ARC 문제 해결을 위한 프롬프트 엔지니어링의 가능성

심우창[1o] · 진혜빈[2] · 김세진[1] · 김선동[1Ψ]
광주과학기술원 AI대학원[1], 광주과학기술원 전자전기컴퓨터공학부[2]
{woochang, hyebiness01}@gm.gist.ac.kr, {sejinkim, sundong}@gist.ac.kr

Engineering Prompts for the ARC Challenge

Woochang Sim[1o] · Hyebin Jin[2] · Sejin Kim[1] · Sundong Kim[1Ψ]
GIST AI[1], GIST EECS[2]

요 약

ARC(Abstraction and Reasoning Corpus)[1]는 범용인공지능을 평가하는 중요한 벤치마크 데이터셋이다. 하지만 ARC 벤치마크 데이터셋에서 좋은 성능을 보여주는 연구가 지금까지 나오지 못하고 있다. 본 논문에서는 사전 학습된 대규모 언어 모델과 프롬프트 엔지니어링을 활용한 범용인공지능 개발의 가능성을 확인해 보고자 했다. 실험 결과, 객체 유형의 문제에 대해서는 정확도가 낮았으나 그 외의 유형에서는 정확도가 높았다. 또한 객체 유형 문제에서 적절한 프롬프트 엔지니어링 방법을 적용했을 때 정확도가 향상되었으며, 이를 통해 프롬프트 엔지니어링의 가능성을 확인해 볼 수 있었다. 향후 연구에서는 이런 점을 보완하기 위해 객체와 객체 간의 관계, 객체의 움직임, 문제 유형 등을 추가한 다양한 프롬프트 엔지니어링 방법을 사용한다면 성능이 개선될 것으로 예상된다.

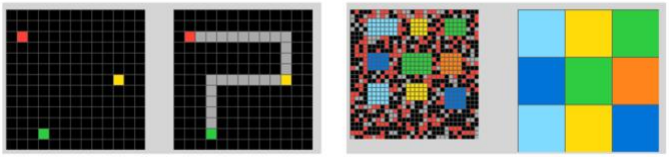
1. 서론

인공지능 기술은 특정 도메인에 적합한 모델을 개발하는 방향으로 발전해 왔다. 이로 인해 한 번 만든 모델이 범용적으로 사용될 수 없다는 단점이 있었다. 만약 범용인공지능이 개발이 된다면 이러한 단점을 극복하여 비용측면에서 큰 효과를 불러올 수 있다. 더 나아가 추상화와 추론을 할 수 있는 범용인공지능은 인공지능 기술의 새로운 혁신을 불러올 것으로 보인다.

기존의 ARC 벤치마크를 활용한 범용인공지능 연구들은 도메인 특화 언어(Domain Specific Language, 이하 DSL)를 적용하는 연구[2] 위주로 진행되었다. 한편, 최근 방대한 양의 데이터를 이용해 다양한 정보를 미리 학습한 대규모 언어 모델(Large Language Model)이 많은 분야에서 활용되고 있다. 본 논문은 기존에 ARC 벤치마크에 DSL을 적용한 연구들과 다르게 대규모 언어 모델을 활용하고자 하며, 프롬프트 엔지니어링이 다양한 도메인에 적용[3] 되는 흐름에 맞추어 프롬프트 엔지니어링을 ARC 벤치마크에 적용해 보고 그 가능성을 확인해 보고자 한다.

1.1 ARC 벤치마크 데이터셋

ARC 벤치마크는 인공지능 모델의 추상적인 사고와 추론 사고를 평가하기 위해 만들어진 벤치마크 데이터셋이다[1]. ARC 벤치마크 데이터셋에는 다양한 패턴의 여러 문제가 존재하며, 문제 당 3~4개 정도의 예제와 출력 값을 맞춰야 하는 하나의 입력 값이 제공된다. 각 예제는 2차원 배열 형태인 입력 값과 출력 값의 쌍으로 이뤄져 있다. 모델은 주어진 예제들을 통해 해당 문제에서의 패턴을 파악하고, 맞춰야 할 입력 값에 대한 적절한 출력 값을 추론해야 한다. [그림 1]은 배열 형태인 입력 값과 출력 값을 시각화한 ARC 문제에 대한 예제들이다.



