공학석사학위논문

수학 문제 풀이에 대한 인공지능 모델의 수 개념 이해도에 대한 고찰

A Study on the Understanding of the Number Concept of the AI Model for Math Word Problem Solving

2023년 8월

서울대학교 대학원

지능정보융합 전공

안 지 수

수학 문제 풀이에 대한 인공지능 모델의 수 개념 이해도에 대한 고찰

A Study on the Understanding of the Number Concept of the AI Model for Math Word Problem Solving

지도교수 권 가 진

이 논문을 공학석사 학위논문으로 제출함

2023년 7월

서울대학교 대학원

지능정보융합 전공

안 지 수

안지수의 공학석사 학위 논문을 인준함

2023년 7월

위 원 장: 이 교 구 (인)

부위원장: 권 가 진 (인)

위 원: 곽 노 준 (인)

초 록

문장형 수학 문제 자동 풀이 연구는 1960년부터 지속적으로 연구되어 온 흥미로운 분야[1]이다. AI의 발전함에 따라 문장형 수학 문제를 풀기 위해 AI를 사용하려는 시도가 늘고 있다. 그러나 최근 문장형 수학 문제 풀이 모델이 문제를 이해하고 추론을 통해 문제를 푸는 것이 아닌 문제에 등장하는 숫자를 적절히 조합하여 답을 도출한다는 문제가 제기[2]됨에 따라 모델의 문장형 수학 문제 이해 여부가 불분명해졌다. 수학 문제를 이해하기 위해서는 문제에 등장하지 않는 문제에 등장하는 숫자의 이해는 선행되어야 하므로, 본 논문에서는 수학 문제 풀이 모델이 문제 풀이 과정에서 숫자의 이해를 돕는 두 가지 Study를 수행한다. Study 1은 명시적 자질 추출 방식을 제안한다. 기존 BERT계열의 사전학습 언어모델을 사용한 연구는 수학 문제를 푸는 과정에서 숫자 정보를 제한적인 사용을 하기 때문에 수학 문제에 등장하는 숫자의 대소관계를 파악하기 어려웠다. 이 방식은 숫자 토큰을 문제 풀이에 활용할 수 있도록 하는 방식으로 자연어 이해 계열 모델 중 SVAMP 데이터셋에서 최고 성능을 보이는 deductive reasoner 모델에 적용해 본 결과 최대 2.8%의 성능 향상을 보였다. 위 실험의 결과로 자연어 이해 모델이 수학 문제를 풀 때, 문제에 등장하는 숫자 토큰을 사용하는 것이 대소관계 파악에 도움을 주어 모델의 정답률을 증가시킬 수 있다는 가능성을 확인하였다. Study 2는 GPT계열 사전학습 언어모델의 구현체인 `gpt3.5-turbo`에서 여러 토큰으로 구성된 숫자의 대소관계를 파악할 때, 자릿수 개념을 완벽하게 이해하지 못하고 있다는 문제를 실험을 통해 보여주고 이를 보완하는 전략을 제안한다. 이 실험의 결과로는 기존 `gpt3.5-turbo`를 사용한 프롬프트 전략의 최고 성능 대비 3.1%(in CoT)와 2.06%(in PoT)의 정답률 상승을 보였다. 위 실험의 결과로 자연어 생성 모델이 수학 문제를 풀 때, 숫자의 자릿수 개념을 추가해 준다면 숫자 토큰의 대소관계 파악에 도움을 주어 정답률이 증가할 수 있다는 가능성을 확인하였다.

주요어: Number Understanding, BERT, GPT, ChatGPT, gpt3.5-turbo, Prompt Engineering, Chain of thought

학 번: 2021-27764

차 례

표 차례

그림 차례

제 1 장 서론

1.1 연구의 배경

자연어 처리(Natural Language Processing) 분야는 컴퓨터가 정보가 전달되는 방식인 자연어를 이해하고 활용하기 위해 활발히 연구되고 있다. 인간은 자연어를 마주할 때 문장 자체에 명시되어 있는 정보뿐만 아니라 사회적으로 약속되어 있거나 문장을 보고 유추할 수 있는 내용까지 포함하여 이해한다. 모델이 자연어를 이해하기 위해서도 마찬가지이다. 그렇기 때문에 모델에게 자연어 추론 능력을 학습시킨다는 것은 흥미로운 문제이다[1]. 최근 연구에서는 자연어 추론 능력을 측정하기 위해 문장형 수학 문제 자동 풀이 연구가 진행되고 있다. 표1.1은 문장형 수학 문제의 예시로 문장형 수학 문제는 일상에서 접할 수 있는 상황을 묘사한 문자열이 주어질 때, 주어진 정보를 이용하여 문제에서 요구하는 수식 또는 정답을 도출하는 문제이다. 인공지능 모델이 문장형 수학 문제를 풀기 위해서는 자연어로 기술된 수학 문제를 이해하고, 문제 상황에서 필요한 정보를 추출한 뒤, 추출한 정보를 기반으로 적절한 수식 혹은 정답을 생성하는 등의 과정을 거치게 된다. 이러한 문장형 수학 문제 자동 풀이를 통해 자연어 추론 능력을 평가하고 모델의 성능을 측정할 수 있다[12, 13].

문장형 수학 문제 자동 풀이 연구는 1960년부터 지속적으로 연구되어 온 분야로[14] 규칙 기반 패턴 매칭 방법론[15, 16], 통계 기반 방법론[17, 18, 19, 20]등을 이용해 문제에서 추출된 자질(feature)을 입력으로 넣어 해답을 도출하는 방식으로 연구가 진행되었다. 이후 2017년부터 딥러닝을 이용한 시도가 시작되었는데[21], 재귀 신경망(Recurrent Neural Network), Seq2Seq(Sequence to Sequence) 모델 같은 인코더-디코더 모델[22], 그리고 강화학습 모델[23] 등이 연구되었다. 그러나 이러한 모델로는 문제에는 등장하지 않지만 문제를 풀기 위해 필요한 맥락 지식을 모델이 학습할 수 없다는 문제가 있다. 이러한 문제를 해결하기 위해 트랜스포머(Transformer)기반의 사전학습 언어 모델(Pre-trained Language model)을 활용하여

표 1.1: 문장형 수학 문제의 예시 (SVAMP)

수학 문제	Bryan took a look at his books as well . If Bryan has 56 books in each of his 3 bookshelves , how many books does he have in total ?
수학 문제 (템플릿)	Bryan took a look at his books as well . If Bryan has number0 books in each of his number1 bookshelves , how many books does he have in total ?
정답 수식 (Answer equation)	Multiply(number0, number1) or Multiply(56, 3)
정답	168

문제에서 자질 임베딩을 추출하고 이를 이용하여 문장형 수학 문제 자동 풀이가 진행되고 있다[6, 7, 24, 25]. 사전학습 언어모델을 이용해 문장형 수학 문제 푸는 연구는 크게 자연어 이해와 자연어 생성 계열로 나뉜다. 자연어 이해 계열의 연구는 BERT 같은 사전학습모델에 수학 문제를 통과시켜 자질 임베딩을 얻고, 이렇게 얻은 자질 임베딩을 수식 생성 모델의 입력으로 사용해 정답 수식을 도출한다. 그리고 자연어 생성 계열의 연구는 문장형 수학 문제를 사전학습 생성 모델에 넣어 다음 단어를 예측하는 방식으로 정답을 생성한다. ChatGPT의 등장 이전에는 자연어 이해 모델의 접근을 통해 수학 문제 풀이 모델의 정답률을 개선하려는 연구가 지속되었지만[24, 25], chatGPT의 등장 이후부터 자연어 생성 모델에 프롬프트 엔지니어링을 활용하여 자연어 이해 모델보다 높은 정답률을 얻고 있다[6, 7]. 또한, 최근 프롬프트 엔지니어링을 이용한 시도는 수학 문제 풀이 모델의 강건함(robustness)과 추론 능력을 측정하기 위해 제안된 데이터셋인 SVAMP[2]에서 자연어 이해기반의 최고성능 모델[7]보다 40% 높은 정답률[9, 10]을 얻을 정도로 큰 성능향상을 이끌어냈다.

1.2 연구의 내용

본 논문에서는 수학 문제 풀이 모델이 문제 풀이 과정에서 숫자의 이해를 돕는 두 가지 Study를 수행한다. 첫 번째 연구는 자연어 이해 모델에서 숫자의 대소관계를 파악하기 위해 숫자 토큰을 사용하도록 하는 연구이고, 두 번째 연구는 자연어 생성 모델에서 숫자의 대소관계 이해를 수행할 수 있는지를 확인하고, 대소관계를 이해하기 위해 자릿수를 파악할 수 있도록 하는 연구이다.

1.2.1 Study 1: 자연어 이해 모델에서 숫자의 이해

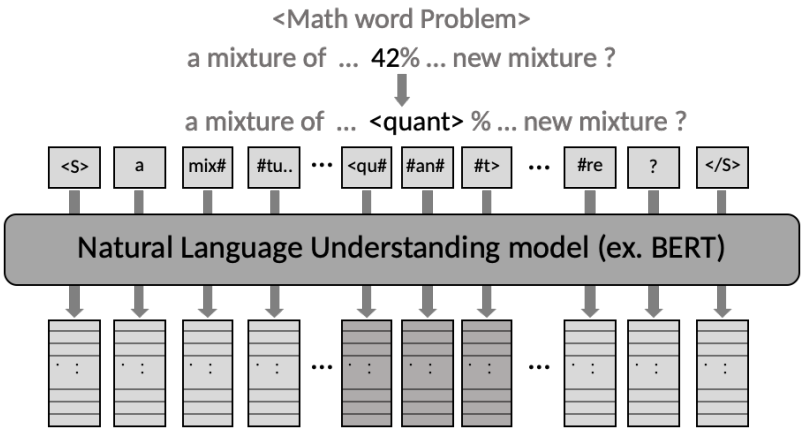

그림 1.1: Deductive Reasoner 모델에서 숫자를 문자로 치환한 뒤 자질 임베딩을 추출하는 과정

자연어 이해기반의 사전학습 언어 모델을 사용하는 선행 연구에서는, BERT 계열의 언어 모델을 인코더로 사용하여 자질 임베딩을 추출하고 추출된 자질 임베딩을 디코더 모델에 입력으로 넣어 수학 문제의 답을 도출한다. 선행연구인 deductive reasoner에서는 그림 1.1과 같이 인코더 수학 문제를 입력으로 넣기 전 전처리 과정에서 숫자 토큰을 그대로 사용하는 것이 아니라, 문제에 등장하는 숫자를 "$< quant >$" 같이 토크나이징 했을 때 구분될 수 있는 문자열로 치환한 뒤 자질 임베딩을 추출

한다[7]. 이러한 방식으로 모델에 치환된 문제를 넣는 이유는 토크나이징 과정에서 숫자와 주변에 있는 문자가 합쳐져서 하나의 토큰으로 인식될 수 있는 문제를 예방하기 위해서이다. 그러나, 숫자 대신 치환된 문자를 넣게 된다면 모델이 문제를 통해 답을 도출할 때, 숫자의 대소관계를 파악하지 못한 채 문제를 풀게 된다. 선행 연구에서 사전학습 언어 모델을 사용하였을 때, 뺄셈과 나눗셈과 같은 피연산자간의 대소관계 혹은 의존관계가 중요한 수식 생성에서 사전학습 언어 모델을 사용하지 않았을 때 보다 높은 정답률을 보였기 때문에[2] 모델이 숫자의 대소관계를 인지할 수 있도록 치환되지 않은 숫자 토큰을 넣어줄 필요성이 있다. Study 1 에서는 숫자를 토크나이징 할 때 "$< quant >$"와 같이 구분가능한 문자열로 치환하지 않고 입력으로 넣어준다면 모델이 문제를 통해 답을 도출할 때 수의 대소관계를 파악하여 정답률이 향상될 수 있는지를 확인한다.

