task>
Your task is to solve a series of math word problems by providing the final
answer.
- Show your answer step by step.
- Use the format #### [value] to highlight your answer. For example, if the answer
is 560, you should write #### 560.
<Task>

<Requirements>
- Please answer the question step by step
```

- Please refer the hint for reference
- The solution you provided should be short and clear
- Each setp should less than 50 words, and you should solve this problem in less than 10 steps

<Requirements>

<Answer>

####

<Answer>

Here are some hints for you to reference

<Hint>

{hints}

<Hint>

...

```
generate the better answer
return better answer
```

```
def is_correct_ans(answer, LLM_ans):
    if answer == LLM generated answer:
        return Ture
    else:
        return False
```

The refine process

```
def self_refine(question, answer):

    weak answer = generate_weak_ans(question)

    if is_correct_ans(answer, weak answer):
        return dialogue

    max_iteration = 3
    i = 1
    while i <= max_iter:
        hints = generate_hint(question, weak_ans)

        weak_ans = get_better_ans(question, hints, dialogue)

        if is_correct_ans(answer, weak_ans) == True:
            return dialogue
        i += 1
    return dialogue
```

Self-Refine Result

Accuracy: 0.9378316906747536 (1237/1319)
 Total inference time (seconds): 2620.072146654129


Average token count per question: 258.1849886277483

Advanced Prompting Method 2: [Progressive-Hint Prompting](#)

Main Principles

Progressive-Hint Prompting gradually guides the model toward the correct answer by using previously generated responses as prompts, allowing for multiple automated interactions between the user and the LLM. This method combines generated answers and questions for double-checking and is divided into two phases.

In the first phase, we generate a foundational answer by combining the current question with a base prompt. In the second phase, we generate subsequent answers through corresponding step-by-step prompts, such as the step-by-step chain of thought (PHP-CoT) or the step-by-step complex chain of thought (PHP-Complex CoT). The interaction stops when two consecutive answers are the same.

 屏幕截图 2024-10-28 203405.png

Strategy improved

1. Initially generate a weak answer using the same zero-shot prompt.
2. If the generated answer is correct, return the answer; otherwise, extract the incorrect answer (number) from the LLM's generated content and include it as a hint in the new round of dialogue, e.g.: **Hint: the answer is near ({hints_str})**.
3. The maximum number of iterations is three times, and each round's incorrect answers can be returned to the LLM as hints. For example, if the incorrect answers from the first three rounds are **[1, 8, 6]**, then inform the LLM that **the answer is near (1, 8, 6)**.

pseudocode

Generate the weak answer first

```
def generate_weak_ans(question):
    here we generate a answer same as the zero-shot
    return weak answer
```

Using hint to get better answer

```
def get_better_ans(question, hints, temp):
    hints_str = ", ".join(map(str, hints))
    query = f'''Here is a Math Question: {question}.
    <Hint>
    Hint: the answer is near ({hints_str})
    <Hint>

    <Task>
    Your task is to solve a series of math word problems by providing the final
    answer.
```

```

- Show your answer step by step.
- Use the format ##### [value] to highlight your answer. For example, if the answer
is 560, you should write ##### 560.
<Task>

<Requirements>
- Please answer this question in the format: `We know the Answer Hints: $Hints$.
With the Answer Hints: $Hints$, we will answer the question`.
- Please answer the question step by step
- Please refer the hint for reference
- The solution you provided should be short and clear
- Each setp should less than 50 words, and you should solve this problem in less
than 10 steps
<Requirements>
'''
    return better answer

```

Progressive-Hint process

```

def progressive_hint(question, answer):
    weak_answer = generate_weak_ans(question)
    if is_correct_ans(answer, weak_answer):
        return dialogue

    max_iteration = 3
    hints = []
    hint = extract(weak_answer)
    hints.append(hint)

    i = 1
    while i <= max_iter:
        weak_ans = get_better_ans(question, hints, dialogue)

        if is_correct_ans(answer, weak_ans) == True:
            return dialogue

        hint = extract(weak_answer)
        hints.append(hint)

        i += 1
    return dialogue

```

Progressive-Hint Prompting Result

```

Accuracy: 0.890068233510235 (1174/1319)
Total inference time (seconds): 1747.111938238144
Average token count per question: 515.7073540561031

```

Combine Method: Strategy 1 + 2

Principles

I choose to combine **Progressive-Hint Prompting** and **Self-Refine methods** to optimize the ability of LLMs to generate math problems. The basic idea remains the same as before. To enhance the performance of the code, I opt to generate a weak answer through **few-shot** prompt, while also providing the previously generated answers and hints generated by the LLM regarding the problem for its reference. The specific steps are as follows:

1. Generate a weak answer through few-shot learning.
2. If the weak answer is incorrect, extract its erroneous components and add them to the hints list. Meanwhile, provide the incorrect solution process and the original problem to the LLM to generate hints. Both the hints list and the hints generated by the LLM are returned simultaneously, and this process iterates. The maximum number of iterations is set to 3.
3. To encourage the model to generate more creative answers, set `top_p = 0.2`.

pseudocode

Generate the weak answer first

```
def generate_weak_ans(question):  
    here we generate a answer same as the few-shot  
    return weak answer
```

Generate Hint

```
def generate_hint_msg(question, weak_answer):  
    query = f'''Question: {question}  
    Now we have a weak answer:  
  
    <weak answer>  
    {weak_answer}  
    <weak answer>  
  
    You should generate some hints to improve the answer.  
  
