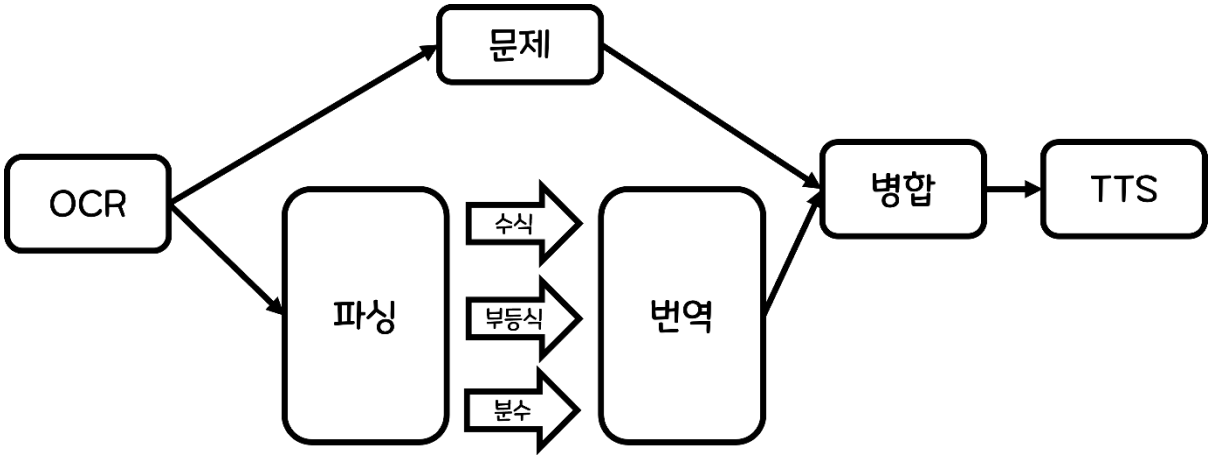
0. 전체적인 구조



1. 이미지를 OCR하면서 문제와 수식을 분리한다.
2. 분리한 수식들에서 부등식과 분수를 추가적으로 분리해낸다.
3. 분리한 모든 식(부등식,분수,수식)을 한국어로 번역한다.
4. 번역한 모든 식들을 원래 자리에 재배치한다.
5. (Optional) 만들어진 텍스트를 토대로 음성을 생성한다.

1. OCR

[설명]

수식을 번역할 때, 수식만 단독으로 있는 것이 문제 전체를 번역하는 것보다 훨씬 성능이 좋았다. 수식을 단독으로 번역하기 위해 OCR 단계에서, OCR를 하는 겸사겸사 문제와 수식을 분리해서 보내주는 역할을 수행하고 있다. GPT-4o의 경우 Multimodal 능력을 가지고 있기 때문에, 동시에 두 가지의 역할을 수행하는 것이 가능했다. 다만, GPT의 OCR 능력이 조금 좋지 않았던 관계로 Google의 Tesseract로 현재 모듈을 전환하였다. 전체적인 OCR능력은 좋아졌으나, 문제에서 수식을 분리하여 정제하는 추가적인 LLM Call이 들어가게 되었다.

Tesseract를 사용할 때 tessearct는 하나의 언어만 제대로 파싱할 수 있으므로, 한국어와 수식을 따로 OCR하게 된다. 이후 수식은 sympy를 통해 LateX로 바꾼다음에 LLM에 넣어주어 최종적으로 JSON 형식으로 정제된 Output을 만들게 된다.

[성능]

- 발표자료 참고.

2. 파싱(분수 및 부등식 분리)

[개요]

가장 많이 번역오류가 발생한 부분은, 분수 및 부등식이었다. 이들은 다른 유형들에 비해서 수가 많았기 때문에, 이들의 번역을 개선함으로써 전체적인 정확도를 올릴 수 있을 것이라 판단했다.