1.2.2 Study 2: 자연어 생성 모델에서 숫자의 이해

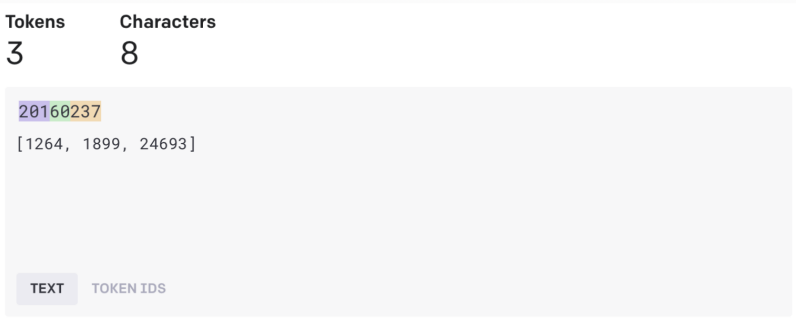

그림 1.2: GPT 모델에서 숫자를 토큰으로 인식하는 방식

자연어 생성 기반인 GPT계열의 언어 모델을 사용하는 연구에서는 end-to-end 방식으로 문제를 해결해 왔다. End-to-end 방식은 모델을 여러 개로 분리하지 않고 하나의 모델로 문제를 해결하는 방식으로 선행연구에서는 사전학습 언어모델의 출력값을 그대로 정답으로 사용하여 문제의 답을 도출한다. 즉, 하나의 모델로 문제를 해결하기 때문에 문제 자체를 입력으로 넣고 정답 또는 수식을 출력으로 얻는 것이

다. 따라서 숫자를 문자로 치환해서 문제를 푸는 자연어 이해 모델과 달리 자연어 생성 모델은 숫자 그대로를 넣어 문제의 답을 도출한다. 그러나 숫자를 그대로 넣는다고 해도 자연어를 모델의 입력으로 넣을 때 토크나이징 과정은 필수적이다. 자연어가 토큰으로 쪼개져 모델의 입력으로 들어갈 때, 문제에 등장하는 숫자는 그림 1.2처럼 ”20160237”이라는 하나의 숫자가 ”201”, ”60”, ”237”이라는 토큰으로 분리된다. 숫자의 값을 이해하기 위해서는 하나의 숫자를 이루고 있는 각 자릿수와 그 자릿수에 해당하는 값을 알고 있어야 하지만, 자연어 생성 모델에서는 숫자가 특정 규칙 없이 여러 개의 토큰으로 분리되어 모델의 입력으로 들어가기 때문에 자릿수 개념에 대한 손상이 발생할 가능성이 있다. 따라서 Study 2에서는 여러 토큰으로 구성된 숫자가 모델의 입력으로 들어갈 때, 토큰들 사이의 자릿수 개념을 보존하고 있는지를 확인하고 이를 개선할 수 있는지에 대한 실험을 수행한다. 먼저 자릿수의 개념을 이해하고 있는지를 확인하기 위해 여러 토큰으로 이루어진 수들 사이의 대소관계 파악을 할 수 있는지 확인하고, 프롬프트 엔지니어링을 통해 자릿수 개념을 추가하였을 때 성능향상이 이루어질 수 있는지 확인한다.

위 두 실험을 통해 얻을 수 있는 본 논문의 기여점은 다음과 같다

1. 자연어 이해 기반 모델에서 숫자 토큰의 사용은 수학 문제 풀이 모델의 문장형 수학 문제에 등장하는 숫자 토큰의 대소관계 파악에 도움을 주어 모델의 정답률을 증가시킬 수 있다는 가능성을 확인한다.

2. 자연어 생성 기반 모델에서 숫자의 자릿수 개념을 추가하여 숫자 토큰의 대소관계 파악에 도움을 주어 모델의 정답률을 증가시킬 수 있다는 가능성을 확인한다.

제 2 장 관련 연구

본 연구의 목표는 인공지능 모델의 수학 문제를 풀 때 수 개념을 이해하고 있는지를 확인하는 것이다. 최근 수학 문제를 풀기 위한 시도는 두 가지 분야인 (1) 자연어 이해 모델을 사용한 수학 문제의 풀이 (2) 자연어 생성 모델을 사용한 수학 문제 풀이로 나뉜다. 첫 번째 분야인 자연어 이해 모델을 사용한 수학 문제 풀이는 문제로부터 자질을 추출하고 추출된 자질을 사용하여 수식 또는 답을 도출하여 문제를 해결한다. 두 번째 분야인 자연어 생성 모델을 사용한 수학 문제 풀이는 수학 문제의 원형을 모델의 입력으로 넣어 문제를 풀게 하거나, 문제에 프롬프트를 추가하여 모델에 입력으로 넣어 수식 또는 답을 도출하도록 한다. 우리는 이 두 가지 분야에서 모델의 활용 방법을 확인하면서 문제를 어떤 방식으로 해결하고 있는지 알 수 있다.

2.1 트랜스포머 구조를 활용한 사전학습 언어 모델

2.1.1 트랜스퍼 러닝

트랜스퍼 러닝(transfer learning)은 이전에 특정 작업을 학습하기 위해 수행되었던 모델을 새로운 작업에 다시 사용하는 기법을 의미한다[26]. 인간 학습자는 새로운 과제(task)가 이전의 경험과 관련이 있을 때, 이전의 경험으로부터 학습했던 지식을 바탕으로 새로운 과제를 학습해 나간다. 하지만 머신러닝 알고리즘의 경우 각각의 과제를 독립적으로 인식하여 문제를 해결해 왔다. 트랜스퍼 러닝은 하나 이상의 기존 과제(source task)로부터 학습된 지식(knowledge)을 사용하여 새로운 목표 과제(target task)를 수행하는 테크닉으로 기존에 학습된 모델에 추가적인 학습을 하도록 하여 이전에 학습된 지식을 전이할 수 있도록 한다[26]. 기존 과제는 통상적으로 업스트림 테스크(upstream task) 그리고 새로운 목표 과제를 다운스트림 테스크(downstream task)라고 부르고, 다운스트림 테스크를 위해 업스트림 테스크만을 학

습하여 활용되는 모델을 사전 학습 모델(pretrained model)이라고 부른다.

2.1.2 트랜스포머의 구조

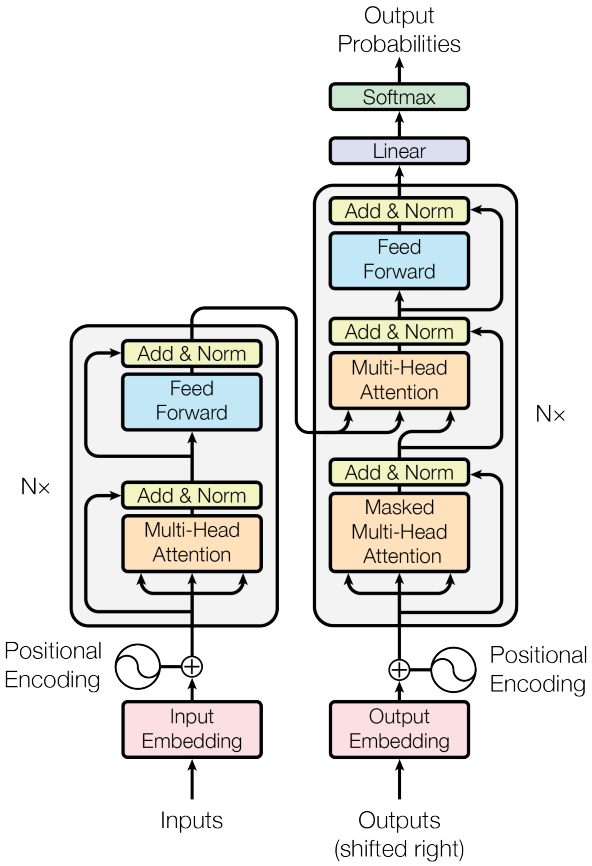

그림 2.1: 트랜스포머 아키텍처 [3]

트랜스포머(transformer)는 인코더-디코더(encoder-decoder) 구조로 이루어진 멀티 헤드 어텐션(multi-head attention)을 사용하는 아키텍처이다[3]. 그림 2.1과 같이 트렌스포머는 문장을 잠재 표현(hidden representation)으로 변환하는 인코더 블록(encoder block)과 이를 다시 복원하는 디코더 블록(decoder block)으로 구성된다. 인코더는 N개의 인코더 블록으로 구성되고 인코더 블록은 두 개의 서브 레이어로

구성된다. 첫 번째 레이어는 멀티 헤드 어텐션 레이어이고, 두 번째 레이어는 간단한 순방향 신경망(feed-forward network)이다. 또한 두 개의 서브 레이어는 잔차연결(residual connection)로 연결된다. 디코더의 경우, N개의 디코더 블록으로 구성되고 디코더 블록은 인코더 블록과 같은 두 개의 서브레이어에 마스크 멀티 헤드 어텐션(masked multi-head attention) 레이어가 추가된 구조로 이루어진다. 이때 사용되는 마스크 멀티 헤드 어텐션은 지금까지 도출된 결과 이후의 정답은 디코더의 입력으로 보지 못하게 하는 기법이다. 이 기법은 기존 Seq2Seq 모델에서 순차적으로 정답이 생성되어 병렬화할 수 없었던 것을 개선하여, 트랜스포머 아키텍처는 학습을 병렬적으로 수행할 수 있도록 설계하기 위해 도입되었다.

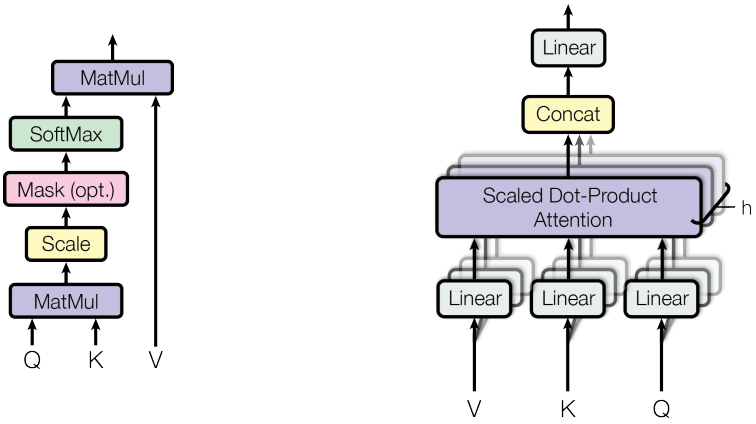

그림 2.2: (왼쪽) 스케일드 닷-프로덕트 어텐션. (오른쪽) 멀티 헤드 어텐션 [3]

트랜스포머에서 사용하는 어텐션은 query와 key-value 임베딩을 입력으로 받아 query와 key의 연관성을 구하여 value의 가중합을 수행하는 기법으로 스케일드 닷-프로덕트 어텐션(Scaled dot product attention)를 사용한다. 그림 2.2과 같이 스케일드 닷-프로덕트 어텐션은 key와 query 임베딩을 곱하여 가중치를 계산하고, 그

가중치만큼 value에 곱해 값을 계산하게 된다.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

멀티 헤드 어텐션은 위의 식과 같이 헤드별로 weight matrix인 W_i^Q, W_i^K, W_i^V를 각 입력에 곱하여 query, key, value를 생성하고 이를 스케일드 닷-프로덕트 어텐션을 수행하여 어텐션 값을 구한다. 이 과정에서 head 별로 다른 weight matrix가 학습되기 때문에 헤드별로 다른 관점에서 어텐션을 계산하고 있다고 해석할 수 있다. 이는 하나의 레이어에서 입력을 여러 방향으로 해석한다는 관점에서 CNN의 컨볼루션 레이어(convolution layer)에서 kernel을 여러 층으로 두는 것과 유사하다. 이렇게 구성된 트랜스포머 모델을 사용하여 다양한 시계열 테스크를 수행했을 때, RNN 계열의 모델을 사용하지 않고 어텐션 기법만 적용하더라도 높은 성능을 보임을 확인할 수 있었다.

2.1.3 자연어 이해 모델

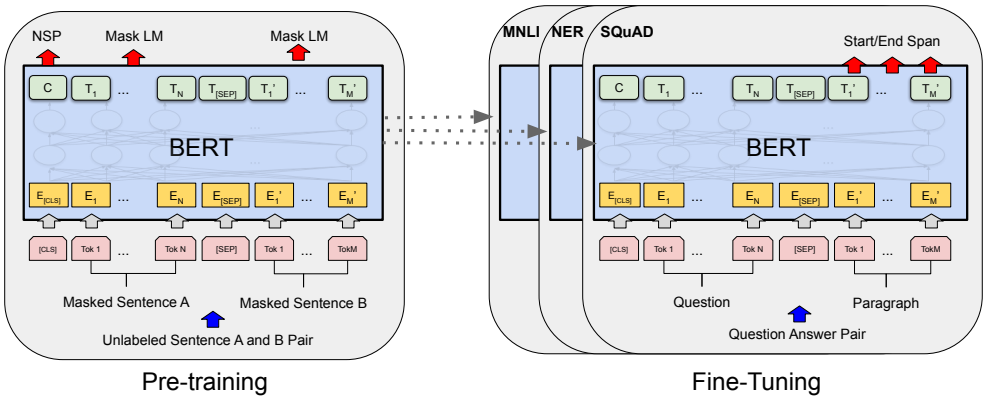

그림 2.3: BERT의 사전학습(pre-training)과 미세조정(fine-tuning)의 절차 [4]

자연어 이해 모델은 자연어의 표현(representation)을 학습하는 모델로 대표적인 자연어 이해 모델로는 Bidirectional Encoder Representations for Transformer(BERT)가 있다. BERT는 트랜스포머의 인코더를 분리한 형태로 Masked Language Model(MLM)을 사용하여 양방향 학습을 가능하게 한 모델이다. MLM은 입력 토큰을 무작위로 마스킹하고, 해당 맥락에서 마스킹한 단어를 예측하는 작업으로 다음 단어를 예측하여 단방향으로 학습하는 방식과 달리 양방향으로 학습할 수 있게 한다. 또한 다음에 등장하는 문장을 예측하는 작업도 수행하도록 하여 사전학습(pre-training)을 수행한다. 그림 2.3과 같이 사전학습된 언어모델은 미세조정의 과정을 거쳐 문장 바꾸기(paraphrasing), 다중 장르 추론(multi-genre natural language inference), 개체명 인식(named entity recognition), 질의응답(question answering) 등 다양한 다운스트림 테스크에서 활용되며 사전학습 언어모델을 사용하지 않을 때 보다 높은 성능을 내고 있다[4]. 또한 최근에는 BERT가 덜 훈련되었다고 주장하여 추가적인 학습을 수행한 Robustly Optimized BERT Pretraining Approach(RoBERTa)[27], 어텐션 가중치(attention weight)를 서로 독립인 내용 벡터와 위치 벡터의 조합으로 표현하는 Disentangled Attention Mechanism을 적용한 Decoding-enhanced BERT with Disentangled Attention(DeBERTa)[28] 등이 등장했다.