[그림 1] ARC 문제 예제들의 예시

1.2 프롬프트 엔지니어링

프롬프트 엔지니어링이란 특히 자연어처리 분야에 해당하는 인공지능의 한 개념으로, 인공지능이 역량을 발휘할 수 있도록 적합한 지시어를 내려주는 역할을 한다. 자연어 처리 분야에서 GPT 모델은 문장 생성 부분에서 뛰어난 성능을 보였으나, 잘못되거나 편향된 정보가 포함된 문장을 생성한다는 한계점을 보였다. 때문에 대규모 언어 모델에서 언어 모델의 패러다임은 모델을 미세 조정하는 모델 엔지니어링 방식에서 모델에게 적합한 지시를 내리는 프롬프트 엔지니어링 방식으로 옮겨가고 있다.

생성형 AI 분야에서 프롬프트는 대규모 언어 모델로부터 결과물을 생성하기 위한 입력을 의미한다. 프롬프트를 구성하는 방식에 따라 결과물의 품질이 크게 달라질 수 있기에 좋은 프롬프트를 구성하는 프롬프트 엔지니어링의 중요성이 부각되고 있으며, 이를

[1] 이 논문은 2023년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (RS-2023-00216011, 사람처럼 개념적으로 이해/추론이 가능한 복합인공지능 원천기술 연구)

활용한 기초 모델의 사용이 기대되고 있다. 본 연구 역시 프롬프트 엔지니어링을 활용한 ChatGPT(GPT-4.0)를 통해 ARC 벤치마크를 해결하고자 했다.

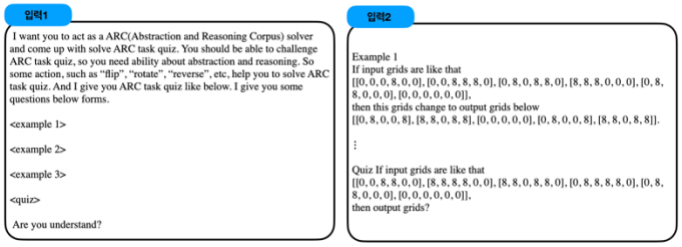
2. ARC 문제에 프롬프트 엔지니어링 적용

2.1 대규모 언어 모델 선택 배경

공개되어 있는 대규모 언어 모델들 중 ChatGPT의 GPT-4.0을 선택한 이유는 편의성과 성능 때문이다. 우선 ChatGPT는 웹서비스와 API서비스를 제공하기 때문에 ARC 벤치마크 데이터셋에 쉽게 적용할 수 있다. 또한 ARC 문제와 같은 퓨샷(few-shot) 상황일 때, GPT-4.0의 성능이 GPT-3.5를 포함한 다른 대규모 언어 모델들보다 대부분의 벤치마크에서 월등한 성능을 보여주었고, 심지어 일부 벤치마크에서는 미세 조정한 SOTA(State-of-the-Art) 모델보다 뛰어났다[4].

2.2 페르소나 프롬프트 엔지니어링

ChatGPT를 보다 잘 활용하기 위해서 널리 알려진 프롬프트 엔지니어링 방법 중 하나인 페르소나 프롬프트를 사용했다. 페르소나 프롬프트는 ChatGPT가 ARC 문제에 초점을 맞출 수 있도록 ARC 문제에 대한 정보들과 수행해야 하는 역할을 프롬프트에 명시하는 방법이다. 본 연구는 ARC 문제를 풀기 전에 페르소나 프롬프트를 우선 입력했다. 그 후, ARC 문제에 대한 예제들과 출력 값을 맞춰야 하는 입력 값을 주었다. 이때 모든 입력 값과 출력 값은 2차원 배열 형태로 주었으며, ChatGPT가 맞춰야 하는 입력 값의 적절한 출력 값을 도출하도록 프롬프트를 구성했다.



[그림 2] 실험에 사용된 프롬프트 (문제2)

2.3 ARC 예제를 특수문자로 표현하는 방법

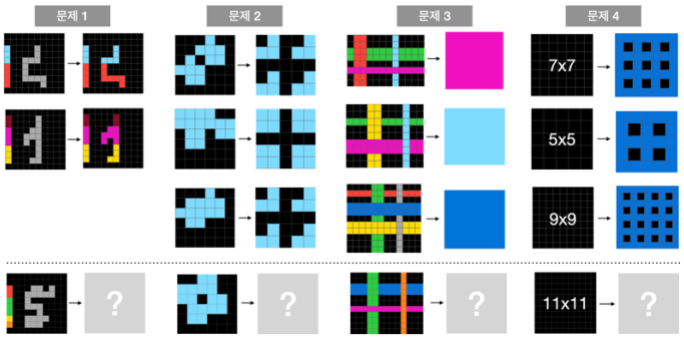
사전에 다양한 프롬프트를 사용해 ARC 문제 해결을 시도했을 때, 색상을 숫자로 표현할 경우 ChatGPT가 해당 숫자를 계산하는 경우가 있었고, 알파벳으로 표현할 경우 단어로 인식해 토큰화하는 문제가 발생했다. 이 문제를 해결하기 위해 색상을 특수문자로 표현하는 방법을 고안했다. ARC 문제의 배열 원소로 숫자를 사용한 경우, 배열 원소는 0~9 사이의 값을 가지며 각 숫자는 서로 다른 색상을 나타냈다. 반면 특수문자를 사용할 경우, 총 10가지의 특수문자(■□○◇¤♤♧△♡☆)로 색상을 표현했다.