분수는 읽는 방식이 A 나누기 B로 말하거나 분수와 분자의 순서를 바꾸는 등의 경우가 많이 존재했다. 영어와 한국어간의 분수를 읽는 방식이 상이하기 때문으로 추정된다. 부등식은 부등호가 하나만 존재할 때는 잘 읽었지만, 부등호가 두개 이상 붙어있을 때는 입력 Sequence 그대로 읽거나, 요소간의 대소관계를 쉽사리 파악하지 못하게 번역하는 경우가 많이 존재했다.

이는 단순하게 정답을 알려줘서 LLM이 그와 비슷하게 생성하도록 하는 Few-Shot으로 넣어줄 때에는 잘 해결이 되지 않았으나, 분수와 부등식 각각에 특화된 Prompt를 통해 번역했을 때는 복잡한 분수와 부등식도 잘 읽는 것을 확인할 수 있었다. 따라서, 이러한 유형별 특화 Prompt를 거치기 위해서라도 분수와 부등식을 따로 분리해낼 필요가 있었다.

[알고리즘]

분수의 경우는 pylatexenc(https://github.com/phfaist/pylatexenc)라는 모듈을 사용하였다. 이 모듈은 형식이 지정된 부분들을 트리 형태로 파싱해주는 역할을 수행한다. 따라서, 우리는 이 파싱된 부분들을 순회하며 분수들을 찾아서 저장하고, 나머지 부분은 원래 있던 LaTeX 그대로 반환하도록 만들었다.

부등식은 조금 까다롭게 진행되었다. 부등식은 중괄호에 Bound되어 있지 않기 때문에, 우리가 직접 부등식의 경계를 찾아야 한다. 또한 부등식 안에 부등식이 있을 수 있기 때문에, 그 역시 잘 캐치해야 한다. 대략적인 알고리즘은 다음과 같다.

1. 먼저 모든 부등호의 위치를 찾는다.
2. 각 부등호에 대하여 왼쪽과 오른쪽을 찾아보면서 경계를 확정짓는다.
    1. 경계는 한글, 괄호, 또는 특정한 패턴이 될수 있다.
3. 모든 부등호에 대하여 부등식을 확정지었다면, 자신의 하위에 부등식이 있는 것과 없는 것을 나눈다.

4. 다른 부등식을 포함하지 않는 부등식을, 포함하는 부등식에 합쳐가면서, 모든 부등식이 하위에 부등호를 포함하지 않도록 만든다.

[예시]

1 < X < P(X<2)가 있으면, 각 부등식을 적절하게 분리한다.

분리하면 1 < X < P(X<2)과 X<2 총 2개의 식이 나오게 될것이다.

X<2는 하위에 부등호를 포함하지 않는다. {[INEQUAL 1]: X<2}의 형태로 저장한다.

1 < X < P(X<2)에서 현재 확정된 부등식이 있는지 살펴본다. 이 경우에는 [INEQUAL 1]을 가지고 있으므로, 다음과 같이 대체할 수 있다: 1< X < P([INEQUAL 1])

이제 이 식도 하위에 부등호가 없으므로, {[INEQUAL 2]: 1 < X < P([INEQUAL 1])}의 형태로 저장한다. 따라서, 총 두개의 부등식이 대체되어 반환된다.

[성능]

정확성: 현재 데이터셋 2000개 기준으로 부등식 926개, 분수 1910개가 존재한다. 이들에 대한 정답 LaTeX, 즉 올바르게 LaTeX로 분리되었을 때 얼마나 분수와 부등식을 잘 분리해내는가에 대해 측정했는데, 파싱 모델은 오류가 존재하지 않았다.

시간: LLM에 수식 하나 넣고 정답을 가져오는 시간보다 전체 데이터셋에서 분수와 부등식을 전부 분리해내는 것이 오히려 더 빠르므로, 신경쓰지 않아도 된다고 판단하였다.