2.1.4 자연어 생성 모델

자연어 생성 모델은 자연어 토큰을 생성하는 모델로 대표적인 자연어 생성 모델로는 Generative pre-trained transformer(GPT)가 있다. GPT는 그림 2.4의 왼쪽 그림과 같이 트랜스포머의 디코더를 분리한 뒤 인코더와 디코더가 연결되는 멀티 헤드 어텐션 레이어를 제거한 형태로 K개의 단어를 보고 다음 단어를 예측하는 형태로 학습이 이루어진다. 이렇게 사전 학습된 모델은 목표 데이터셋의 문제에 대해서 정답을 맞히도록 학습된다. 그림 2.4의 오른쪽은 다운스트림 테스크의 예시로 분류(classification), 전제(promise)에 대한 가설(hypothesis)의 참(entailment) 거짓을 판단, 두 문장의 유사성(Similarity) 판단, 질의응답(question answering) 등 12개의 테스크를 수행하여 9개의 테스크에서 최고 성능을 보였다. 또한 최근에는 GPT의 파라

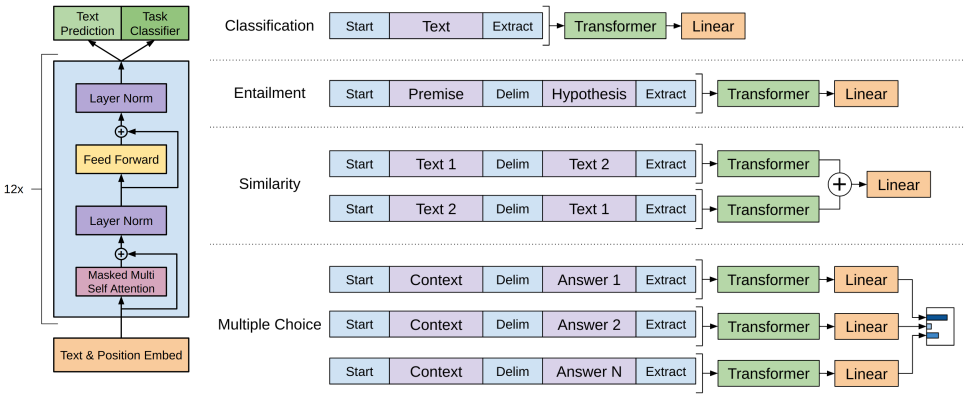

그림 2.4: (왼쪽) GPT의 모델 아키텍처. (오른쪽) GPT의 다운스트림 테스크 [5]

미터 사이즈를 키워 GPT-2[29], GPT-3[30], GPT-4[31] 등으로 발전해 나가고 있다.

2.2 문장형 수학 문제 자동 풀이

문장형 수학 문제 자동 풀이는 1960년대부터[14] 시작하여 지금까지 연구되고 있는 분야이다. 규칙 기반 패턴 매칭 알고리즘[15, 16]에서 통계 기반 방법론[17, 18, 19, 20], 트리 기반 방법론[23, 24, 25, 32, 33, 34] 그리고 최근에는 딥러닝 기반 방법론[24, 25, 9, 10]으로 이어지고 있다. 문장형 수학 문제 자동 풀이를 하기 위해서는 문제 자체에 명시되어 있는 정보뿐 아니라 사회적으로 약속되어 있거나 문장을 보고 유추할 수 있는 내용까지 포함하여 문제를 이해해야 한다. 세계 지식(global knowledge)을 고려하여 문제를 풀기 위해 최근 인공지능 수학 문제 풀이 모델은 트랜스포머 구조를 활용한 사전학습 언어모델을 사용하여 연구를 진행하고 있다[6, 7, 24, 25, 9, 10].

2.2.1 자연어 이해 모델을 사용한 문장형 수학 문제 풀이

세계 지식을 고려하여 수학 문제를 이해하기 위해, 연구자들은 BERT같은 사전학습된 자연어 이해 모델을 사용하여 수학 문제의 자질 임베딩을 추출하고 추출

된 자질을 이용하여 수학 문제의 수식이나 정답을 도출해 왔다[24, 25, 6, 7, 9, 10]. 그리고 최근에는 자연어 이해 모델 중 ELASTIC 모델은 MathQA 데이터 셋에서 Deductive Reasoner는 SVAMP 데이터 셋에서 최고 수준의 결과(State-of-the-Art)를 보이고 있다.

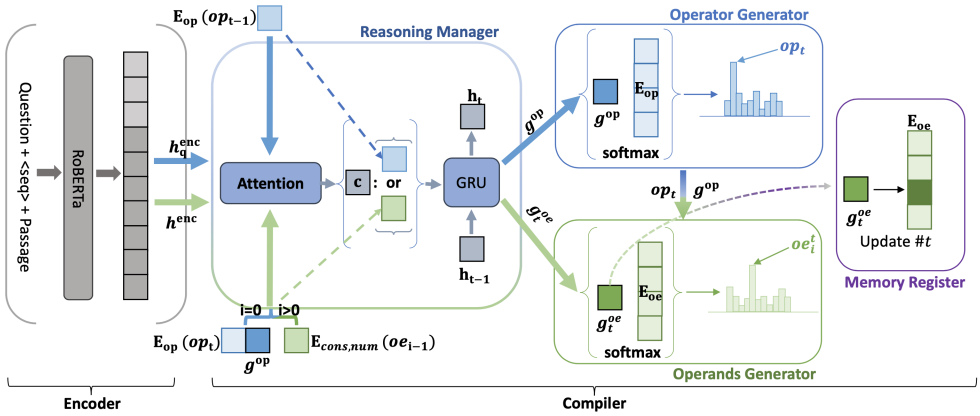

그림 2.5: ELASTIC 모델 아키텍처 [6]

ELASTIC 모델 현재 MathQA 데이터 셋에서 가장 높은 성능을 내고 있는 ELASTIC 모델이다[6]. ELASTIC 모델은 그림 2.5과 같이 인코더 부분과 컴파일러 부분으로 나뉜다. 인코더 부분에서는 사전학습 자연어 이해 모델에 문제를 통과시켜 나온 자질 임베딩을 추출한다. 컴파일러 부분은 4개의 모듈로 이루어지게 되는데 (1) 추론 관리자(reasoning manager) (2) 연산자 생성기(operator generator) (3) 피연산자 생성기(operand generator) (4) 메모리 레지스터(memory register)이다. 먼저 하나의 수학 문제를 풀기 위해서는 여러 단계의 식을 생성해야 하는데, 추론 관리자는 임베딩에서 얻은 자질 임베딩을 이용하여 현재 단계 수식의 연산자를 생성하기 위한 가이드 임베딩 g^{op}을 만든다. 그리고 피연산자 생성기를 통과시켜 피연산자 임베딩 oe_1^t을 얻게 된다. 하나의 연산자에 해당하는 피연산자는 여러 개이기 때문에 다음 피연산자 임베딩은 이번에 얻은 피연산자 임베딩을 사용하여 얻게 된다. 이 과정을

통해 피연산자가 모두 결정되고 나면 마지막으로 얻은 피연산자 임베딩을 메모리 레지스터에 저장하고 다음 단계 수식을 계산한다. 메모리 레지스터는 이전 수식의 정답을 저장함으로써 이전 단계의 결과를 다음 수식의 피연산자로 활용할 수 있도록 한다. 이 과정을 반복하면 문제에 해당하는 정답 수식을 얻을 수 있다. 한편, ELASTIC 모델에서 문제에 해당하는 숫자에 대한 자질 임베딩을 추출할 때 숫자 토큰을 전부 사용하지 않고 첫 번째 토큰만을 사용하여 숫자 임베딩을 추출하는 제한적인 방식을 사용하고 있다. 첫 번째 토큰만을 사용하는 이유는 추출된 자질을 사용하는 과정에서 차원을 맞추기 위해서라고 해석할 수 있다.

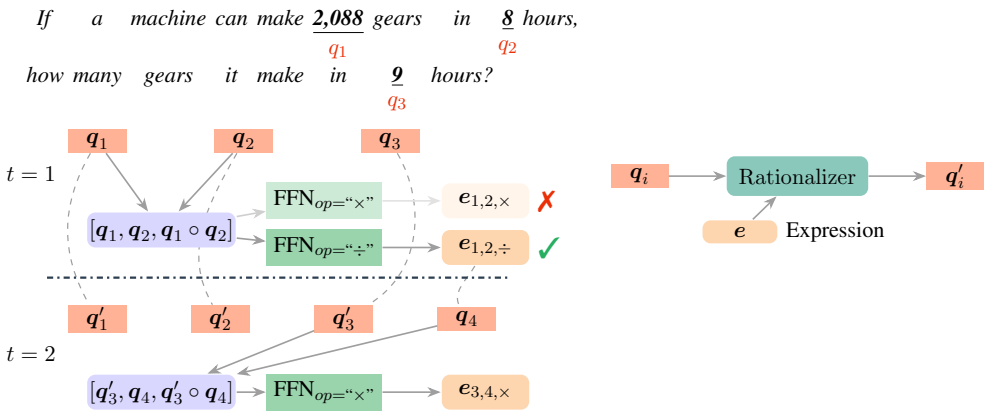

그림 2.6: (왼쪽) Deductive reasoner의 동작 과정. (오른쪽) Quantity의 Rationalizing[7]

Deductive Reasoner 모델 SVAMP 데이터셋을 푸는 사전학습 자연어 이해 계열 모델 중 가장 높은 성능을 보이는 모델은 Deductive Reasoner이다. Deductive reasoner는 그림 2.6 과같이 동작하며 (1) Reasoner와 (2) Rationalizer로 이루어진다. Reasoner의 목적은 가능한 피연산자에 대한 모든 순서쌍에 대한 표현 임베딩(representation)을 얻는 것으로, 문제에 등장하는 세지 않지만, 문제를 풀 때 활용될 수 있는 숫자에 대해 모든 가능한 순서쌍을 구하고 문제를 풀 때 사용되는 모든 연산자에 대하여 임베딩 벡터를 구한다. 이후 임베딩 벡터에 대해 스코어를 계산하여 가장 높은 스

코어의 순서쌍을 이번 step의 수식으로 결정한다. Rationalizer는 이번 step의 수식이 결정되었을 때 기존의 수량 표현을 업데이트하는 기능을 수행하며, 피연산자로 가능한 모든 임베딩을 Rationalizer를 통과시켜 업데이트를 수행한다. Deductive reasoner는 위 두 과정을 반복하여 정답 수식을 완성한다. 또한 수식의 생성을 멈출 수 있도록 Reasoning 단계에서 임베딩 벡터에 대한 스코어를 계산할 때, 각 벡터에 대한 종료 스코어도 계산토록 하여 종료 스코어가 선택되었을 때 수식의 생성을 멈춘다[7]. 한편, Deductive Reasoner는 문제에 등장하는 숫자로부터 임베딩을 추출할 때 숫자를 $< quant >$로 바꾼 뒤 임베딩을 추출하는 방식을 사용하고 있다. 이러한 방식으로 모델에 치환된 문제를 넣는 이유는 토크나이징 과정에서 숫자와 주변에 있는 문자가 합쳐져서 하나의 토큰으로 인식될 수 있는 문제를 예방하기 위해서이다.

2.2.2 자연어 생성 모델을 사용한 문장형 수학 문제 풀이

사전학습의 발전과 instructGPT의 도입으로 연구자가 사전학습언어 모델에 직접적인 명령을 내리게 되면서[8] 자연어 생성 모델들은 산술(arithmetic)[35], 상식 이해(commonsense) [36, 37] 그리고 상징적 추론(symbolic reasoning)[38]등의 분야에서 추론 능력을 보이게 되었다. 또한 최근에는 Chain of thought나 Program of thought와 같이 프롬프트 기법을 통해 문장형 수학 문제를 푸는 시도에서 높은 정답률을 보이고 있다.