3. 실험

3.1 실험 설정

OpenAI의 API로 davinci모델을 ARC 학습 데이터셋을 가지고 1에폭 미세 조정을 시켜준 후, 해당 모델이 평가 데이터셋에서 맞춘 문제 4개를 실험에 사용했다. 해당 문제들을 ChatGPT가 어느 정도로 정확하게 풀 수 있는지, 그리고 배열 원소를 숫자 또는 특수문자로 주었을 때 어떤 영향을 미치는지 실험했다. 정확한 실험을 위해 각 문제별로 총 10회의 테스트를 반복했다.

실험에 사용한 4 문제의 대표적 예제는 아래 [그림 3]과 같다.



[그림 3] 실험에 사용한 4 문제의 예제

문제 1은 회색 부분의 값을 해당 행의 첫 번째 원소의 색상으로 바꿔주는 문제로, 행의 개념을 이해하고 있다면 풀 수 있다. 문제 2는 입력 값에서 4개의 같은 모양으로 나눌 수 있는 도형을 찾은 후 해당 도형을 좌측 상단, 우측 상단, 좌측 하단, 우측 하단에 배치해야 하는 문제로, 객체라는 개념을 이해해야 풀 수 있다. 문제 3은 행 혹은 열 중에서 중간에 끊어지지 않고 끝까지 이어진 색상을 찾는 문제로, 행과 열의 개념을 이용해야 풀 수 있다. 문제 4는 배열의 행과 열이 주어졌을 때 홀수 행에는 파란색으로 다 채우고 짝수 행에는 파란색과 검은색을 번갈아 가며 채우는 문제로, 패턴을 그리드에 맞게 채우는 문제이다.

3.2 실험 결과

[표 1] 실험 결과

		맞춘 횟수	틀린 횟수	정확도
문제 1	숫자	8	2	80%
	특수문자	2	8	20%
문제 2	숫자	0	10	0%
	특수문자	4	6	40%
문제 3	숫자	8	2	80%
	특수문자	9	1	90%
문제 4	숫자	10	0	100%
	특수문자	10	0	100%

위의 [표 1]에 정리된 실험 결과를 통해 알 수 있는 유의미한 부분은 크게 2가지이다. ChatGPT가 풀기 어려워하는 유형의 문제를 파악했다는 점과 배열 원소 기호와 문제 유형에 따라 성능의 변화가 있다는 점이다.

예를 들어 문제 4는 배열 원소의 종류에 상관없이 100%의 정확도를 달성한 반면, 문제 2는 평균 20%의 가장 낮은 정확도를 기록했다. 이를 통해 ChatGPT는 문제 4와 같이 반복되는 패턴을 파악하고 적용하는 문제는 쉽게 풀 수 있지만, 문제 2와 같이 객체를 찾고 활용하는 문제는 풀기 어렵다는 사실을 알 수 있다.

특히, 문제 2와 같이 객체를 찾고 이를 활용하는 문제에서는 배열 원소가 특수문자일 때의 성능이 더 좋았다. 반면 문제 1과 같이 행이라는 개념을 이해하고 풀어야 하는 문제에서는 특수문자보다 숫자일 경우에 정확도가 더 높다는 것을 통해, 문제에 따라 적절한 배열 원소를 입력하는 것이 성능에 영향을 준다는 결론을 얻었다.

3.3 심층 분석

배열 원소의 종류에 따른 ChatGPT의 문제 접근법을 파악하고자, 정확도에 큰 변화가 있었던 문제 1과 문제 2의 경우를 추가로 분석해 보았다.

문제 1에서 ChatGPT의 접근법은 총 6가지가 있었다. (1) 기호(5, ♧)를 해당 행의 첫 번째 열의 숫자 혹은 특수 문자로 대체한다. (2) 기호(5, ♧)에 대해서 특정 기호(0, 5, ■, ♧)를 제외한 가장 가까운 기호로 대체한다. (3) 예제에서 관측된 패턴을 그대로 적용한다. 예를 들어 [2, 0, 5, 5]가 입력 값이고 출력 값이 [2, 0, 2, 2]일 때, 풀어야 할 문제의 첫 번째 열의 값이 2라면 [2, 0, 2, 2]를 넣어준다. (4) 입력 값을 그대로 복제하여 출력 값으로 사용한다. (5) 각 행마다 그 행에 존재하는 유일한 기호로 기호(5, ♧)를 대체한다. (6) 각 행에 존재하는 기호(5, ♧)를 바로 위 행의 유일한 기호로 대체한다.