[한계점 & 개선점]

1. 이전 단계의 오류에 취약
   A. OCR이 항상 완벽할 수 없기에, 잘못된 형태의 수식이 추출될 수 있다. 그럴 경우에는, 제대로 파싱이 되지 않는다. 다만 치명적인 오류가 아니라면, 파싱만 되지 않을 뿐, 파이프라인은 정상적으로 동작하여서 어느 정도 정상적으로 동작할 수 있다.
2. 부등식 파싱의 문제점
   A. 현재 부등식을 분리하는 모델은, 재귀적으로 부등식을 분해해주는 모듈을 찾을 수 없어서 직접 구현한 모델이다. 그렇다 보니 현재의 데이터셋에서는 문제가 없지만, 잠재적으로는 문제가 발생할 수 있다.
   B. 부등식이 어디에서 끝나는지 경계를 알기 어렵다. 일반적이 사람이 부등식을 볼때는 글꼴이나 띄어쓰기, 또는 문맥을 통해 쉽사리 부등식의 범위를 알 수 있지만, LaTeX로 변형된 부등식에서는 이를 알기 쉽지 않다. (i) X < 1과 같은 예의 경우, 저 괄호가 번호인지 곱셈인지 판별하기 쉽지 않다. 또한, 지금까지 보지 못했던 기호들은 처리가 되어있지 않기에, 부등식에 포함

시키면 안 된다 해도 현재 모델은 포함시킬 것이다.

C. 다만 파싱을 LLM에 맡기면 복잡한 형태의 부등식은 제대로 파싱하지 못했기에(GPT-4o기준), 기계적으로 분리해내는 방향이 맞다고 생각된다.

3.번역

[개요]

OCR에서 나온 수식과, 파싱 단계에서 추가로 생성된 부등식과 분수를 전부 한국어로 번역하는 단계이다.

[설명]

일반적인 수식: 과정없이 LLM이 자동으로 번역하게 한다.

분수: 분자와 분모를 나누어서, 분모와 분자의 자리를 바꾼 다음 그 사이에 "분의"를 붙이는 과정을 따르도록 프롬프트를 설정하였다.

부등식: 부등호를 기준으로 모든 구성요소를 나눈 다음, 각각의 구성요소를 차례차례 읽되, 부등호 하나하나가 식이 되도록 읽게 프롬프트를 설정하였다.

공통: 전 단계의 파싱으로 인해 수식 중간중간에 [INEQUAL 1]이나 [FRAC 1] 등의 라벨이 붙어 있다. 이들은 절대 번역하지 말도록 프롬프트를 수정하였다.

자세한 프롬프트는 models/prompts.py에 위치하여 있다.

<Few-Shot>

- 퓨샷은 연립방정식, 복잡한 형태의 지수, 적분, 선분, 파싱 라벨처리로 이루어져 있다.
- 퓨샷은 많이 넣으면 오히려 역효과가 날 수 있기에, 가장 많이 등장하면서도 잘 안 되는 유형들만 집어넣었다. 또한 파싱한 라벨들은 번역하면 안 되므로, 이에 대한 퓨샷도 필수로 넣어 주었다.

<RAG>

- 실제로 오류가 발생한 문제들은 한 유형에 국한되어 있지 않고, 여러 유형에 골고루 나왔다. 이를 전부 퓨샷으로 하는 것은 오히려 원하는 결과를 내지 못할 수도 있어서, 특정 예시와 비슷한 수식을 찾아올 수 있는 RAG를 사용하는 것이 낫다고 판단하였다.
- 수식과 번역을 key-value의 형태로 저장해놓아서 key와 현재 수식간의 유사도를 측정하여 value를 retrieval한다.

[성능]

2000개에서 200개 랜덤 샘플링 기준.

정확도:

	Few-shot	RAG
에러 개수	24개	20개

RAG를 사용했을 때 복잡한 식이 들어있는 지수나, 괄호의 표기 여부에서 기존보다 나아진 결과를 보였음. 아무래도 고정되어 있는 Few-shot보다 조금 더 관련된 예시를 찾아왔기 때문으로 추정.