InstructGPT InstructGPT는 사람들의 의도에 더 잘 부합하는 언어 모델을 훈련하는 방법으로 Reinforcement Learning from Human Feedback(RLHF)이라는 방식을 사용한다. 그림 2.7는 instructGPT를 학습시키는 단계를 나타낸다. 첫 번째 단계로 라벨러가 작성한 프롬프트를 통한 supervised learning으로 GPT모델을 미세 조정한다. 두 번째 단계로 GPT모델로부터 여러 개의 모델 출력을 받아 라벨러가 순위를 매기도록 하고, 순위를 매기는 방식을 보상모델이 학습하도록 한다. 이후 세 번째 단계에서는 학습된 보상모델을 사용하여 PPO방식으로 GPT 모델을 미세조정한다. 이렇게 생성된 모델이 InstructGPT로 사용자의 의도와 일치한 답변을 한다는 점이

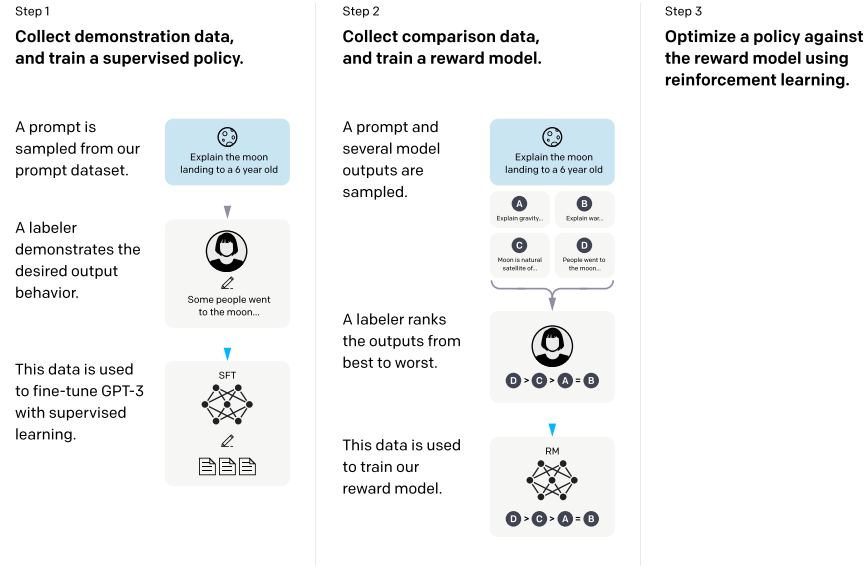

그림 2.7: InstructGPT의 세 가지 단계 (1) 지도학습 미세조정(supervised fine-tuning), (2) 보상 모델(reward model) 훈련, 그리고 (3) 근접 정책 최적화(proximal policy optimization)를 사용한 강화학습(reinforcement learning)[8]

GPT-3 모델보다 개선되었다[8]. 또한 이러한 방식으로 학습된 instructGPT는 GPT-3보다 100배 작은 모델 파라미터로 GPT-3보다 선호되는 답변을 생성한다. 이를 통해 단순히 모델 사이즈를 늘리는 방법이 아닌 RLHF를 사용한 학습으로 인간이 직접 모델에 명령을 내리고 원하는 답변을 얻을 수 있게 된 것이다.

Chain of thought Chain of thought는 자연어 생성 모델이 추론의 과정에서 중간 단계를 생성함으로써 추론 과정을 자연스럽게 나타나게 하는 기법을 의미한다. 그림 2.8은 chain of thought를 사용한 프롬프팅방식을 나타내는데, chain of thought는 답만을 도출하게 하는 기존의 방식과 달리 예시를 함께 보여줘서 질문을 하는 few-shot learning 기법을 사용하여 중간 과정도 생성하도록 한다. 또한 중간 단계인 "자연어적 근거(natural language rationales)"를 생성함으로써 어디에서 추론 경로가 잘못되었는지 확인할 수 있어 모델을 디버깅 할 수 있는 기회도 얻을 수 있다. 그리고

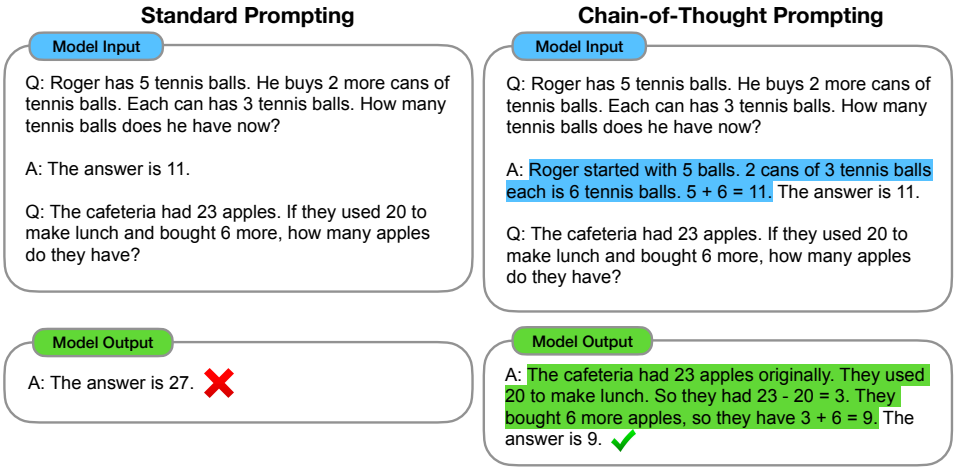

그림 2.8: (왼쪽) 기존의 프롬프트 방식. (오른쪽) Chain of thought를 적용한 프롬프팅 방식[9]

chain of thought 방식은 산술, 상식 이해, 상징적 추론 등의 과제를 수행할 때 뛰어난 성능을 보이고 있다. 이 결과로, 프롬프트를 통한 접근이 언어 모델의 추론 능력을 향상시킬 수 있다는 것을 확인했다[9].

Program of thought Program of thought는 추론의 과정을 프로그래밍 언어 형태로 표현하는 방법을 제안하는 기법을 의미한다. 그림 2.9은 Chain of thought와 program of thought를 비교하는 그림으로 program of thought는 정답을 글로 생성하는 것이 아닌 파이썬 코드로 생성하여 실제 연산은 파이썬 인터프리터가 수행하도록 한다. 이 방식의 핵심은 계산과 추론을 분리하는 것으로, 프로그램은 추론의 과정을 설명하고, 실제 계산은 프로그램 인터프리터가 처리한다. 이렇게 함으로써 자연어 생성 모델은 복잡한 문제를 이해하고 추론 과정을 설계하는 데 집중할 수 있다. 이 방법을 통해 PoT는 수학 문제를 포함한 다양한 데이터셋에서 기존의 성능을 크게 향상시켰다[10].

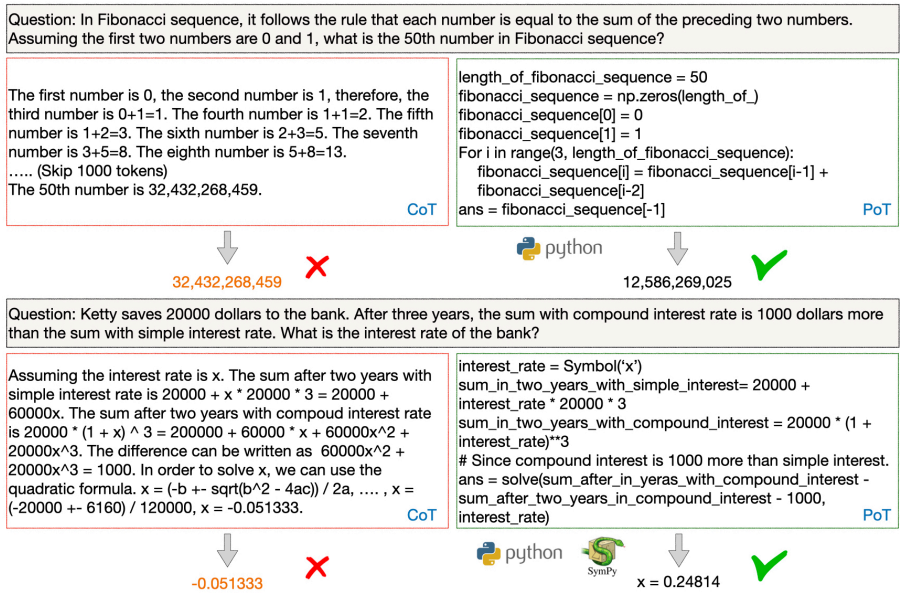

그림 2.9: Chain of Thoughts와 Program of Thoughts의 비교[10]

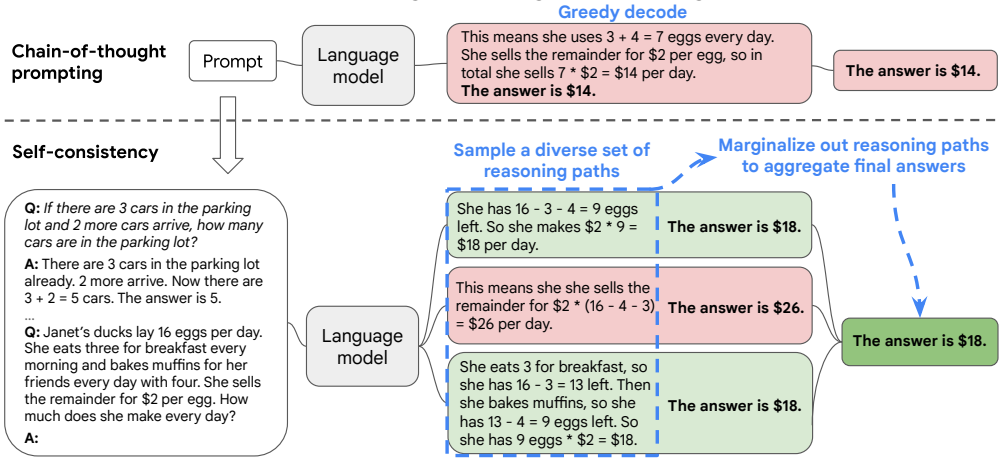

그림 2.10: chain of thought의 정답 추론 과정과 self-consistency를 사용한 디코딩 방식[11]

Self-Consistency Self-consistency는 자연어 생성 모델의 디코딩 전략 중 하나이다. 그림 2.10은 chain thought의 추론 과정과 디코딩 전략으로 self-consistency를 선택했을 때 발생하는 추론의 과정을 나타낸다. self-consistency 방식은 세 가지 과

정으로 이루어진다 (1) chain of thought 프롬프팅을 사용하여 답을 도출한다. (2) CoT의 greedy 디코딩 과정을 다양한 추론 경로를 샘플링하도록 변경한다 (2) 가장 많이 등장한 추론 경로를 마진화(marginalize) 하여 가장 일관성 있는 답을 선택한다. 이 방법은 복잡한 추론 문제에서 여러 가지 다른 사고방식이 그 문제의 유일한 답으로 이어질 수 있다는 직관을 기반에 두고 있는 방법으로 추가적인 데이터 수집의 필요나 추가적이니 훈련 없이 바로 동작하기 때문에 비감독적인(unsupervised) 방식이라고 할 수 있다. 또한 chain of thought에 self consistency를 적용했을 때 프롬프팅의 성능이 크게 향상되는 것으로 보아 self-consistency가 모델의 추론 성능을 향상시키는 데 중요한 역할을 할 수 있음을 보였다[11].

제 3 장 연구 방법

본 논문은 그림 3.1와 같이 두 개의 study로 구성된다. 첫 번째 study는 자연어이해 모델에 수학 문제에 등장하는 숫자를 그대로 토크나이징 하여 입력으로 넣었을 때 수학 문제 풀이에 도움이 되는지 확인하는 실험이고, 두 번째 study는 자연어 생성 모델에서 숫자 토큰이 주어질 때 자릿수의 개념을 완전히 이해하는지 확인하고, 자릿수의 개념을 추가해 주었을 때 수학 문제 풀이에 도움이 되는지 확인하는 실험이다.

