Pusan National University Computer
Science and Engineering Technical Report
2025-09

- 영어 객관식 문제 정답 생성 서비스 개발

저자1 강형원

저자2 박규태

저자3 주연학

지도교수 탁성우

목 차

1. 서론

1.1. 연구 배경

- 최근 AI의 발전으로, LLM이라고 하는 대형 언어 모델들의 사용량이 늘어나고 일상 생활에 많이 이용되고 있다. 이런 LLM의 발전 속도는 대단하여, 코딩에도 사용되며 일상 생활의 여러 문제점을 해결해주고 있다.
- 따라서 앞으로 LLM을 활용하는 능력은 현대인에 있어 필수가 될 것이며, LLM을 활용하여 원하는 문제를 더 정확하게 해결해 낼 수 있는 서비스를 만들어 내는 것은 중요한 기술이 될 것이다.
- 이러한 이유에서, 우리는 일상생활에서 마주칠 수 있는 문제점 중 하나인 “영어 객관식 문제 정답 생성”에 특화된 AI 모델을 제안 및 구현하고자 한다.

1.2. 기존 문제점

- LLM의 특성으로 인하여 다양한 응용 개발이 가능한데, 그 중 Re:Fresh 팀이 관심이 있어하는 분야는 LLM을 이용한 영어 문장 분석과 추론을 하여 영어 객관식 문제를 풀어내는 문제를 해결하고자 한다. 다만, 국내 외에서 이런 분야와 관련된 특정 응용 서비스는 거의 없는 실정이다.
- 그렇기 때문에, 해당 분야에서 정답률을 조금 더 높일 수 있지 않을까 기대한다.

1.3. 연구 목표

- LLM을 영어 독해 객관식 문제 풀이에 맞게 개량을 하고 해당 모델이 추론을 하게 하여 이를 통해 객관식 답을 유추하고자 한다.
- 단일 모델의 성능을 개량하는 방법과 복합 모델을 사용하여 성능을 개량하는 방법을 통해 그 정확도를 최대한 높이는 방법을 연구한다.
- 이렇게 연구한 최종 모델의 결과를 편하게 보기 위하여 웹 api 형태로 가공하여 웹 서비스 형식으로 서비스 시연을 한다.

2. 연구 배경

2.1. LLM(Large Language Model)

LLM은 언어 생성 ai 모델로, 우리가 흔히 접하는 ChatGPT가 대표적인 LLM이다. LLM은 사전에 학습되어 있던 데이터의 크기에 따라 기본 모델의 정확도 및 성능이 달라지게 되는데, 사전학습된 데이터의 양이 작으면 기본 정확도는 낮아지지만 이를 개량하는 속도는 빠르며, 이식성이 뛰어나다. 사전학습된 데이터의 양이 크면 기본 정확도가 높아지지만 학습 비용이 비싸지며, 모델을 pc에 올려놓는 비용도 비싸 이식성이 떨어진다. 그렇기 때문에 LLM을 활용한 서비스를 개발할 때는 현재 개발이 진행되는 환경과 필요한 정확도, 사용될 환경 등을 고려하여 모델을 선정할 필요가 있다.

현재 Re:Fresh 팀이 가지고 있는 환경은 아래와 같다.

Linux Server - Ubuntu 20.04.6

Python 3.10.18 설치된 conda 가상환경

pip 25.1.1

GPU - RTX A 5000 (24GB) - CUDA 12.7

CPU - AMD EPYC

RAM - 1TB

2.2. LLM 모델을 활용한 학습 및 추론, 평가

HuggingFace의 Transformer 라이브러리를 활용하여 LLM 모델을 손쉽게 불러올 수 있고, 이에 학습을 진행할 수 있으며 학습된 모델을 활용하여 추론도 진행할 수 있다. 모델을 활용하는데 있어 핵심이 되는 것은 “프롬프트”로, 모델은 프롬프트에 기반하여 추론을 진행하며 그 결과를 생성한다. 모델을 학습하고 추론하는 상세 코드는 아래 모델 연구에서 다룬다.

LLM 모델에 대해 추가적인 학습을 진행하는 것을 파인튜닝이라고 하며, “미세 조정”이라고도 한다. 파인 튜닝을 통해 모델에 추가적인 정보를 제공하여 우리가 원하는 방향으로 모델의 출력을 제어한다. 파인튜닝을 위해서는 우리가 제공하고자 하는 분야에 대해 대량의 데이터셋이 필요하며, 소량으로도 파인 튜닝의 시도 자체는 가능하지만 그 효과를 크게 보기는 어렵다. 우리가 원하는 분야에 대해 생성되는 정답의 정확도를 더 높이기 위해서는 충분히 많은 양의 데이터셋을 확보하는 것이 필수이다.

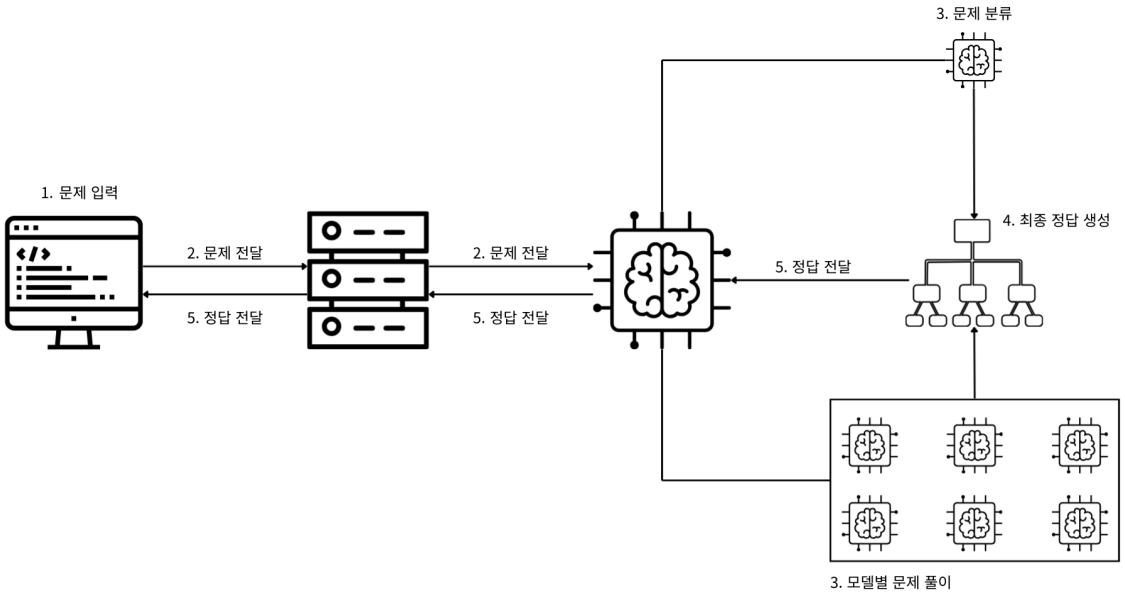
LLM 모델을 하나만 사용하는 것이 아닌 여러 개를 사용하여 그 결과를 혼합하여 사용할 수도 있다. 이를 앙상블(Ensemble)이라고 하며, 자세한 내용은 연구 내용 파트에서 다룬다. 앙상블 모델의 경우 여러 모델을 혼합해서 사용하기 때문에, 사용되는 모델들의 비중을 계속해서 조절해 나가면 모델의 정확도는 점차 사용 모델 중 가장 좋은 모델에 수렴하는 형태로 모델이 구성된다.