[그 외 리스트]

예시를 넣어 해결될 수 잇는 것: 합성함수, 조건부 확률, 자연로그, 명제

가끔씩 잘 안되는 것: 대소문자, 언더바, 요소 누락, 순서쌍

시간: 평균적으로 6 API Call / Question.

[한계점 & 개선점]

1. Slow
    A. 현재 구현상으로는 모든 수식에 대하여, 또 그 안에 들어있는 모든 자잘한 분수 및 부등식에 대하여 LLM을 사용하고 있으나, 이로 인해 전체적인 파이프라인이 느려지고 있다.
    B. 우리가 현재 데이터를 주고받는 형태인 JSON형태 그대로 LLM에 입력으로 넣는 것도 방법 중 하나이다. 그렇게 하면, 분수와 부등식을 제외한 일반적인 수식들은 하나의 API Call로 전부 번역할 수 있다.
    C. 초기에는 그렇게 진행했으나 Few-shot을 넣기 어렵고, 이러한 특이한 JSON 형태의 구조가 번역에 문제를 끼칠 수 있는 것은 물론, 가끔씩 JSON형태를 제대로 맞추지 않은 Output이 나와서 파이프라인이 중간에 멈추는 현상이 많이 발생한 바, 현재의 형식으로 바꾸었다.
2. Unstable
    A. 파싱하여 분수와 부등식을 분리한 것은 좋지만, 이들이 원래 어디 있었는지를 알기 위해서 원래 수식에 [FRAC 1]과 같은 라벨을 붙이는 것이 필연적이게 되었다.
    B. 퓨샷 및 프롬프트를 통해 이들의 번역을 어느정도 막을 순 있지만, 완벽하지는 않다. 라벨이 번역되면 다음 단계에서 제대로 수식이 교체되지 않기

때문에 고쳐야 할 문제중 하나이다. 가장 쉬운 방법은, 번역된 수식에서 라벨들이 전부 다 들어 있는지 확인하고, 없으면 제대로 될때까지 재시도하는 방법이 있겠다.

C. 현재 번역 오류의 경향성을 보았을 때는 여러 유형에 조금씩 안되는 것으로 보인다. 각 유형별로 확인했을 때, 같은 유형인데도 잘 되는 것과 안되는 것이 갈리는 것을 확인할 수 있었다. Self-consistency처럼 여러 답변을 생성해서 그 중 주류인 번역을 택하는 것도 좋은 선택으로 보인다.

3. RAG

A. RAG로 답변을 생성할 때가 Few-shot으로 생성할 때보다 파싱에서 생성된 라벨을 더 자주 번역하는 경향이 있다. 최대한 프롬프트를 바꾸어 가며 방지하려고 했지만, 여전히 빈번하게 번역되고 있다.

B. 퓨샷 DB를 만들때, 파싱했던 라벨이 들어있는 예시 역시 들어가야 한다. 그런 예시가 없으면, 위의 문제가 더 빈번하게 발생한다.

C. 굳이 RAG처럼 벡터를 이용한 유사도 검색이 아니더라도, 특정 연산자나 기호가 들어있는지 검색해서, 대응되는 예시들을 가져오는 방법도 좋을 것으로 생각된다. 현재 우리도 유형을 분류할 때, 적분, 분수, 지수 등등 어떤 기호가 들어있는지에 따라 분류를 하기 때문에 오히려 RAG보다 더 유사한 예시를 찾아올 수 있을 것이다.

4. Merge & Smoothing

[설명]

그동안 분리되었던 수식, 분수, 부등식들을 번역한 후, 전부 원래 자리에 재배치하여 말을 다듬는 과정이다. 원래 수식이 위치해야 할 자리는 [Latex 1], [FRAC 1] 등과 같이 표시가 되어있고, 분수와 부등식 역시 그와 동일한 라벨로 저장되어 있기 때문에 쉽게 Replace할 수 있다.