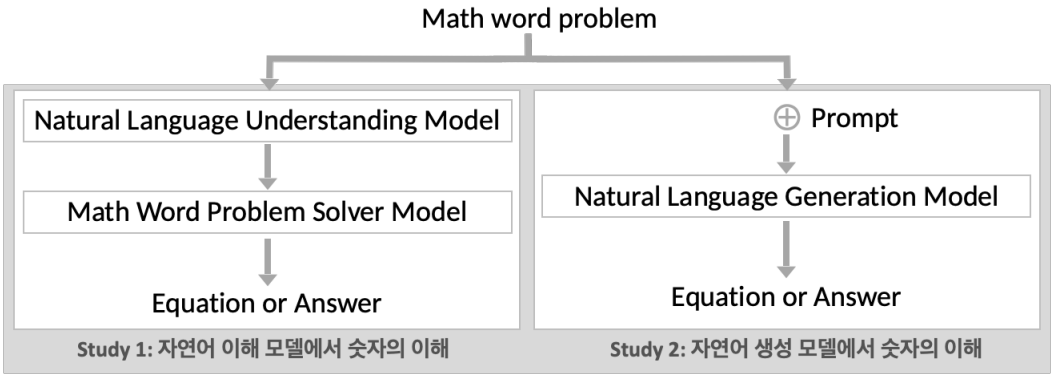

그림 3.1: 두 가지 Study의 overview

3.1 실험 환경

실험이 진행되는 환경은 표 3.1와 같다. 자연어이해 모델실험에 사용되는 모델은 직접 구현하여 사용하였으며, 자연어 생성 모델실험에 사용되는 모델은 GPT3.5 기반 구현체인 `gpt3.5-turbo` API를 사용하였다. 이 API는 OpenAI에서 제공되며 chatGPT서비스에서 동작하고 있는 모델과 동일한 성능을 가진다. 가격은 1000 토큰당 0.002달러이다.

Processor	AMD Ryzen Threadripper 3980X CPU
Memory	192GB DRAM
Gpus	GeForce RTX 3090 * 4
Framework	PyTorch 2.0.1 & Pytorch lightning 2.0.1 &Sympy
OpenAI API	`gpt3.5-turbo`
Python	python 3.9
OS	Ubuntu 20.04.5 LTS

표 3.1: 실험 환경

3.1.1 실험 데이터셋 및 구현 세부 사항

	Mawps	**ASDiv-A**	**SVAMP**
problems	2,373	1,218	1,000

표 3.2: 데이터셋 별 문항 수

표 3.2는 수학 문제 데이터 셋의 문항 수를 나타낸다. 본 논문에서는 문장형 수학 문제 자동 풀이 모델의 성능을 평가하기 위해 Simple Variations on Arithmetic Math word Problems(SVAMP) 데이터 셋을 사용한다. SVAMP는 모델의 강건함을 테스트하기 위한 데이터셋으로 MAWPS[39]와 ASDiv-a[40] 데이터 셋에서 몇 개의 문제를 뽑아 9가지 방식의 변형 기법으로 변형하여 1,000개의 샘플을 만들어 놓은 데이터셋이다. 이때 사용한 변형 기법은 문장의 순서 바꾸기, 불필요한 숫자를 문제에 추가하기, 질문의 대상 바꾸기 등 문제의 난이도는 유지되는 상태에서 새로운 문제를 풀게 하여 모델의 추론 능력을 측정한다. 따라서 이 데이터셋에 대한 성능을 확인할 때는 MAWPS와 ASDiv-a 데이터 셋의 모음을 학습 집합으로 사용하고, SVAMP를 테스트 집합으로 사용한다[2].

NLU기반 모델을 실험하는 Study 1에서 자연어 이해 모델은 MAWPS와 ASDiV-a로 학습하고 SVAMP로 검증하지만, NLG기반 모델을 실험하는 Study 2에서 자연어 생성 모델은 추가 학습 없이 프롬프트 엔지니어링을 사용하여 SVAMP에 대해 검증한다. 평가 기준의 경우, 선행연구와 같이 Study 1과 Study 2 모두에서 정답을 정확하게 생성하는지 확인한다.

추가로, Study 2를 위해 CoT와 PoT 기법을 변형해서 사용했다. 본 실험에서 사용된 프롬프트는 CoT에서 가져온 8개의 예시와 PoT에서 가져온 7개의 예시를 활용했다[9, 10]. 또한 선행연구와 같이 그리디 디코딩을 사용하는 실험에서는 정답에 다양성을 부여하는 척도인 temparature를 0으로 설정했으며, self-consistency 방식을 사용하는 실험에서는 temparature를 0.7로 설정했다.

3.2 Study 1: 자연어 이해 모델에서 숫자의 이해

3.2.1 명시적 자질 추출 방식의 제안과 활용 방법

그림 3.2는 문제에 해당하는 숫자에 대한 자질을 명시적으로 추출하는 과정이다. 문제에 등장하는 숫자의 자질 임베딩을 추출하기 위해 문장형 수학 문제는 사전학습 언어 모델의 토크나이저를 통해 각각의 토큰으로 분할되고, 분할된 토큰을 사전학습 언어 모델을 통과시켜 각 토큰의 임베딩 벡터를 얻게 된다. 이때 토크나이저가 생성하는 토큰들은 띄어쓰기 단위가 아닌 바이트 페어 인코딩(BPE) 같이 언어모델을 학습시킬 때 같이 등장했던 바이트들의 묶음 중 높은 빈도로 등장한 토큰이 되기 때문에 숫자와 문자가 겹쳐서 하나의 토큰이 될 수 있다는 문제가 있다[25]. 선행연구에서는 이 문제를 해결하기 위해 숫자와 문자가 겹치는 문제에서 숫자와 문자를 구분하기 위해 1번 단계와 같이 토큰화가 되었을 때 토큰이 유일한 형태로 나타나는 특수한 문자열로 숫자를 대체하여 토큰화한다. 명시적 자질 추출방식은 2번 단계와 같이 치환된 토큰자리에, 문제에 등장하는 숫자를 토크나이저에 따로 통과시켜 대체한다. 그렇게 얻은 문장 전체에 대한 토큰들을 사전학습모델에 통과시키면 숫자의 대소관계를 이해할 가능성이 있는 임베딩 벡터를 얻을 수 있다. 이때

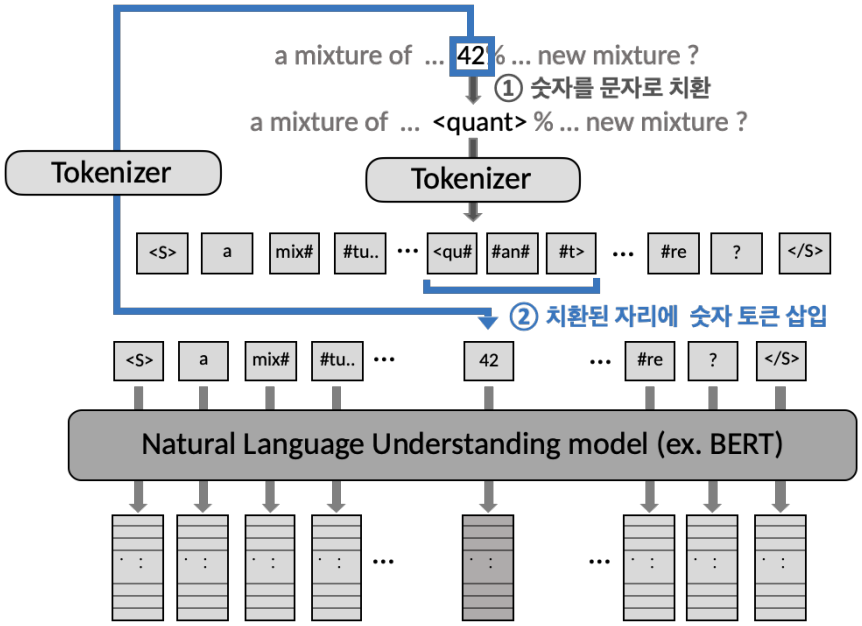

그림 3.2: 자연어이해 모델의 명시적 자질 추출 방식

만약 숫자가 여러 개의 토큰으로 토크나이징되면 숫자 토큰의 첫 번째 토큰의 임베딩과 마지막 토큰의 임베딩을 이어 붙이고(concatenate) 이를 선형 레이어같은 투사(projection) 레이어에 통과시켜 임베딩을 얻는다.

수학 문제 풀이 모델에서 자연어이해 모델을 통해 추출할 수 있는 자질은 피연산자와 연산자에 해당하는 자질 임베딩이다. 피연산자에 해당하는 자질 임베딩은 문제에 등장하는 숫자와 문제에는 등장하지 않지만, 문제를 풀기 위해 필요한 상수(ex. 3.14(원주율), 24(시간), 60(분. 초))에 대한 자질 임베딩이고, 연산자에 해당하는 자질 임베딩은 문제를 풀기 위해 필요한 연산(addition(덧셈), subtraction(뺄셈), multiplication(곱셈), division(나눗셈) 등)이 있다.

그림 3.3는 피연산자에 해당하는 후보군 임베딩을 초기화하는 과정이다. 우리는 그림 3.2와 같은 단계로 명시적 자질 추출 방식을 통해 문제에 등장하는 숫자에 대한 임베딩 벡터를 얻을 수 있었다. 한편, 수식을 세우기 위해서는 문제에 등장하지

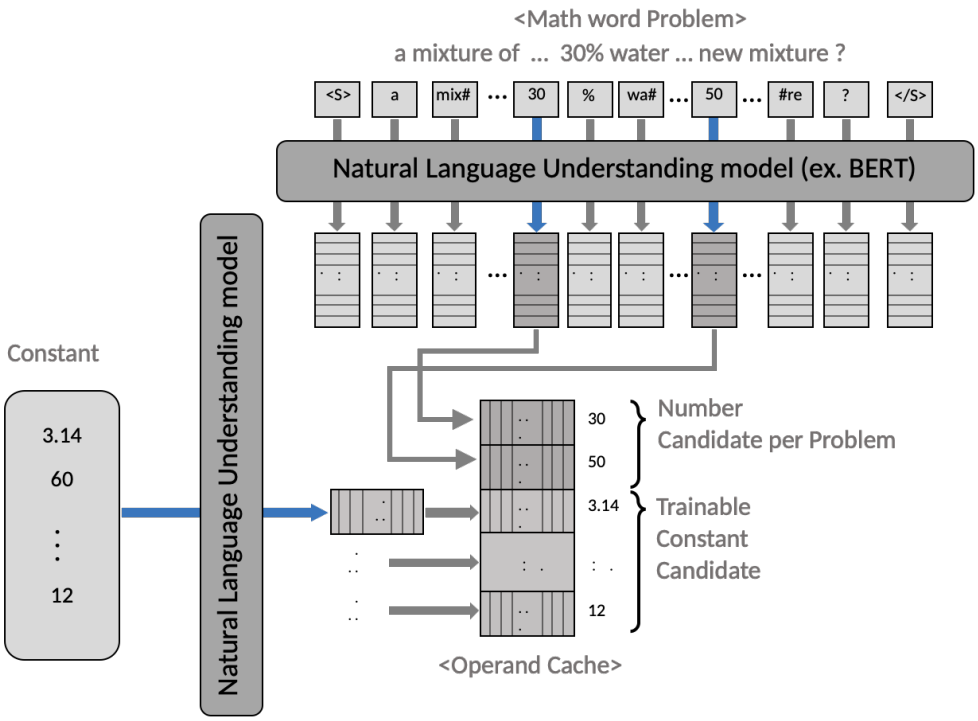

그림 3.3: 명시적 자질 추출을 사용한 피연산자 후보군 초기화

않지만, 필요한 상수들의 임베딩을 얻어야 할 필요가 있다. 그렇기 때문에 해당 데이터셋을 풀기 위해 필요한 상수들 또한 명시적 자질 추출 과정을 통해 후보군 임베딩에 추가한다. 이때 만약 숫자가 여러 개의 토큰으로 토크나이징되면 숫자 토큰의 첫 번째 토큰의 임베딩과 마지막 토큰의 임베딩을 이어 붙이고 이를 선형 레이어같은 투사 레이어에 통과시켜 임베딩을 얻었다.

그림 3.4는 연산자에 해당하는 자질 임베딩을 추출하여 후보군 임베딩을 초기화하는 과정이다. 연산자의 자질 임베딩을 추출하기 위해 각 연산자를 따로 토크나이징 하고, 나눠진 각 토큰을 사전학습 언어모델에 통과시켜 임베딩 벡터를 얻는다. 연산자도 숫자와 마찬가지로 여러 개의 토큰으로 나눠질 수 있으므로 이전과 마찬가지로 첫 번째 토큰의 임베딩과 마지막 토큰의 임베딩을 이어 붙이고 이를 투사

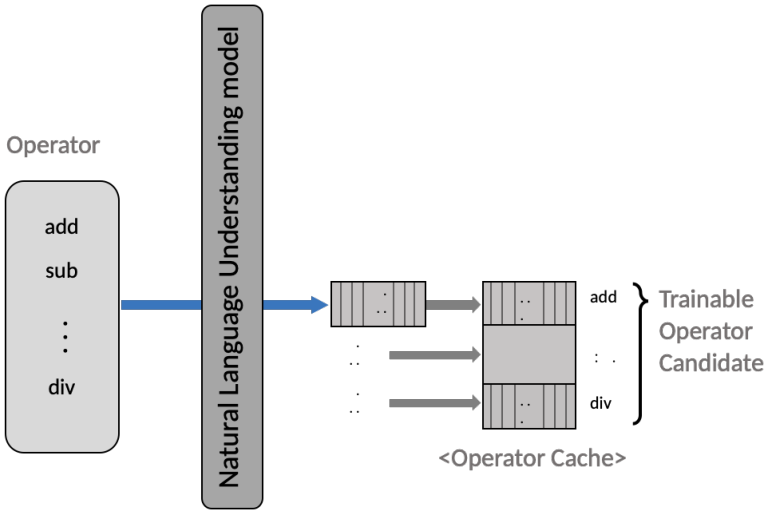

그림 3.4: 명시적 자질 추출을 사용한 연산자 후보군 초기화

레이어에 통과시켜 임베딩을 얻는다. 이렇게 얻은 연산자의 자질 임베딩은 문제마다 바뀌는 임베딩이 아니므로 이 operator cache에 저장하여 모델과 함께 학습시켜 연산자가 서술형 수학 문제에서 동작하는 역할을 학습할 수 있게 한다.