본 연구에서는 여러 사전학습된 LLM을 Hugging Face Transformers로 불러와 개별 학습을 진행하고, 각 미세 조정된 모델들의 출력을 앙상블해 최종 성능을 끌어올린다.

3. 연구 내용

3.1. 전체 시스템 구성도

연구를 위한 전체적인 시스템 구성도는 아래 그림과 같다.



연구의 핵심은 모델 서버이다. 모델 서버에 문제가 전달이 되면 모델 서버는 해당 문제에 대해 문제 풀이를 실행하여, 본인이 내놓은 결과를 웹 api 형식으로 중계서버에 전달하고, 중계서버는 받은 답을 프론트 웹 페이지에 전달하여 그 답을 가시적으로 사용자에게 보여준다.

모델 서버에서 문제 풀이를 진행할 때는 다음과 같은 과정을 거친다. 문제가 모델 서버에 전달이 되면, 모델 서버는 문제를 각각 문제 분류 모델과 문제 풀이 모델에 전달하여 어떤 종류의 문제인지, 그리고 각 모델이 문제를 풀었을 때 보기에 대한 확률 값이 어떻게 되는지를 산출하게 한다. 각 모델 별 산출된 확률 값은 이후 분류 모델이 분류한 문제의 종류에 따라 모델 별 가중치를 고려하여 최종 정답 산출에 반영이 되고, 그렇게 최종적으로 산출한 정답을 도출한다. 이후 이 결과 값을 웹 api 형식으로 전달하여 중계 서버, 프론트 웹페이지로 전달하게 한다.

3.2. 학습을 위한 데이터셋 선정

3.2.1. RACE

RACE Dataset [1] 은 자연어 처리 분야에서 자주 사용되는 Open Dataset 중 하나로, 28000여개의 지문과 10만여개의 질문으로 구성된 규모가 큰 영어 객관식 문제 데이터셋이다. 데이터셋 이용 규정 상 상업적인 용도로는 사용할 수 없으며, 상업적인 용도로의 가공 역시 금지된다.

RACE Dataset은 HuggingFace를 통하여 쉽게 불러와 사용하는 것이 가능하다. HuggingFace의 datasets를 import하여 load_dataset 함수를 통해 불러오는 것이 가능하며, 매개변수를 통해 중학 수준의 문제 / 고등 수준의 문제 / 모든 문제 불러오기가 가능하다. Re:Fresh 팀은 RACE dataset의 가공형 중 “ehovy/race”를 선택하여 해당 데이터셋을 활용한다.

```
from datasets import load_dataset
dataset = load_dataset("ehovy/race", "all") # ”all”, “middle”, “high”
```

RACE Dataset의 구성은 [example_id, article, answer, question, options] 로 구성되어 있다. example_id는 해당 문제의 난이도와 문제 번호 등을 기록한 필드이며, article은 문제에 대한 지문을 제공한다. answer는 객관식 문제에 대한 정답을 ABC order로 표기하여 제공한다. question은 지문에 대한 문제를 제공해주며, options는 문제에 대한 보기를 사지선다로 제공한다.


```
{
        "example_id":"high19088.txt",
        "article":"Last week I talked with some of my students about what they wanted to do
        after they graduated, and what kind of job prospects they thought they had. Given
        that I teach students who are training to be doctors, I was surprised do find that
        most thought that they would not be able to get the jobs they wanted without
        "outside help". "What kind of help is that?" I asked, expecting them to tell me
        that they would need a or family friend to help them out. "Surgery ," one replied.
        I was pretty alarmed by that response. It seems that the graduates of today are
        increasingly willing to go under the knife to get ahead of others when it comes to
        getting a job . One girl told me that she was considering surgery to increase her
        height. "They break your legs, put in special extending screws, and slowly expand
```

```
        the gap between the two ends of the bone as it re-grows, you can get at least 5 cm
        taller!" At that point, I was shocked. I am short, I can't deny that, but I don't
        think I would put myself through months of agony just to be a few centimetres
        taller. I don't even bother to wear shoes with thick soles, as I'm not trying to
        hide the fact that I am just not tall! It seems to me that there is a trend towards
        wanting "perfection" , and that is an ideal that just does not exist in reality. No
        one is born perfect, yet magazines, TV shows and movies present images of thin,
        tall, beautiful people as being the norm. Advertisements for slimming aids, beauty
        treatments and cosmetic surgery clinics fill the pages of newspapers, further
        creating an idea that "perfection" is a requirement, and that it must be purchased,
        no matter what the cost. In my opinion, skills, rather than appearance, should
        determine how successful a person is in his/her chosen career.",
        "answer":"C",
        "question":"We can know from the passage that the author works as a_.",
        "options":[
                "doctor",
                "model",
                "teacher",
                "reporter"
        ]
}
```


Load 된 dataset의 형식을 json으로 다듬으면 다음과 같다.


```
{

        "DatasetDict":{
                "test":{
                        "features":[
                                "example_id",
                                "article",
                                "answer",
                                "question",
                                "options"
                        ],
                        "num_rows":4934
                },
                "train":{
                        "features":[
                                "example_id",
                                "article",
                                "answer",
                                "question",
                                "options"
                        ],
                        "num_rows":87866
                },
                "validation":{
                        "features":[
                                "example_id",
                                "article",
                                "answer",
                                "question",
                                "options"
                        ],
                        "num_rows":4887
                }
        }
}
```

해당 dataset은 python의 dictionary와 유사하게 사용할 수 있다. dataset['train'], dataset['validation'], dataset['test'] 를 통해 각각의 dataset을 불러올 수 있으며, [index]를 붙여 문제별로 추출도 가능하며 ['feature']를 덧붙여 세부적인 내용을 추출하는 것도 가능하다.