배열 원소로 숫자를 사용했을 경우, 접근법 (1), (2), (3), (4)를 각각 6회, 2회, 1회, 1회 사용했다. 이 중에서 접근법 (1), (2)로 총 8번의 정답을 맞혔다. 반면 특수문자를 사용했을 경우, 접근법 (1), (2), (4), (5), (6)을 각각 2회, 1회, 1회, 3회, 3회 사용했으며, 이 중에서 접근법 (5)로 총 2회 맞혔다.

배열 원소를 숫자로 사용했을 경우에는 주로 접근법 (1)을 사용했지만, 특수 문자를 사용했을 경우에는 주로 접근법 (5), (6)과 같이 각 행의 유일한 문자로 ♧를 대체하는 경우가 많았다. 그러나 접근법 (6)과 같이 다른 행의 유일한 문자로 대체해서 잘못된 출력 값을 도출하는 경우도 있었다. 그리고 올바른 접근법 (5)를 선택했지만 이를 제대로 적용하지 못해서 부적절한 출력 값을 도출하는 경우도 있었다.

문제 2에서 ChatGPT의 접근법은 배열 원소의 종류에 관계없이 두 가지로 나눌 수 있었다. 첫 번째 접근법은 각 행과 열을 제거, 교환, 변환, 삽입 등의 변형 과정을 통해 5x5 형태를 갖춘 후 가운데 행이나 열을 기호(0, ■)로 채우는 방식이며, 다소 복잡한 과정을 거치기에 정답을 맞히는 비율이 낮았다. 두 번째 접근법은 예제의 출력을 그대로 출력하는 방식이다. 문제 2의 경우 예제에서 제시한 출력 값이 2종류밖에 없었기에 이러한 접근법을 시도했던 것으로 예측되었다. 이 경우는 결과를 도출하는 과정에 별다른 연산이 필요하지 않기에 상대적으로 높은 성능을 보였다.

다만 배열 원소로 숫자를 사용한 경우에는 첫 번째 접근법을, 문자를 사용한 경우에는 두 번째 접근법을 채택한 비율이 높았기에 두 가지 프롬프트 방식에 따른 성능 차이를 보였다. 또한 이러한 결과를 통해 숫자 형태의 입력은 ChatGPT가 연산을 시도하도록 유도하고, 문자 형태의 입력은 ChatGPT가 배열을 하나의 그림처럼 인식하도록 유도한다고 유추했다.

4. 결론 및 향후 연구

본 논문은 ARC 문제를 풀기 위해 기존 연구들이 활용한 DSL을 적용하는 대신에 프롬프트 엔지니어링 방법을 활용했다. 실험 결과를 통해 ARC 문제를 일부 풀 수 있는 것을 확인했으며, 이를 통해 ARC 문제를 풀 수 있는 새로운 가능성을 보였다.

향후 연구에서는 객체나 대칭 이동, 회전과 같은 사전 개념들을 프롬프트에 사전 학습시키면 성능이 더욱 좋아질 것으로 예상된다[1]. 이 외에도 인간이 이해할 수 있는 형태의 프롬프트가 아닌 모델이 학습을 통해서 만든 프롬프트를 적용[5] 및 프롬프트에 문제 유형과 객체 정보들을 추가하는 등의 방법으로 더욱 정교한 프롬프트를 구성한다면 성능 향상에 도움이 될 것으로 기대된다[5, 6, 7].

5. 참고 문헌

[1] François Chollet, “On the measure of intelligence,” arXiv:1911.01547, 2019.

[2] Raphael Fischer et al, “Solving Abstract Reasoning Tasks with Grammatical Evolution”, CEUR-WS:Vol-2738, 2020.

[3] Tuan Dinh et al, “LIFT: Language-Interfaced Fine-Tuning for Non-Language Machine Learning Tasks”, NeurIPS, 2022.

[4] OpenAI, “GPT-4 Technical Report”, arXiv:2303.0877, 2023.

[5] Pengfei Liu et al, “Pre-train, prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing”, ACM:Vol.55, No.9, 2023

[6] Samuel Acquaviva et al, “Communicating Natural Programs to Humans and Machines”, arXiv:2106.07824, 2022

[7] Jason Wei et al, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models”, arXiv:2201.11903, 2023.