3.2.2 Elastic transformer 구조

본 아키텍처는 명시적 자질추출 방식을 활용하기 위해 제안된 구조이다. 그림 3.5은 Elastic transformer를 간략하게 나타낸 그림으로 ELASTIC 모델 구조에서 GRU를 트랜스포머 구조로 변형하여 구성하였다. 이 구조의 동작 과정은 다음과 같다.

먼저 모델에 수학 문제를 입력으로 넣으면 (1) 명시적 자질 추출 방식을 이용하여 연산자와 피연산자에 대한 임베딩 벡터를 초기화한다. (2) 자연어 이해 모델을 통해 사전학습 자연어 모델은 모델에 입력된 수학 문제로부터 문맥 벡터를 추출하고 추출된 문맥 벡터를 연산자 선택층(operator selection layer)에 넣어 연산자를 선택한다. (3) 연산자가 선택되면 선택된 연산자와 문제의 문맥 벡터를 피연산자

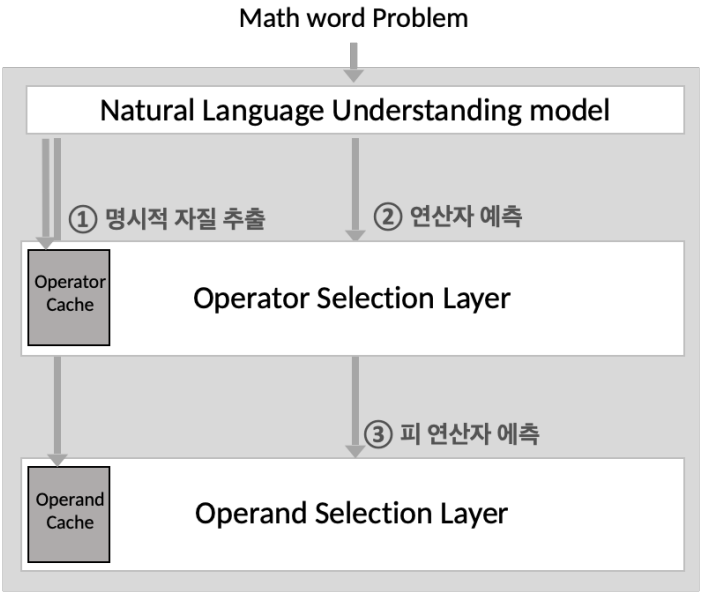

그림 3.5: 명시적 자질추출 방식을 활용하기 위한 Elastic transformer 아키텍처

선택층(operand selection layer)에 넣어 피연산자를 선택한다. 이 과정을 반복하여 전체 수식을 얻는다. 그리고 생성된 정답 수식은 피연산자로 활용될 수 있도록 피연산자 캐시에 저장된다. 본 실험에서는 이러한 구조로 문제에 해당하는 정답 수식을 추출하여 답을 도출한다. 또한 위 실험에서는 명시적 자질추출을 할 때와 하지 않을 때를 비교하기 위해 명시적 자질추출을 하지 않은 경우는 연산자, 피연산자 후보군 임베딩을 학습 가능한 임의의 벡터로 초기화한 뒤 진행한다.

3.2.3 Deductive reasoner 모델에서의 명시적 자질 추출 방식 적용

Deductive reasoner는 NLU계열 모델 중 SVAMP 데이터셋에서 최고 성능을 얻고 있는 모델로 그림 2.6과 같이 문제에 등장하는 숫자에 대한 임베딩을 추출하고, 추출된 임베딩을 이용해 수식을 도출한다. 또한 Deductive reasoner 모델은 문제로부터 숫자 임베딩을 추출할 때 문제에 등장하는 숫자를 $< quant >$로 치환하여 임베딩을

추출하기 때문에, 본 실험에서는 deductive reasoner 모델에 명시적 자질 추출 방식을 도입하여 숫자 토큰을 그대로 사용하여 문제를 풀었을 때 성능향상이 가능한지 확인한다.

3.3 Study 2: 자연어 생성 모델에서 숫자의 이해

최근 자연어 생성 모델의 크기가 기하급수적으로 커지면서 연구실 단위에서 자연어 생성 모델을 실행하기 어려워졌다. 따라서 Study 2의 모든 실험은 자연어 생성 모델 GPT기반 모델의 구현체인 `gpt3.5-turbo` API를 사용하여 진행되었다. 2개의 실험으로 구성된 Study 1과 달리 Study 2는 3개의 실험으로 이루어진다. 실험 1은 자릿수 개념을 이해하는지 확인하는 실험이고, 실험 2와 3은 프롬프트 엔지니어링을 통해 자릿수 개념을 추가하였을 때, 문제에 등장하는 숫자를 이해할 수 있는지에 대한 가능성을 확인하는 실험이다.

3.3.1 여러 토큰으로 이루어진 숫자의 정렬 실험

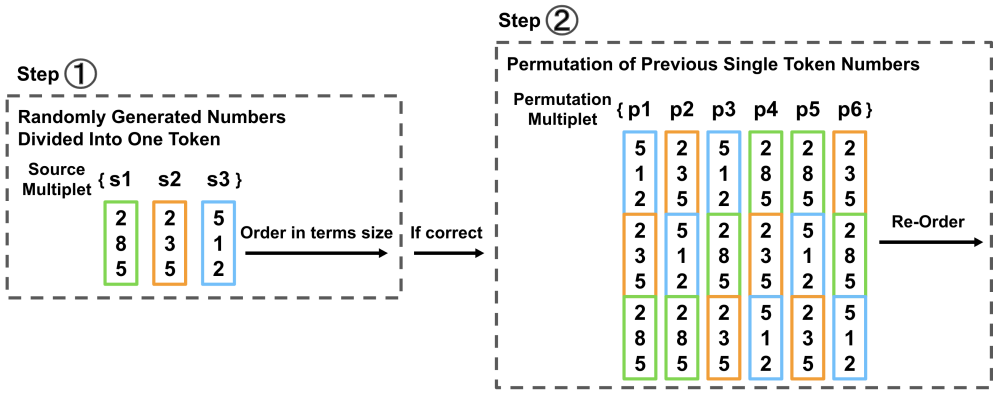

그림 3.6: 여러 토큰으로 이루어진 숫자의 정렬 실험 단계

여러 토큰으로 이루어진 수에서 자릿수(place value)의 개념을 인식할 수 있는지

판단하기 위한 실험이다. 숫자들의 크기를 이해하고 있는지 알아보기 위해서 주어진 숫자들을 증가하는 순서대로 나열하는 방법을 활용한다. 만약, 숫자들의 크기를 정확히 인지한다면, 해당 task에서 정확하게 나열이 가능할 것이다. 실험은 숫자를 구성하는 token의 수에 따라 총 2단계로 수행된다. 첫 번째 단계는, GPT가 하나의 token으로 나누어지는 숫자들을 크기가 증가하는 순서대로 배열하도록 한다. 나열하는 숫자의 수는 3개부터 6개까지 변화하면서 실험한다. 만약 GPT가 첫 번째 단계에서 주어진 숫자를 정확히 나열했다면, 해당 숫자들은 이어지는 단계의 source multiplet이 된다. 두 번째 단계에서는 source multiplet들의 permutation으로 새로운 숫자 6개의 숫자를 만든 후, 다시 해당 숫자들을 증가하는 순서대로 나열한다.

Step 1: 하나의 토큰으로 이루어진 수(source multiplet) 정렬하기 첫 번째 단계에서 우리는 GPT가 하나의 토큰으로 나누어지는 숫자의 크기를 이해할 수 있는지 확인한다. 실험을 위해 100부터 520 사이의 숫자를 3-5개를 랜덤으로 생성(그림 3.6에서 s1-s3)하여 GPT가 정확한 순서로 나열할 수 있는지 실험한다. 그림 1.2와 같이 OpenAI에서 제공하는 tokenizer를 이용해서 사용되는 입력이 어떻게 token으로 나뉘는지 쉽게 확인할 수 있는데, 1부터 1000까지의 숫자들을 확인해 본 결과 1부터 520까지의 숫자들은 전부 1개의 token으로 구성되는 것을 확인했다. 또한 100부터의 숫자를 선택한 이유는 두 번째 단계에서 만들어지는 permutation multiplet의 길이를 보장하여 여러 토큰으로 구성된 숫자를 만들기 위해서이다. 이렇게 하나의 묶음으로 처리되는 숫자들은 자릿수에 대한 개념을 알지 못하더라도 크기를 학습할 수 있다. 예를 들어, 변수 A, B, C가 있을 때, 각 변수들의 크기를 비교한 데이터가 있다면, 해당 변수들이 실제 어떤 값을 갖고 있지 않더라도 크기 비교를 할 수 있다. 따라서, 본 단계를 통해 GPT의 숫자 값의 이해 여부를 판단하는 것이 아닌, 토큰 단위에서의 숫자 크기 비교가 가능한지를 확인한다. 이 단계에서 크기 비교가 가능한 숫자 토큰들은 두 번째 단계에서 활용된다.

Step 2: 여러 개의 토큰으로 이루어진 수(permutation multiplet) 정렬하기 토큰들이 나열되는 순서에 따라 갖게 되는 자릿수를 GPT가 정확히 이해하고 있는지 확인하는 단계이다. GPT는 이전단계에서 얻은 크기비교가 가능한 숫자 토큰을 활용하여 permutation multiplet를 6(p1-p6)개 생성한다(이때, permutation의 수가 6개보다 많을 경우, 동등 비교를 위해 6개를 랜덤으로 선택한다.). 이후 GPT는 생성된 6개의 숫자를 증가하는 순서대로 하는 작업을 수행한다. 긴 길이의 숫자의 대소 관계를 파악하기 위해서는 하나의 숫자를 구성하는 연속된 토큰 $s3$, $s2$, $s1$를 자릿수가 곱해진 $s3 \times 1000^2 + s2 \times 1000^1 + s1 \times 1000^0$ 와 같은 방식으로 이해해야 숫자의 값을 이해해야 한다. 첫 번째 단계에서 각 토큰 $s1$,$s2$,$s3$의 대소관계 파악이 가능함을 확인했으므로. 두 번째 단계에서는 대소관계가 파악된 토큰들이 연속으로 나열될 때, 해당 토큰들 사이의 자릿수를 파악할 수 있어야 여러 토큰으로 구성된 숫자의 대소관계를 판단할 수 있을 것이다.

3.3.2 숫자표현의 영어표현 대체 실험

본 실험은 숫자표현을 영어표현으로 변경하였을 때 수학 문제 풀이에 도움을 줄 수 있는지 확인하는 실험이다. 이 실험은 그림 3.7과 같이 프롬프트와 dataset에 존재하는 아라비아 숫자를 영어 표현으로 대체했을 때의 성능을 측정한다. 해당 실험을 통해 우리는 CoT와 PoT를 baseline으로 하고 PoT_Eng가 SVAMP dataset을 풀 때 의 정답률의 차이를 비교한다. 또한 동일한 질문에 대한 여러 번 시도 후 가장 많이 나온 답변을 정답으로 채택하는 self-consistency 기법을 적용했을 때의 정답률도 측정한다.