3.2.2. CLOTH

CLOTH dataset [2] 은 영어 빈칸 문제들에 대해 제공하는 dataset이다. RACE와 마찬가지로 상업적인 이용이 불가능하며, 상업적인 목표로 dataset을 가공하는 것 역시 금지된다.

RACE와 마찬가지로 HuggingFace를 통한 불러오기가 가능하다. Re:Fresh 팀은 CLOTH dataset의 가공형인 "AndyChiang/cloth" dataset을 활용한다.

```
from datasets import load_dataset
dataset = load_dataset("AndyChiang/cloth")
```

CLOTH dataset의 구조는 ['distractors', 'sentence', 'answer'] 로 구성되어 있다. 'distractors'는 오답인 문항 3개를 저장하고 있는 필드이며, sentence는 [MASK] 라는 필드로 빈칸을 표기하여 풀어야할 문항을 보여준다. 그리고 answer 필드에는 최종적으로 들어가는 정답을 보여준다.


```
{
        "distractors":[
                "ached",
                "beat",
                "rose"
        ],
        "sentence":"My heart [MASK] when I was asked to the back room by the immigration
        officer.",
        "answer":"sank"
}
```


영어 객관식 문제와는 거리가 있는 형태의 데이터셋 구조이기 때문에, 사용을 위해서는 반드시 가공을 해야한다. RACE가 객관식 문제로서 잘 어울리는 dataset 구조를 가지고

있기 때문에, RACE dataset의 구조를 참고하여 dataset을 가공하여 사용하도록 한다. 이때, 일부 한국어로 문제가 제시되는 것에 내성을 갖기 위해 일부 문제에 한해 한국어로 문제를 제작한다.

```
import random

cloth_train = cloth_dataset['train']
train_len = len(cloth_train)

index_to_letter = ["A", "B", "C", "D"]

korean_question = [
    "빈칸에 들어갈 말로 적절한 것은?",
    "빈칸에 들어갈 적절한 말을 고르시오.",
    "다음 중 빈칸에 들어가기 가장 적절한 말은?"
]

example_id = []
articles = []
questions = []
options = []
labels = []

for i in range(train_len):
    example_id.append(i)
    modified_setence = cloth_train[i]['sentence'].replace(' [MASK] ', '_')
    articles.append(modified_setence)
    rand_num = random.random()
    if(rand_num < 0.7):
      questions.append("Which word best fits in the blank?")
    else:
```

```
        questions.append(random.choice(korean_question))

    choices = cloth_train[i]['distractors'] + [cloth_train[i]['answer']]
    shuffled = random.sample(choices, k=4)

    options.append(shuffled)
    labels.append(index_to_letter[shuffled.index(cloth_train[i]['answer'])])

data = {
    "example_id": example_id,
    "article": articles,
    "question": questions,
    "options": options,
    "answer": labels
}

new_cloth_dataset_train = Dataset.from_dict(data)
```

이렇게 제작한 Dataset을 DatasetDict 객체에 담아 저장하면, HuggingFace에서 불러온 데이터셋처럼 데이터셋을 사용할 수 있게 된다. local에서 데이터셋을 불러와, RACE Dataset을 사용하는 것처럼 사용할 수 있게 된다.

```
ds_dict_cloth = DatasetDict({
    "train": new_cloth_dataset_train,
    "validation": new_cloth_dataset_validation,
    "test": new_cloth_dataset_test
})

ds_dict_cloth.save_to_disk("my_cloth")

real_loaded_cloth = load_from_disk("my_cloth")
```

3.2.3. 한국 수능/검정고시

한국 수능/검정고시 문제는 교육청에서 제공하며, 역시나 상업적인 용도로의 활용은 불가능하다. 또한 따로 데이터셋을 주는 것이 아니기 때문에, 직접 데이터셋을 제작해야할 필요가 있다.

최초 pdf 파일의 파싱을 통해 문제를 추출하려고 했으나, pdf의 형식이 통일되지 않아 파싱 중 일부 문장이 잘리고 형식이 깨지는 등의 애로 사항을 겪어, 대략 1200여 문제에 대해 수작업하여 dataset 제작을 위한 csv 파일을 제작한다.

제작한 csv파일을 load 해와, pandas를 통해 하나의 변수에 저장하고 비율을 정하여 train, validation, test로 분리한다.

```
train_df, temp_df = train_test_split(
    df_all, #csv 파일들을 불러와 합친 것
    test_size=0.2,
    random_state=42,
    shuffle=True
)

val_df, test_df = train_test_split(
    temp_df,
    test_size=0.5,
    random_state=42,
    shuffle=True
)

train_df = train_df.reset_index(drop=True)
val_df = val_df.reset_index(drop=True)
```

```
train_df['options'] = train_df['options'].apply(ast.literal_eval)
val_df['options'] = val_df['options'].apply(ast.literal_eval)
test_df['options'] = test_df['options'].apply(ast.literal_eval)
test_df = test_df.reset_index(drop=True)

data = {
    "example_id": list(range(len(train_df))),
    "article": train_df["article"].tolist(),
    "question": train_df["question"].tolist(),
    "options": train_df["options"].tolist(),
    "answer": train_df["answer"].tolist(),  # "A", "B", "C", "D"
}

new_train_df_dataset_train = Dataset.from_dict(data)

...
```

분리한 데이터셋은 이후 DatasetDict 객체에 담아 로컬에 저장한다. 저장하고 나면, 앞선 RACE와 CLOTH 처럼 HuggingFace에서 불러온 데이터셋을 사용하는 것처럼 데이터셋을 사용할 수 있게 된다.

```
ds_dict_race_korean = DatasetDict({
    "train": new_train_df_dataset_train,
    "validation": new_train_df_dataset_val,
    "test": new_train_df_dataset_test
})

ds_dict_race_korean.save_to_disk('my_korean')

my_korean = load_from_disk("my_korean")
```

3.3. 세부 모듈의 구성

3.3.1. Base Model 선정

파인튜닝이 가능한 HuggingFace 오픈 웨이트 공개 모델 중 파라미터 4B 이하인 모델을 1차적으로 선택 후 각 모델의 기술 문서를 참고하여 최종 Google의 Gemma3-4B-it 모델과 Microsoft의 Phi-4-Mini-instruct를 최종 모델로 선택한다.