    Your creteria should include:  
  
    <criterion>  
    - The final answer must include "#### [value]" format (e.g., "#### 500").  
    - Every number used in the steps must be provided by the question. For example,  
    you should check if any number in the weak answer can not be found in the  
    question. The intermediate results calculated through the variables in the  
    question are not counted.  
    - The answer must directly correspond to the question asked. For example, if the  
    question asks the number of Apple, the answer should be answer the amout of Apple,  
    not banana.
```



```

- Each step should have a logical explanation justifying its inclusion (e.g.,
formulas).
<criteria>

<requirements>
- The hint should be less than 50 words.
- The hint should not give the answer directly, so avoid any calculation.
- You Should not return improved answer, or comments to the weak answer.
- The hint should be return in bullet points.
- Hint SHOULD NOT provide steps to solve the question.
<requirements>'''

```

Get better answer

```

def get_better_ans(question, hints, hint_msg, temp):
    hints_str = ", ".join(map(str, hints))

    query = f'''Here is a Math Question:
<Question>
{question}
<Question>

<Task>
Your task is to solve a series of math word problems by providing the final
answer.
- Show your answer step by step.
- Use the format ##### [value] to highlight your answer. For example, if the answer
is 560, you should write ##### 560.
<Task>

<Requirements>
- Please answer the question step by step
- Please refer the hint for solution
- The solution you provided should be short and clear
- Each setp should less than 50 words, and you should solve this problem in less
than 10 steps
<Requirements>

Here are some hints for you to reference:

<Hint>
Hint: the answer is near ({hints_str})
{hint_msg}
<Hint>
'''

    return better answer

```

Combine the methods

```
def combine_method(question, answer):  
  
    max_iter = 3  
    temp = []  
    hints = []  
  
    weak_ans = generate_weak_ans(question)  
  
    if is_correct_ans(answer, weak_ans) == True:  
        return dialogue  
  
    hint_num = extract_ans_from_response(weak_ans)  
    hints.append(hint_num)  
    hint_msg = generate_hint_msg(question, weak_ans)  
  
    i = 1  
    while i <= max_iter:  
        weak_ans = get_better_ans(question, hints, hint_msg, temp)  
        if is_correct_ans(answer, weak_ans) == True:  
            return dialogue  
  
        i += 1  
  
        hint_num = extract_ans_from_response(weak_ans)  
        hint_msg = generate_hint_msg(question, weak_ans)  
        hints.append(hint_num)  
  
    return dialogue
```

Combine Method Result

Accuracy: 0.9241849886277483 (1219/1319)
Total inference time (seconds): 2105.558645963669
Average token count per question: 606.9529946929492

Overview

Results of Accuracy:

zero shot	few shot	Self-Refine	Progressive-Hint	Combine Methods
0.82714	0.84837	0.93783	0.89007	0.92418
5	4	1	3	2

Results of Inference Time

zero shot	few shot	Self-Refine	Progressive-Hint	Combine Methods
-----------	----------	-------------	------------------	-----------------

zero shot	few shot	Self-Refine	Progressive-Hint	Combine Methods
1139.20	1214.86	2620.07	1747.11	2105.56
1	2	5	3	4

Results of Average token

zero shot	few shot	Self-Refine	Progressive-Hint	Combine Methods
141.22	89.17	258.18	515.71	606.95

Conclusion

- The combination of methods can significantly improve the accuracy of the baseline, but it does not surpass the performance of the Self-Refine method alone. This suggests that not all optimizations lead to cumulative effects when combined.
- Iteration can increase accuracy, but at the cost of increased inference time.
- External signals provide better guidance than the model's self-referencing of previous outputs.
- In summary, when choosing a method, one should consider the specific goals between accuracy and response generation efficiency.

Future Research Directions

While increasing the number of iterations can improve accuracy, it is necessary to study the specific impact of different iteration counts on inference time and resource consumption in order to find the optimal balance.

Reference:

1. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegrefe, S., ... & Clark, P. (2024). Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

2. Zheng, C., Liu, Z., Xie, E., Li, Z., & Li, Y. (2023). Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv:2304.09797*.