숫자를 영어로 표현하는 것은 GPT에 자릿수 정보를 효과적으로 제공하는 방법으로 사용될 수 있다. 연속된 숫자 토큰의 나열에서 자릿수의 개념을 파악해야 하는 아라비아 숫자와 달리 영어 표현의 경우 자릿수가 명시적으로 각 단어에 포함된다. 예를 들어 그림 3.7의 PoT 예시에서 숫자 36의 경우, 숫자 3과 숫자 6으로 이루어져 있지만, 실제 의미하는 값은 자릿수 개념이 반영되어 30과 6으로 해석해야 한다. 그에 반해 PoT_Eng의 프롬프트에서, English expressions로 표현된 thirty six의

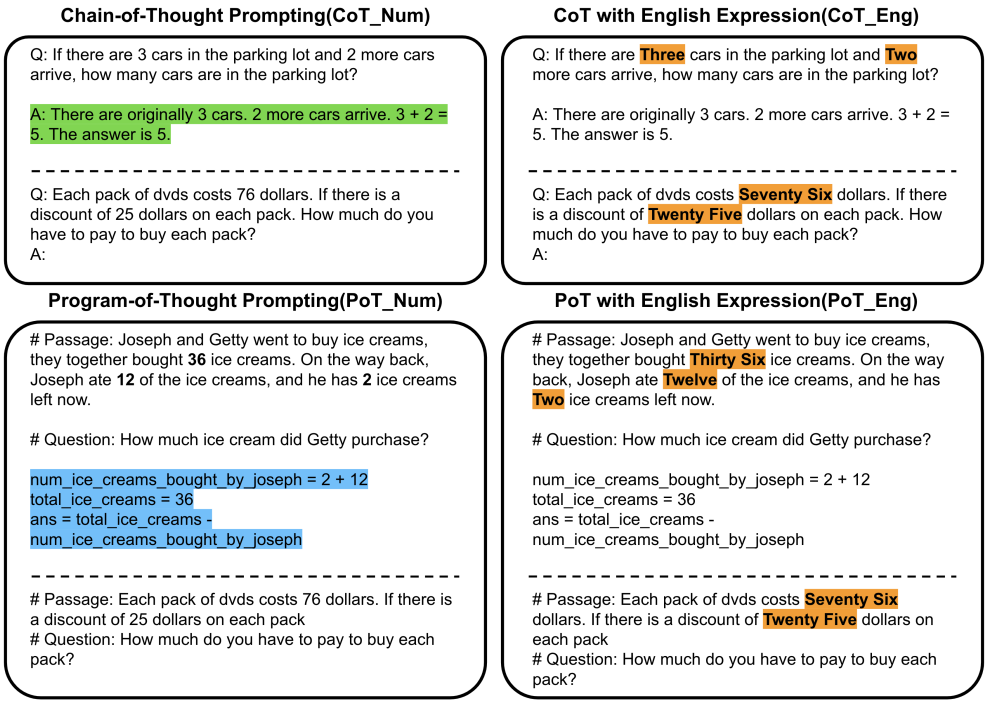

그림 3.7: CoT_Num, CoT_Eng, PoT_Num, PoT_Eng 프롬프트의 예시

경우, 각 단어에 자릿수 개념이 포함되어 있는 형태를 띈다. SVAMP dataset에 등장하는 아라비아 숫자들은 대부분 3자리 이하의 작은 숫자이기 때문에 하나의 token으로 나뉘어 처리된다. 따라서, GPT가 하나의 토큰으로 이루어진 숫자들을 정확히 이해하고 있다면, English expressions를 사용했을 때 기존 PoT에 대비해서 유사한 성능을 보이거나, 숫자의 변환 과정에서 증가된 토큰으로 인해 다소 하락한 성능을 보일 것이다. 하지만, 자릿수 개념에 모호함이 있다면, place value가 명시적(explicit)으로 반영된 PoT_Eng에서 성능 향상을 기대할 수 있다.

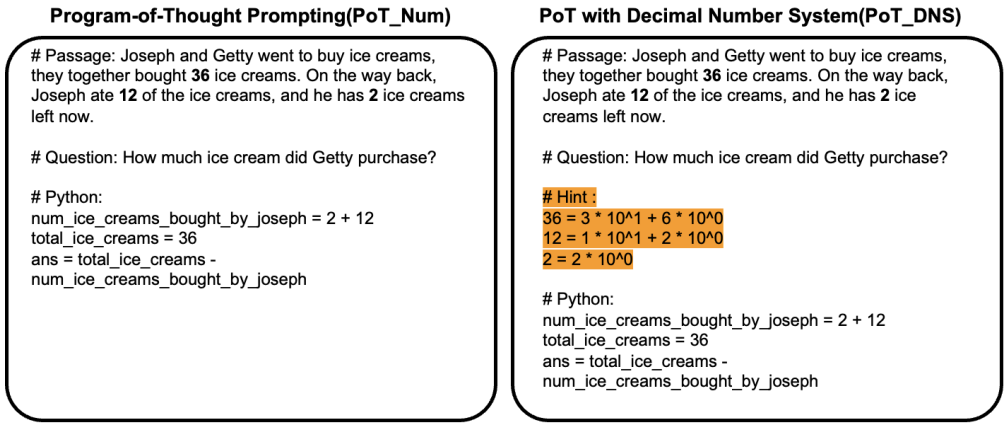

그림 3.8: PoT_Num, PoT_DNS 프롬프트의 예시

3.3.3 십진수 개념 추가 실험

본 실험은 십진수 개념을 힌트로 추가해 주었을 때 수학 풀이에 도움을 줄 수 있는지 확인하는 실험이다. 이 실험은 그림 3.8과 같이 프롬프트와 dataset에 존재하는 아라비아 숫자를 십진수 체계로 풀어쓰는 힌트를 줌으로써 자릿수 개념에 대한 이해도를 높여주어 성능을 측정하고자 했다. 해당 실험 또한 CoT와 PoT를 baseline으로 하고 PoT_DNS가 SVAMP dataset을 풀 때의 정답률 차이를 비교한다. 또한 실험 2와 마찬가지로 self-consistency를 적용하였을 때의 정답률도 측정한다.

제 4 장 실험 결과 및 분석

4.1 Study 1: 자연어 이해 모델에서의 숫자의 이해

4.1.1 명시적 자질 추출 방식 적용 실험

	Model	Val Acc.
S2S	GroupAttn [41]	21.5
	BERT-BERT [42]	24.8
	Roberta-Roberta [42]	30.3
S2T/G2T	GTS [43]	30.8
	Graph2Tree [44]	36.5
	BERT-Tree [45]	32.4
	Roberta-GTS [2]	41.0
	Roberta-Graph2Tree [2]	43.8
OURS	ROBERTA-LARGE-ELASTICTRANSFORMER	21.5
	+ *Explict Feature Extract*	**43.9**
	ROBERTA-BASE-DEDUCTREASONER	47.3
	+ *Explict Feature Extract*	**48.4**
	ROBERTA-LARGE-DEDUCTREASONER	48.9
	+ *Explict Feature Extract*	**51.7**

표 4.1: 자연어 이해 계열 모델에서의 SVAMP 데이터셋의 정답률

표 4.1은 자연어 이해 계열 모델에서의 SVAMP 데이터셋의 정답률을 나타낸다. 이때 사용한 베이스라인 모델은 sequence-to-sequence(S2S), sequence-to-tree(S2T) and graph-to-tree(G2T)기반 모델들이다. 우리가 실험한 모델은 elasitc transformer와 deductive reasoner로 명시적 자질추출 방식에 유무에 따른 정답률을 비교한다. 본 실험의 결과로 elastic transformer 모델에서는 명시적 자질추출을 수행했을 때, 수행하지 않았을 때보다 22.4% 높은 정답률을 얻을 수 있는 것을 확인했다. 또한 현재 최고성능을 보이는 모델인 RoBERTa-large를 사용한 deductive reasoner 모델에

명시적 자질 추출 방식을 도입했을 때 2.8%의 성능향상을 얻어 기존 자연어 이해 모델의 최고성능을 갱신할 수 있었다. 이는 자연어 이해 계열의 모델을 사용할 때 숫자 토큰을 사용하는 것이 숫자의 대소관계 파악할 수 있는 가능성을 주고, 이러한 가능성은 모델의 정답률 상승에 도움을 줄 수 있었음을 의미한다. 따라서 선행 연구 중 숫자 토큰을 사용하지만, 숫자를 $< quant >$로 치환하는 과정을 거치지 않아 숫자와 문자가 하나로 합쳐진 상태로 토크나이징 수 있는 선행연구[25, 24]나 문제에 등장하는 숫자 토큰을 모두 사용하지 않고 하나의 토큰만을 사용하는 선행연구[6]에서도 명시적 자질 추출을 도입할 수 있도록 모델을 수정한다면 성능이 올라갈 가능성이 있다고 예상한다.

4.2 Study 2: 자연어 이해 모델에서의 숫자의 이해

4.2.1 여러 토큰으로 이루어진 숫자의 정렬 실험

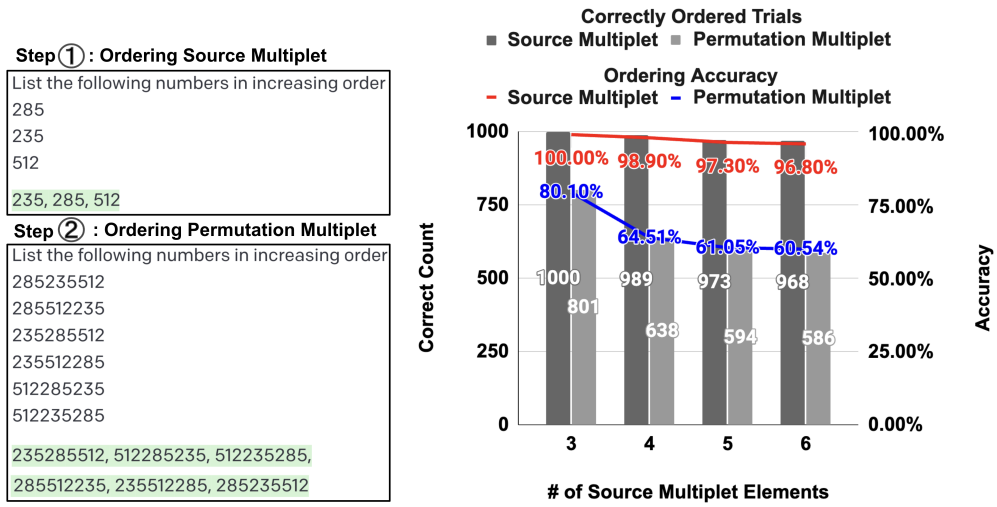

그림 4.1: (왼쪽) 실험 1의 질의 예시. (오른쪽) 3-6개의 토큰으로 이루어진 숫자에 대한 “정답 횟수”와 “정답률”

그림 4.1은 실험 1의 결과를 보여준다. 그림 4.1(왼쪽)은 숫자를 정렬하도록 하

는 질의들의 실제 예시이다. 상단 부분은 단계 1에 대한 질의 예시로 GPT는 주어진 3개의 숫자를 정확하게 정렬한다. 첫 번째 단계에서 하나의 토큰으로 이루어진 수(source multiplet)가 바른 순서로 정렬되면, 이를 바탕으로 여섯 개의 숫자(permutation multiplet)가 생성된다. 하단 부분은 단계 2에 대한 질의 예시로, GPT는 여러 개의 토큰으로 갈수록 숫자의 대소관계를 정확하게 파악하지 못하는 것을 보여준다.

그림 4.1(오른쪽)은 실험 1의 결과를 보여준다. 그래프의 x축은 실험에서 사용되는 숫자를 구성하는 토큰의 개수를 나타낸다. 왼쪽에 표시된 막대 그래프와 y축은 정답 횟수를 나타낸다. 또한 오른쪽에 표시된 선 그래프와 y축은 정렬 정확도를 나타낸다. 두 번째 단계는 첫 번째 단계에서 바르게 정렬된 숫자 토큰을 사용하여 만들어진 여섯 개의 숫자(permutation multiplet)를 정렬하는 과정이다. 그림 4.1(오른쪽)에서, 한 토큰으로 이루어진 수를 정렬하는 경우 정렬시켜야 하는 수를 늘려도 정답률이 100.00%에서 96.80%로 비교적 유지되지만, 여러 토큰으로 이루어진 수를 정렬하는 경우 정답률이 80.10%에서 60.54%로 크게 하락하는 경향을 볼 수 있다. 위 결과를 통해, 자연어 생성 모델은 각각의 토큰에 대한 대소 관계는 비교적 파악할 수 있지만, 숫자가 여러 토큰으로 구성될 때 자릿수에 대한 개념이 온전히 학습되지 않았음을 확인할 수 있다.

4.2.2 숫자표현의 영어표현 대체 실험 및 십진수 개념 추가 실험

	Temperature	CoT_Num	CoT_Eng	PoT_Num	PoT_Eng	PoT_DNS
Accuracy	0.0	76.80	79.90	**80.04**	**82.00**	**82.10**
Accuracy(SC)	0.7	81.20	82.50	82.60	83.70	82.20

표 4.2: 숫자표현의 영어표현 대체 실험 및 십진수 개념 추가실험에서 SVAMP dataset의 정답률

표 4.2은 SVAMP 데이터 셋에서 숫자 표현을 영어 표현으로 변환했을 때(PoT_Eng)와 십진수 개념을 추가했을 때(PoT_DNS)의 실험의 결과이다. 이 표에 제시된 self-

consistency의 temparature는 0.7로 설정하고, 5회 시도 중 가장 많이 등장한 답을 답으로 결정했다. 모든 경우에서 자릿수 개념을 추가했을 때(CoT_Eng, PoT_Eng, PoT_DNS), 숫자 표현을 사용하는 것(CoT_Num, PoT_Num)보다 높은 성능을 보였다.