Gemma3-4B-it는 해당 모델의 테크니컬 보고서 [3] 에서 제시하는 향상된 포스트-트레이닝 레시피를 통해 지시를 따르는 능력과 다국어 성능 등에서 이전 세대 모델 대비 개선을 이루었다고 명시하고 있다.

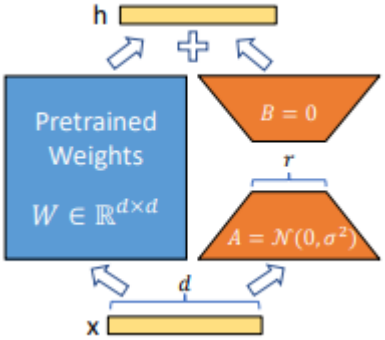
Phi-4-Mini-instruct는 해당 모델의 테크니컬 보고서 [4] 에서 동급 오픈 모델 대비 유의미한 성능 향상으로 38억 개의 매개변수로 구성된 작은 크기에도 DeepSeek-R1-Distill-Qwen-7B 및 DeepSeek-R1-Distill-Llama-8B를 포함하여 훨씬 더 큰 모델과 동등하거나 더 나은 추론 성능을 달성했다고 명시하고 있다.

위 두 모델의 오류가 상보적으로 나올 것으로 예상되며 앙상블로 결합하면 단일 모델 대비 전체 성능이 개선될 가능성이 높다고 판단된다. 따라서 위 두 베이스 모델 각각에 대해 서로 다른 3가지 미세조정 레시피를 적용해 총 6개의 모델을 제작할 계획이다.

3.3.2. 공통 적용 기술 - Parameter Effective Finetuning

사전 학습 모델을 특정 도메인에 맞게 성능 향상시키기 위해 미세조정을 수행한다. 전체 파라미터를 업데이트하는 Full fine-tuning은 연산, 메모리 비용이 매우 커 실용성이 떨어진다. 이에 따라 매개변수 효율 미세조정 (PEFT)을 적용해 적은 비용으로 파인튜닝을 진행한다.

Low-Rank Adaptation(LoRA)은 PEFT 기법 중 하나로 기존 베이스 모델의 가중치를 동결 후 별도로 구분된 어댑터를 학습하는 방식이다. 모델 추론 시에는 기존 모델의 출력과 어댑터의 출력을 더해 미세조정의 효과를 낸다. LoRA 기법을 통해 Full fine-tuning에 비해 최대 3배 까지 하드웨어 성능 요구를 낮출 수 있다. [5]



LoRA 어댑터 학습은 TRL의 SFTTrainer에 peft.LoraConfig를 전달해 수행한다.

```
peft_config = LoraConfig(
        lora_alpha= 32,
        lora_dropout=0.05,
        r=16,
        bias="none",
        target_modules="all-linear",
        task_type="CAUSAL_LM",
        modules_to_save=["lm_head", "embed_tokens"]
    )
# …
trainer = SFTTrainer(
        model=model,
        args=args,
        train_dataset=train_dataset,
        eval_dataset=eval_dataset,
        peft_config=peft_config,
        processing_class=tokenizer,
    )
```

인자 값 'r' 은 업데이트 값을 랭크로 근사할 때의 랭크 값이다. R 값이 커질수록 어댑터의 표현 용량이 커진다.

인자 값 'lora_alpha'는 어댑터 출력을 스케일하기 위해 사용한다. Lora_alpha / r 으로

어댑터의 영향이 커진다.

3.3.3. 레시피1. 파인튜닝 모델

모델에 데이터셋의 정보를 바탕으로 문제, 지문, 선지와 정답을 하나의 프롬프트로 가공 후 학습 데이터로 제공한다. 정답에는 별도의 해설 데이터를 제공하지 않고 선지 문자 하나만을 제공한다.

학습 프롬프트 생성 코드 예시:

```
system_message = (
    "You are a helpful AI assistant."
    "Please answer the user's questions correctly."
    "Look for the evidence in the text when answering."
    "Underlined replaced with highlights. Example: **Was Underline**"
    "Return only the letter corresponding to the best answer."
)

user_prompt = """
[QUESTION]
{question}

[PASSAGE]
{article}

[OPTIONS]
{options_block}

[ANSWER]
"""

def format_options(opts):
    letters = string.ascii_uppercase
    return "\n".join(f"{letters[i]}. {opt}" for i, opt in enumerate(opts))
```

```
def create_conversation(sample):
  opts = sample["options"]
  return {
    "messages": [
      {"role": "system", "content": system_message},
                                        {"role":      "user",      "content":
user_prompt.format(article=sample["article"]  ,  question=sample["question"],
options_block=format_options(opts))},
      {"role": "assistant", "content": sample["answer"]}
    ]
  }

train_dataset = raw_dataset.map(create_conversation)
```

생성된 학습 프롬프트 예시:

```
You are a helpful AI assistant.Please answer the user's questions
correctly.Look for the evidence in the text when answering.Underlined
replaced with highlights. Example: **Was Underline**Return only the letter
corresponding to the best answer.
[QUESTION]
Which word best fits in the blank?

[PASSAGE]
I gave him my _ .

[OPTIONS]
A. address
B. house
C. home
D. place

[ANSWER]
A
```

모델 사용 시에는 프롬프트를 입력 후 바로 다음 토큰 분포에서 선지 문자의 토큰 확률을

구한다.

```
with torch.no_grad():
    out = model(prompt_ids)
    next_logits = out.logits[:, -1, :].squeeze(0)


option_logits = next_logits[option_token_ids]
option_probs = torch.softmax(option_logits, dim=0)
probs = option_probs.detach().cpu().float().tolist()
```

3.3.4. 레시피2. 모델 데이터 생성

현재 데이터 셋에는 문제의 정답만 존재하고 해설 데이터는 없다. LLM은 Chain of Thought를 활용할 시 즉답에 비해 성능이 높아지는 경향이 있기 때문에 해설이 포함된 학습 데이터가 유리하다. 그러나 해설 데이터를 수작업으로 만들기는 비용이 크다. 이를 해결하기 위해 베이스 모델을 활용해 해설을 자동으로 생성한다.