먼저, 분리된 분수 및 부등식을 각각의 수식에 집어넣는다. 그 다음에 각 수식들을 문제의 원래 위치에 집어넣는다. 그런 다음, 집어넣은 수식들을 LLM을 토대로 양 옆의 문맥과 부드럽게 이어지도록 만든다. 각각의 수식들은 원래 자리에 들어갈 때 대괄호에 쌓여 들어가게 된다. 예를 들어,

'[이분의 일]엑스 제곱 더하기 엑스'

와 같은 식으로 삽입되는 형태이다. 이렇게 함으로써, LLM은 어디에 수식이 삽입되었는지를 쉽게 판별할 수 있다.

모든 수식을 문제에 재삽입했다면, LLM을 거쳐 말을 다듬는다. 주로 고치는 것들은

"[X는 2보다 작다.]일 때, Y는~"

와 같이 수식의 끝이 문맥과 제대로 이어지지 않은 것들이다.

[성능]

2000개의 데이터셋 중, 랜덤으로 200개 샘플하였을 때 기준

	에러발생율	시간
Metric	8%	1 API / Question

에러 리스트:

에러	문제 곡해	요소 누락	문맥 어색
개수	14개	1개	1개

문제 곡해: 문제의 말투를 다듬거나, 문제를 요약 또는 풀이를 작성함.

요소 누락: 병합 과정에서 일부 라텍스가 누락되어서 표시되지 않음.

문맥 어색: 병합 과정에서 텍스트가 제대로 다듬어지지 않았음.

[한계점 & 개선점]

1. Merge의 불안정성
   A. 현재 Merge는 과도한 API의 사용을 막기 위해서 모든 분수와 부등식, 더 나아가 수식까지 문제에 전부 재배치한다음에, LLM이 한꺼번에 대괄호 제거 및 Smoothing하는 역할을 수행하고 있다.
   B. 아무래도 말을 다듬는 과정이기 때문에, 문제를 이상하게 곡해하는 현상이 많이 일어날 수밖에 없다. 이를 해결하기 위해 원래 원본이 되는 이미지나, 단순히 수식의 분리없이 OCR한 결과를 같이 GPT에 넣는 방식도 고려할 수 있다. Reference가 될 수 있는 데이터를 넣어서, 문제의 변형을 방지하겠다는 취지이다.
   C. 이미지를 Reference로 해서 수식을 Merge하고 Smoothing하려 시도하였으나, 이미지의 번역되지 않는 수식을 그대로 반환하는 문제가 발생했다. 이미지를 단지 Reference로만 참고하라는 Prompt를 말을 바꾸어 가면서 해도 문제는 동일, 조금 더 프롬프트에 대한 연구가 필요해 보인다. OCR한 결과를 같이 넣어주는 방식은 테스트해보지 않았으나, 이미지를 넣는 것보다는 훨씬 좋은 결과를 보여주리라 예상한다.

5. Text To Speech (Optional)

결국 최종목표는 이미지에서 텍스트를 잘 뽑아내어서 음성으로 바꾸는 것이었기 때문에, 현재 우리가 쓸 수 있는 OpenAI TTS모델을 사용해서 진행해보았다. AI가 아무래도 한국인이 아니다 보니 재외동포 같은 억양에 더하여, 보기나 문제 번호등의 띄어서 읽어야 하는 경우에도, 텍스트가 잘 띄워져 있지 않는다면 붙여서 읽는 문제등이 존재했다. 그럼에도 불구하고, 전체적으로 듣기에 불편한 정도는 아니었다고 생각된다. 아마, 더 나은 TTS 모델을 사용하면 완화될 수 있을 것이다.

6. 최종 성능

2000개의 데이터중에서 랜덤으로 200개를 샘플링하여 Processing했을 때의 결과이다.