영어 표현을 사용했을 때 모델 성능이 증가한 것에 대해서는 두 가지 설명이 가능하다. 첫 번째, 영어 표현은 명시적으로 자릿수의 개념을 포함하고 있기 때문에, 암시적으로 자릿수의 개념을 포함하는 숫자 표현보다 모델이 이해하기 쉬울 수 있을 수 있다. 두 번째, 단순히 GPT가 숫자보다 자연어를 더 잘 처리할 수 있도록 훈련되어 있어서 일 수 있다.

추가로, PoT_Eng에서 나타나는 성능 향상(1.96%, PoT_Num)이, CoT_Eng에서 나타나는 성능 향상(3.10%, CoT_Num)에 비해 더 큰 것을 확인할 수 있다. CoT는 수학 문제에 답을 하기 위해서 문제에 대한 이해뿐만 아니라, 정답을 직접 도출해야 한다. 그러나 PoT의 경우, CoT와 동일하게 문제를 이해하고 풀이 과정을 만들어야 하는 것까지는 동일하지만, 문제에 활용되는 숫자들을 변수에 넣고, 숫자들의 값에 대한 이해를 요구하는 계산은 python interpreter에 맡긴다. 따라서 산술적인 연산을 직접 해야 하는 CoT가 자릿수에 대한 정보의 영향을 더 크다는 것을 볼 수 있다.

십진수 개념을 추가 했을 때 또한 성능이 향상되었다. 그러나 self-consistency를 수행했을 때는 성능이 향상되지 않았는데, 이를 통해 십진수의 개념을 추가하여 각 토큰의 자릿수 개념을 이해하도록 하는 것에 어려움이 있을 수 있다고도 해석할 수 있다. 현재 자릿수를 이해하지 못하는 이유는 토크나이징 단계에서 거치게 되는 구조적인 한계로 하나의 숫자가 여러 토큰으로 쪼개지는 것인데, 10진수의 개념을 추가한 것은 그 문제를 해결한 것이 아니기 때문이다. 따라서 자릿수의 개념을 주기 위해 토크나이징 이후 생기는 토큰별로 자릿수의 개념을 추가하는 실험이 필요하다.

제 5 장 결론 및 한계점

위 실험의 결과로 숫자의 대소 관계 이해에 도움을 줄 수 있는 정보를 추가해 주었을 때, 인공지능 모델의 수학 문제 풀이에서 성능 향상의 가능성을 확인할 수 있었다. Study 1을 통해 elastic transformer모델과 deductive reasoner 모델에서 임의의 문자열 대신 숫자 토큰을 넣어주었을 때 성능 향상을 확인할 수 있었다. 이는 숫자 토큰을 사용하여 숫자의 대소관계를 파악할 수 있도록 해주는 것이 인공지능 모델이 수학 문제를 해결할 때 도움을 줄 수 있다는 것을 확인해 주었다. Study 2의 첫 번째 실험을 통해서는 숫자가 많은 토큰으로 쪼개질수록 대소관계 파악에서 성능이 저하되는 것을 확인할 수 있었다. 이 결과를 통해 GPT모델의 경우 숫자를 이해할 때 자릿수를 이해하는 것에 어려움을 겪을 수 있다는 것 확인할 수 있었다. 또한 두 번째 실험에서는 문장형 수학 문제 풀이의 성능을 통해, 자릿수를 명시적으로 인식할 수 있도록 했을 때 정답률이 향상되는 것을 확인할 수 있었다. 그리고 세 번째 실험을 통해 자릿수의 정의를 추가해 주는 것이 성능 향상에 큰 영향을 끼치지 못한다는 것을 확인할 수 있었다.

본 연구의 한계점 Study 1의 경우, NLU계열의 모델에서는 숫자 자체를 문제 풀이에 사용하지 않고 있었기 때문에 숫자의 임베딩을 문제 풀이 과정에 추가하는 것에 집중하고, 이후 자릿수 개념을 이해하고 있는지에 대한 확인은 수행하지 않았다. Study 2의 첫 번째 실험의 경우, GPT가 자릿수 개념을 이해하지 못할 수 있다는 것을 보여주기 위해 세 자리로 이루어진 숫자의 토큰만을 실험에 이용한다. 하지만 수학 문제에서는 한 자리나 두 자리수가 자주 등장하기 때문에, 다른 길이의 토큰에 대해서도 유사한 현상이 발생하는지 알아보기 위해 더 작은 자릿수에 대한 실험을 진행해볼 수 있다. 두 번째 실험의 경우, SVAMP 데이터셋의 숫자들은 상대적으로 작으며, 대부분 한자리 또는 두 자리이다. 만약 숫자가 더 커지면 실험 결과가 재현되지 않을 수 있기 때문에, 이 현상을 완전히 검증하기 위해서는 세 자리 이상의 더 큰

숫자들에 대한 실험이 필요하다. 세 번째 실험의 경우 self-consistency를 수행했을 때는 기존 PoT_Num에 비해 성능향상이 되지 않는다는 한계점이 존재했다.

본 연구의 의의 이러한 한계에도 불구하고, Study 1의 경우 숫자 토큰을 사용하였을 때 모델의 풀이 성능이 기존보다 상승할 수 있음을 보여주었고, Study 2에서는 숫자가 GPT의 토큰으로 표현될 때 자릿수에 대한 개념이 충분히 표현되지 않을 가능성이 있음을 보여주었다. 따라서, GPT 모델의 숫자가 여러 토큰으로 분리되어 있음에도 불구하고 자릿수를 이해하도록 훈련될 수 있다면, 모델은 수학 문제를 푸는 작업이나 숫자의 자릿수 이해를 요구하는 문제를 해결하는 데 더 높은 성능을 보일 수 있을 것이다. 이를 위해 숫자를 이해할 수 있도록 언어모델을 미세 조정(find-tuning)하는 것도 가능한 방향성이 될 수 있다.

참고 문헌

[1] Yorick Wilks. Natural language understanding systems within the ai paradigm—a survey and some comparisons. *American Journal of Computational Linguistics*, 1976.

[2] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, 2021.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[6] Jiaxin Zhang and Yashar Moshfeghi. Elastic: numerical reasoning with adaptive symbolic compiler. *arXiv preprint arXiv:2210.10105*, 2022.

[7] Zhanming Jie, Jierui Li, and Wei Lu. Learning to reason deductively: Math word problem solving as complex relation extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5944–5955, 2022.

[8] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[9] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[10] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

[11] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023.

[12] Peter Clark. Elementary school science and math tests as a driver for ai: Take the aristo challenge! In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, pages 4019–4021, 2015.

[13] Peter Clark and Oren Etzioni. My computer is an honor student—but how intelligent is it? standardized tests as a measure of ai. *AI Magazine*, 37(1):5–12, 2016.

[14] Daniel Bobrow et al. Natural language input for a computer problem solving system. 1964.

[15] Charles R Fletcher. Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers*, 17(5):565–571, 1985.

[16] Ma Yuhui, Zhou Ying, Cui Guangzuo, Ren Yun, and Huang Ronghuai. Frame-based calculus of solving arithmetic multi-step addition and subtraction word problems. In *2010 Second International Workshop on Education Technology and Computer Science*, volume 2, pages 476–479. IEEE, 2010.

[17] Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*, pages 523–533, 2014.

[18] Sowmya S Sundaram and Deepak Khemani. Natural language processing for solving simple word problems. In *Proceedings of the 12th international conference on natural language processing*, pages 394–402, 2015.

[19] Arindam Mitra and Chitta Baral. Learning to use formulas to solve simple arithmetic problems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2144–2153, 2016.

[20] Chao-Chun Liang, Kuang-Yi Hsu, Chien-Tsung Huang, Chung-Min Li, Shen-Yu Miao, and Keh-Yih Su. A tag-based english math word problem solver with understanding, reasoning and explanation. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Demonstrations*, pages 67–71, 2016.

[21] Yan Wang, Xiaojiang Liu, and Shuming Shi. Deep neural solver for math word problems. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 845–854, 2017.

[22] Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. Translating a math word problem to an expression tree. *arXiv preprint arXiv:1811.05632*, 2018.

[23] Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[24] Bugeun Kim, Kyung Seo Ki, Donggeon Lee, and Gahgene Gweon. Point to the expression: Solving algebraic word problems using the expression-pointer transformer model. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3768–3779, 2020.

[25] Kyung Seo Ki, Donggeon Lee, Bugeun Kim, and Gahgene Gweon. Generating equation by utilizing operators: Geo model. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 426–436, 2020.

[26] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[27] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[28] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020.

[29] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[30] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[31] OpenAI. Gpt-4 technical report, 2023.

[32] Subhro Roy and Dan Roth. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*, 2016.

[33] Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597, 2015.

[34] Subhro Roy and Dan Roth. Unit dependency graph and its application to arithmetic word problem solving. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[35] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022.

[36] Jaehun Jung, Lianhui Qin, Sean Welleck, Faeze Brahman, Chandra Bhagavatula, Ronan Le Bras, and Yejin Choi. Maieutic prompting: Logically consistent reasoning with recursive explanations. *arXiv preprint arXiv:2205.11822*, 2022.

[37] Jiacheng Liu, Skyler Hallinan, Ximing Lu, Pengfei He, Sean Welleck, Hannaneh Hajishirzi, and Yejin Choi. Rainier: Reinforced knowledge introspector for commonsense question answering. *arXiv preprint arXiv:2210.03078*, 2022.

[38] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

[39] Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies*, pages 1152–1157, 2016.

[40] Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing english math word problem solvers. *arXiv preprint arXiv:2106.15772*, 2021.

[41] Jierui Li, Lei Wang, Jipeng Zhang, Yan Wang, Bing Tian Dai, and Dongxiang Zhang. Modeling intra-relation in math word problems with different functional multi-head attentions. In *Proceedings of the ACL*, 2019.

[42] Yihuai Lan, Lei Wang, Qiyuan Zhang, Yunshi Lan, Bing Tian Dai, Yan Wang, Dongxiang Zhang, and Ee-Peng Lim. Mwptoolkit: An open-source framework for deep learning-based math word problem solvers. *arXiv preprint arXiv:2109.00799*, 2021.

[43] Zhipeng Xie and Shichao Sun. A goal-driven tree-structured neural model for math word problems. In *Proceedings of IJCAI*, 2019.

[44] Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. Graph-to-tree learning for solving math word problems. In *Proceedings of ACL*, 2020.

[45] Zhongli Li, Wenxuan Zhang, Chao Yan, Qingyu Zhou, Chao Li, Hongzhi Liu, and Yunbo Cao. Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems. *arXiv preprint arXiv:2110.08464*, 2021.

ABSTRACT

A Study on the Understanding of the Number Concept of the AI Model for Math Word Problem Solving

Jisu An
Graduate School of Convergence Science and Technology
Seoul National University

The field of math word problem solving, an intriguing area of study since the 1960s, has been consistently researched. As AI advances, attempts to use AI for solving sentence-type mathematical problems have been increasing. However, recent concerns that the math word problem solving models do not actually understand and solve the problem through reasoning, but derive answers by appropriately combining the numbers appearing in the problem, have brought ambiguity to the understanding of these models. To comprehend mathematical problems, understanding the numbers that appear in the problem is prerequisite, and thus this paper conducts two studies to aid in the understanding of numbers during the problem-solving process.

Study 1 proposes an explicit feature extraction method to address the issue arising from the limited use of number tokens in the problem-solving process of pre-trained BERT-series language models. This method aids in understanding the magnitude of the numbers that appear in the problem. Furthermore, the application of this explicit

feature extraction method has shown a 2.8% performance improvement in the natural language understanding model among those in the SVAMP dataset, surpassing the performance of the previous best-performing model. These results confirm the potential of using the number tokens that appear in the problem to help comprehend the magnitude, which can subsequently increase the accuracy of the model.

Study 2 demonstrates the problem that the implementation of the pre-trained GPT-series language model, `gpt3.5-turbo`, fails to fully comprehend the concept of digit position when discerning the magnitude of numbers comprised of multiple tokens. It proposes a strategy to supplement this issue. The result of this experiment showed a 2.06% increase in accuracy compared to the best performance of the previous prompt strategy using `gpt3.5-turbo`. These results confirm the potential that if the concept of digit position is incorporated when natural language generation models solve math word problems, it can aid in understanding the magnitude of number tokens, which can then increase the accuracy rate.

Keywords: Number Understanding, BERT, GPT, ChatGPT, Prompt Engineering, Chain of thought

Student Number: 2021-27764