STaR(Self-Taught Reasoner)는 베이스 모델의 추론을 통해 문제의 해설을 생성후 정답 여부를 필터링한다. 모델이 맞추지 못한 문제의 경우 모델에 정답 힌트를 주고 다시 해설을 생성하게 하는 합리화 과정을 통해 데이터 셋에 대한 해설을 다시 생성 한다. 이렇게 만들어진 해설 포함 데이터로 모델을 학습 시키고, 이 과정을 반복한다. [6]

본 과제에서는 STaR의 아이디어를 토대로 베이스 모델을 활용해 현재 데이터셋에서 해설 데이터를 생성한다. 틀린 문제에 대해 프롬프트 마지막에 정답 선지를 알려주는 단서를 추가 후 다시 생성을 유도해 합리화를 유도한다. 반복 절차는 적용하지 않고 최초 1회 수행한 결과를 데이터셋으로 활용한다. 학습은 전체 20,000개 데이터 중 16,000개를 사용한다.

초기 생성 시 모델의 출력을 안정화시키기 위해 few-shot 프롬프팅으로 예시를 주어 해설을 생성한다. 그 결과 생성된 학습에 사용되는 데이터는 다음과 같다.

```
You are a skilled English test-solving tutor.
In the passage, any text referred to as 'underlined' is shown in bold. Example: **Was Underline**
Read the problem and reason step by step.
Use no more than 5 steps, shorter is better.
```

max 50 characters per step, and write the final answer at the end.

[QUESTION]

According to the passage, _ help children most.

[PASSAGE]

Most people want their children to be successful in school and a parent's role in that success must be very important. Parents should help children to build their confidence and achievements. Parents should also play the role of a friend as well as a teacher in children's education.

{이하 생략}

[OPTIONS]

A. teachers

B. friends

C. parents

D. classmates

[ANSWER]

1. Keyword: help children most

2. Text: parents are key support

3. A: limited role

4. B, D: minor

answer is: C

모델 사용 시 정답을 파싱하기 위해 transformers의 StoppingCriteria를 활용해 모델 정지 조건을 설정한다.

```
class StopOnAnswerIs(StoppingCriteria):
          def __init__(self, tokenizer, pattern=r"answer\s*is\s*:?\s*$",
lookback_tokens=64):
        super().__init__()
        self.tokenizer = tokenizer
        self.regex = re.compile(pattern, re.IGNORECASE)
        self.lookback = lookback_tokens
```

```
            def __call__(self, input_ids: torch.LongTensor, scores:
torch.FloatTensor, **kwargs):
        tail = input_ids[0, -self.lookback:].tolist()
        text = self.tokenizer.decode(tail, skip_special_tokens=True)
        return bool(self.regex.search(text))
```

생성 시 마지막 64 토큰을 텍스트로 변환해 'answer is' 문자열을 정규식으로 검색해 해당 문자열이 있는 경우 생성을 정지한다.

그 후 정지 직후의 다음 토큰 분포로 선지의 확률을 구한다.

3.3.5. 레시피3. 외부 생성형 LLM API를 활용한 Knowledge Distillation

큰 규모의 Teacher 역할의 모델 응답 데이터를 바탕으로 작은 규모의 모델인 Student를 파인튜닝해 Teacher에 비해 빠르고 경제적이지만 기존 소형 모델에 비해 높은 성능을 지니는 모델을 만든다. Teacher 역할로 외부 API인 Gemini API를 통해 문제의 해설을 생성해 Student의 학습에 활용한다.

배치 API는 한번에 100개의 요청을 보낼 수 있기 때문에 보낼 요청을 100개 단위의 jsonl 파일로 나눈다.

```
all_reqs = []
for row in raw_dd["train"]:
    key = f"train:{row['example_id']}"
    req_obj = {
        "key": key,
        "request": {
            "contents": [{
                "role": "user",
                "parts": [{
                    "text": build_request_text(
                        question=row["question"],
                        passage=row["article"],
                        options=row["options"],
```

```
                    )
                }]
            }],
            "generation_config": {
                "temperature": model_temperature,
                "max_output_tokens": max_output_tokens,
            },
            "system_instruction": {
                "role": "system",
                "parts": [{"text": system_message}]
            }
        }
    }
    all_reqs.append(req_obj)

jsonl_paths = []

n_chunks = math.ceil(len(all_reqs) / 100)
for i in range(n_chunks):
  part = all_reqs[i*100:(i+1)*100]
  jsonl_path = os.path.join(BATCH_DIR, f"batch_{i+1:05d}.jsonl")
  with open(jsonl_path, "w", encoding="utf-8") as f:
      for req in part:
          f.write(json.dumps(req, ensure_ascii=False) + "\n")
  jsonl_paths.append(jsonl_path)

return jsonl_paths
```

제미니 api와 연결해 배치 파일을 읽어 요청을 보낸다.

```
client = genai.Client(api_key=GOOGLE_API_KEY)

uploaded = client.files.upload(
```

```
    file=jsonl_path,
    config=types.UploadFileConfig(
        display_name=os.path.basename(jsonl_path),
        mime_type="jsonl"
    ),
)
job = client.batches.create(
    model=model_name,
    src=uploaded.name,
    config={"display_name": display_name},
)
```

그 후 해당 배치 파일에 대한 응답을 받기 까지 폴링으로 상태를 확인한다. 상태 값이 완료가 되면 결과를 다운로드 받는다.

```
terminal = {
    "JOB_STATE_SUCCEEDED",
    "JOB_STATE_FAILED",
    "JOB_STATE_CANCELLED",
    "JOB_STATE_EXPIRED",
}
while True:
    job = client.batches.get(name=job_name)
    state = getattr(job.state, "name", str(job.state))
    print(f"[Batch] {job_name} state: {state}")
    if state in terminal:
        break
    time.sleep(60)
if getattr(job, "dest", None) and getattr(job.dest, "file_name", None):
    return client.files.download(file=job.dest.file_name)
```

생성된 모든 배치 파일에 대해 위 과정을 반복한다. 완료된 응답 중 정답인 응답만을 학습 데이터로 활용한다.

API 비용을 고려해 본문 길이가 짧은 CLOTH 문제 12000개를 외부 API인 Gemini 2.5 Flash-Lite로 요청한다. 그 중 정답인 요청 9489개를 선별해 그 중 7591개를 학습 데이터로 949개를 검증 데이터으로 활용한다.

학습에 사용되는 데이터의 예시는 다음과 같다.

```
You are a helpful assistant.
Provide reasoning only (no mention or hints about options).
At the very end, output exactly: 'answer is X' where X is one of A,B,C,D,E.
The phrase 'answer is' must appear once, in lowercase, and only on the final line.
Do not reveal or imply the answer prior to that final line.
[QUESTION]
Which word best fits in the blank?

[PASSAGE]
My _ , with his very American last name, had no trouble at all.

[OPTIONS]
A. son
B. husband
C. friend
D. daughter

[ANSWER]
The appositive “with his very American last name” marks a male referent and highlights a contrast that explains differential treatment at immigration. The construction “My \_\_\_, with his …, had …” most naturally denotes a close, primary companion in travel and paperwork contexts rather than a child or casual acquaintance, and the tone suggests an adult relationship central to the speaker.

answer is B
```

레시피 2번과 같이 모델 사용시 정답을 파싱하기 위해 모델 정지 조건을 설정해 생성 정지 직후의 다음 토큰 분포로 정답 확률을 구한다.

3.4. 여러 모델을 활용한 정확도를 높이는 연구

3.4.1. Ensemble

단일 모델은 주어진 데이터와 학습 과정에서 형성된 특정 패턴에만 집중하기 쉬우며, 그 결과 편향과 분산으로 인해 정답 예측에 한계가 있다. Ensemble은 이러한 한계를 보완하기 위해 여러 모델을 결합하는 방법이다. 이렇게 하면 각 모델의 강점은 살리고 약점은 상호 보완할 수 있다. 대표 방식으로 Bagging, Boosting, Stacking, Voting이 있으며, 본 연구는 Voting을 중심으로 논의한다. [7]

이는 마치 시험에서 여러 학생의 답안을 모아 보는 것과 같다. 한 학생은 특정 과목에 강하지만 다른 과목에서는 약할 수 있으며, 반대로 또 다른 학생은 다른 영역에서 강점을 가진다. 이러한 의견을 종합하면, 단일 학생의 답안만 참고하는 것보다 정답에 근접할 가능성이 높아진다.

그러나, 학생들의 실력이 항상 동일하지는 않듯이, 모델도 항상 같은 성능을 보장하지는 않는다. 따라서 모델들의 성능에 따른 가중치를 각각 부여하는 접근이 필요하다. 다음 문단에서, 이 가중치를 어떻게 부여하고 반영할지에 대해 다루어 보도록 한다.

3.4.2. LOP(Logarithmic Opinion Pool) 기반 확률 결합 및 예측

Ensemble 기법을 쓰기 위해서는 각 모델이 도출한 결과를 하나로 합쳐야 한다. 여기서 사용할 수 있는 기법에는 여러 방법이 있는데, 본 연구에서는 'LOP' 라는 방법을 사용한다. 이는 각 분포의 로그를 가중합해 지수화하는, 가중 기하평균 형태의 결합이다. [8] 이 기법을 쓰려면 가중치를 구해야 한다. 여기서는 조건부로 가중하는, 게이팅 방법을 쓰기로 한다.

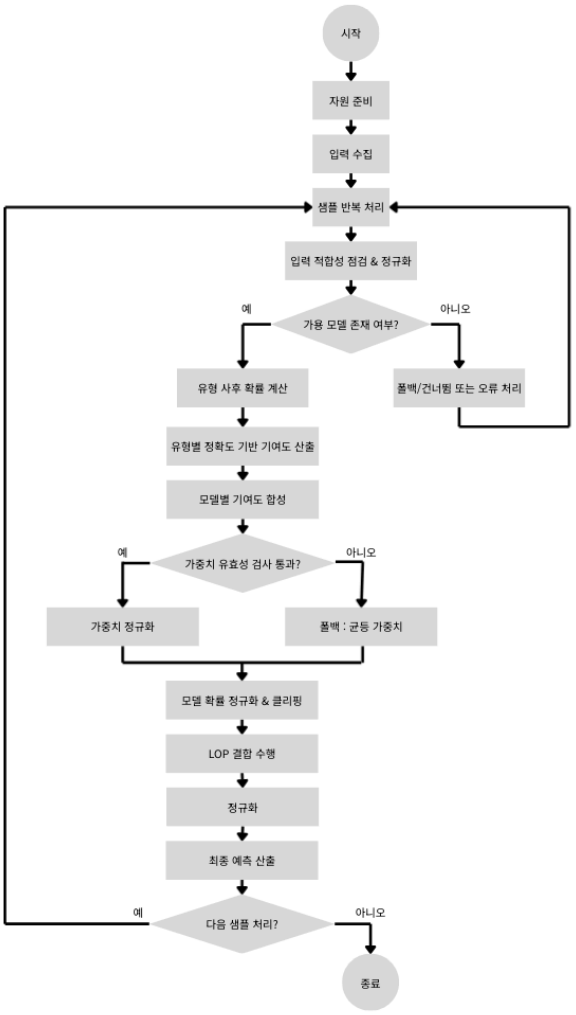
여기서는 앞선 비유에서 학생들마다 강점을 보이는 과목이 다른 것처럼, 모델마다 문제 유형에 따라 성능이 다르다는 점에 주목한다. 따라서 문제의 분류를 게이팅 조건에 사용하기로 했다. 문제 유형 분류를 위해 다음과 같이 여섯 가지 유형으로 분류하였으며, 이에 대한 세부 분류 기준은 이후 설명한다:

```
{cloth, korean, race_high_long, race_high_short, race_middle_long
,race_middle_short}
```

유형 분류기는 BERT 계열 두 모델(**microsoft/deberta-v3-base**, **skt/kobert-base-v1**)을 환경에 맞게 학습하여 사용한다. 영어 전용/한국어 포함 문항을 각각 담당하도록 설계했고, (i) KoBERT 단독, (ii) Marian 번역 후 DeBERTa 단독, (iii) 두 모델 병용을 비교한 결과, 병용이 가장 높은 정확도를 보여 해당 방식을 채택한다. 분류기의 출력은 유형별 로짓으로 제공되며, 소프트맥스로 정규화해 유형 확률로 사용한다.
아래의 코드가 이 부분을 담당한다.

```
type_names = list(type_logits.keys())
q = _softmax(np.array([type_logits[t] for t in type_names],
 dtype=np.float64))
```

문제 유형을 분류했다면 이것으로 적절한 가중치를 구해야 한다. 여기서는 사전에 테스트 셋 일부로 확인한, 모델별 각 유형의 문제에서 달성한 정확도를 기준으로 삼는다. 이때 사용한 알고리즘을 아래에서 설명한다.



1) 조건부 가중치(게이팅) 산출

각 기본 모델 i와 문항 유형 t에 대한 사전 측정 정확도를 활용해, 해당 문항에 사용할 가중치를 계산한다. 문항의 보기 수를 K라 하면 무작위 정답 확률은 $\frac{1}{K}$이므로, 모델의 “의미 있는 우위”를 정확도에서 $\frac{1}{K}$을 뺀 값으로 정의한다. 이 값이 0보다 작은 경우는 무작위보다 열등하므로 기여를 0으로 둘 수 있다. 우위는 지수화로 반영한다. 여기서 eta는 우위의 반영 강도를 조절하는 하이퍼파라미터다. 유형 분류기의 불확실성을 반영하기 위해, 유형 확률으로 가중 평균하여 최종 기여도를 합성한다.

이 부분까지의 코드는 아래와 같다:

```
base = _chance_base(K) # 무작위 정답 확률: 1 / K
 raw = np.zeros(len(available_ids), dtype=np.float64)
 for ti, t in enumerate(type_names):
     row = acc_table.get(t, {})
     if not row:
         continue
     for mi, mid in enumerate(available_ids):
         a = row.get(mid)
         if a is None:
             continue
         a = max(min(_to_float01(a), 1.0 - acc_eps), acc_eps)
         if drop_below_chance and a < base:
             contrib = 0.0
         else:
             contrib = math.exp(eta * (a - base))
         raw[mi] += q[ti] * contrib
```

마지막으로 가용 모델 집합에 대해서 정규화하여 모델 가중치를 얻는다.

정규화 부분의 코드는 아래와 같다:

```
if not np.isfinite(raw).all() or raw.sum() <= 0:
     return {mid: 1.0 / len(available_ids) for mid in
         available_ids}
raw /= raw.sum()
return {mid: float(w) for mid, w in zip(available_ids, raw)}
```

이 코드에서는 안정성을 위해 가중치를 구하기 전, 수치에 NaN이나 무한대 값은 없는지, 합이 0 이하는 아닌지를 검증한 후 최종적인 가중치를 반환한다.

2) LOP로 확률 결합

각 모델이 산출한 입력받은 문제에 대한 선지 분포를 입력으로 받아, LOP로 최종

분포를 계산한다.

모델이 logit을 출력하면 해당 값들을 소프트맥스한다. 일부 모델이 실패하거나 누락되면 제외하고 남은 모델 가중치를 재정규화한다. LOP의 구현은 아래와 같다.

```
logp = np.zeros(K, dtype=np.float64)

for mid, w in W.items():
      p = np.asarray(model_probs[mid], dtype=np.float64)
      p = np.clip(p, 1e-12, 1.0)
      logp += w * np.log(p)
ens_probs = np.exp(logp - _logsumexp(logp))
```

위 코드에서, 최종적으로 구한 ens_probs 중 가장 높은 값이 최종 예측 정답이 된다.

4. 연구 결과 분석 및 평가

학습한 개별 모델의 베이스 모델 대비 성능 향상을 비교한다. 이 후 모델들을 앙상블한 성능을 개별 모델들과 비교한다.

0. 분석을 위한 테스트 데이터셋 선정

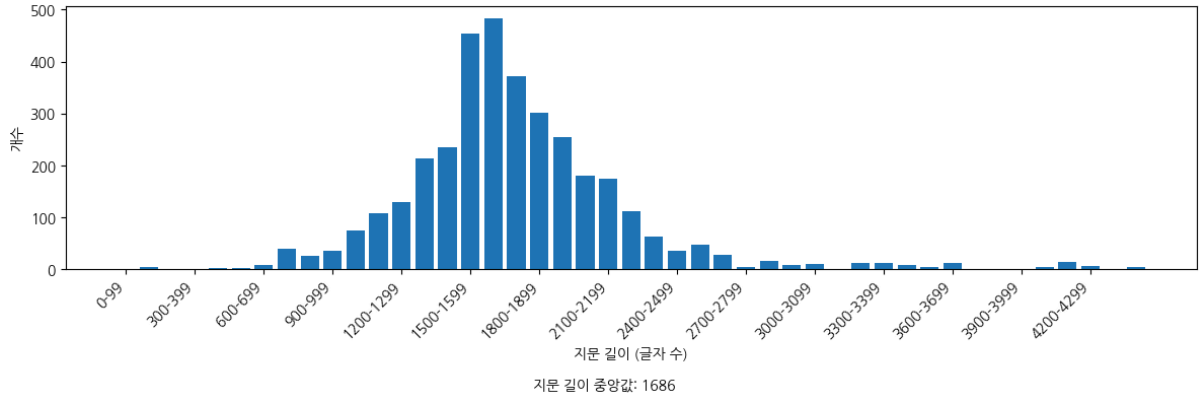
본 연구에서는 RACE, CLOTH, 한국 검정고시/수능 문제 3개의 데이터셋을 RACE dataset의 형태로 가공하여 활용하였다. 테스트를 위한 데이터셋 역시, 해당 데이터셋을 활용한다.

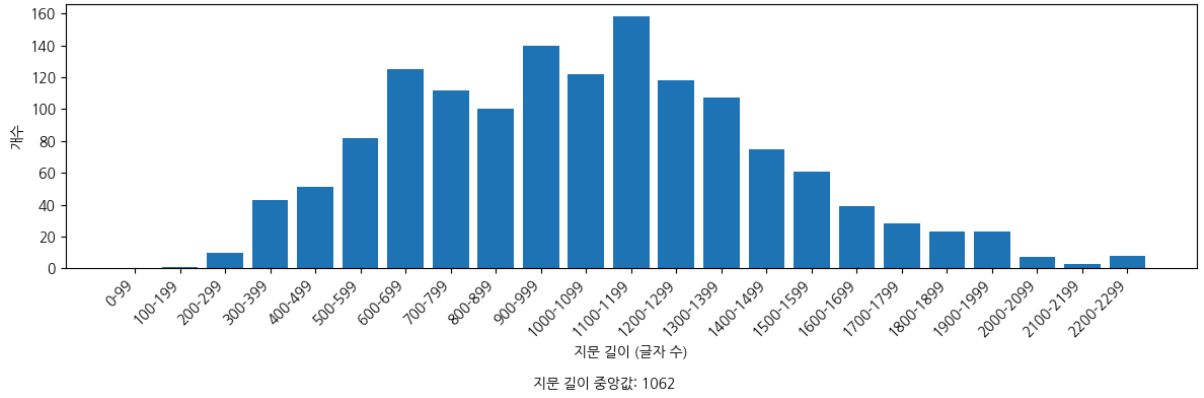
학습 및 평가 데이터셋 제작에 있어 한국 검정고시/수능 의 데이터셋의 양이 절대적으로 부족하다. 학습 및 평가 과정에 있어 데이터셋의 과편향이 일어날 가능성이 있기 때문에, 이를 고려하여 데이터셋을 제작했다. 학습 및 평가 과정에 있어 최대한 불공평함이 일어나지 않도록, 학습 및 평가 시 각 데이터셋의 비율을 중요하게 생각하였다.

최초 한국 검정고시/수능 문제를 활용해 데이터셋을 만들 때 949개의 문제가 train으로, 124개의 문제가 validation으로, 124개의 문제가 test로 분리되었기 때문에 해당 문제는 최대한 전부 활용할 수 있도록 비율을 조정한다. 최종적으로 제작한 데이터셋은 학습용으로 최대 20000개를 활용하기로 결정하였고, validation 데이터는 학습용 데이터의 양의 기준에 따라 조절하여 최대 2000개를 활용한다.

학습용 데이터는 최대 20000개 기준 각각 RACE 고등문제 10500개, RACE 중등 3500개, CLOTH 5100개, 한국 문제 900개로 선정하여 제작하며, validation 용 데이터는 최대 2000개 기준 RACE 고등 1050개, RACE 중등 350개, CLOTH 500개, 한국 문제 100개로 총 2000개의 데이터를 선정하여 제작한다.

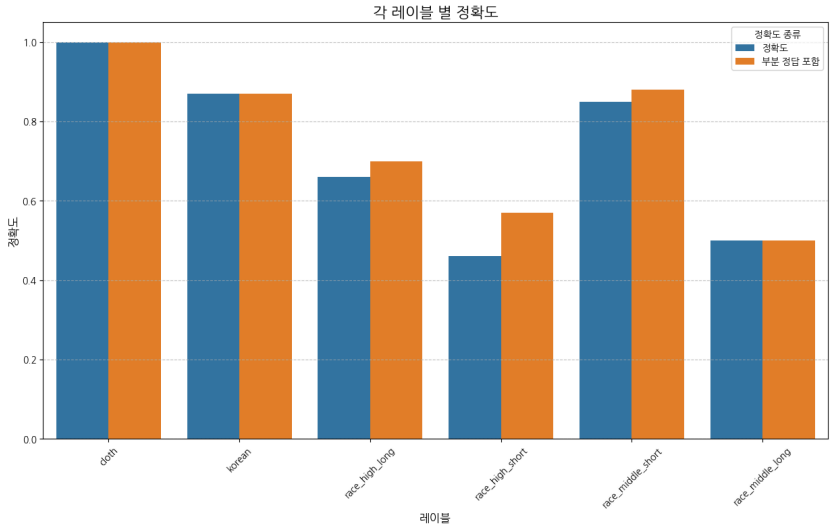
테스트 데이터의 경우 최종 2000개를 추출하여 사용한다. 모델이 문제를 푸는데에 있어 지문의 길이가 길면 길수록 정답률이 떨어지는 경향을 보였고 이에 대한 정확한 파악을 위해 RACE dataset에서 추출한 데이터는 문제 지문의 길이를 나누어 long과 short로 구분하여 테스트를 진행할 수 있도록 한다. long과 short를 나누는 기준은 아래 그래프에 따라 지문 길이의 중위값을 파악하여 중학 문제의 경우 1000, 고등 문제의 경우 1700을 기준으로 기준치보다 짧은 경우 short, 긴 경우 long으로 분리한다.





또한, 평가 과정에 있어 평가 데이터셋 역시 과편향의 우려가 있을 수 있어, 추출한 2000개의 데이터를 절반으로 나누어 eval_1, eval_2로 나누어 평가를 진행한다. 개별 모델의 성능은 eval_2를 통해 평가하고, 이 결과를 바탕으로 모델간 가중치 계산에 활용한다. 앙상블 최종 성능은 eval_1을 통해 테스트한다.

1. 분류 모델 성능 테스트

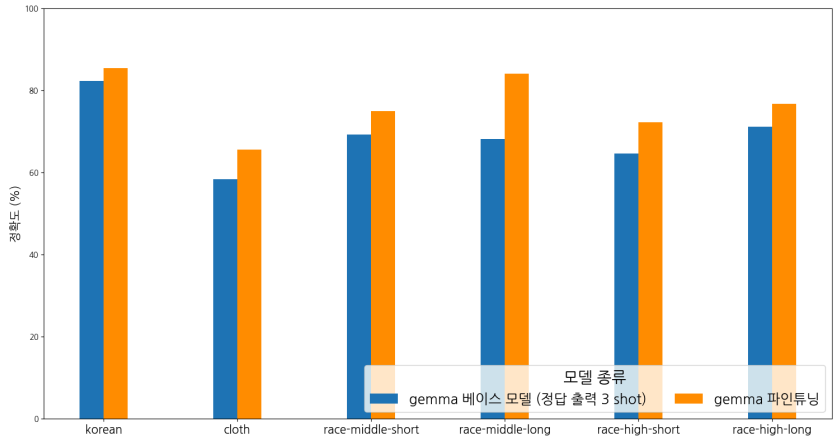


학습 데이터에 문제 유형을 각각 cloth, korean, race_high_long, race_high_short, race_middle_long, race_middle_short로 레이블링하여 분류 모델을 학습시켰다. 그리고 이 분류가 모두 포함된 혼합 데이터셋으로 분류 성능 테스트를 진행하였다. 여기서는 600개의 데이터셋을 사용했으며, 이 중 434개를 올바르게 분류해 약 72.33%의 정확도를 기록했다. 만약 기준을 좀 더 완화해, race라는 대분류와 길이(long, short)까지는 맞은 것을 부분 정답으로 인정한다면 정확도는 약 75.33%로 상승했다. 사용한 분류 모델(KoBERT, DeBERTa)별로 보면, KoBERT는 약 89.60%, DeBERT는 약 67.79%, 부분 정답 포함 시 약 71.58%의 정확도를 기록했다.

2. 개별 문제 풀이 모델 성능 테스트

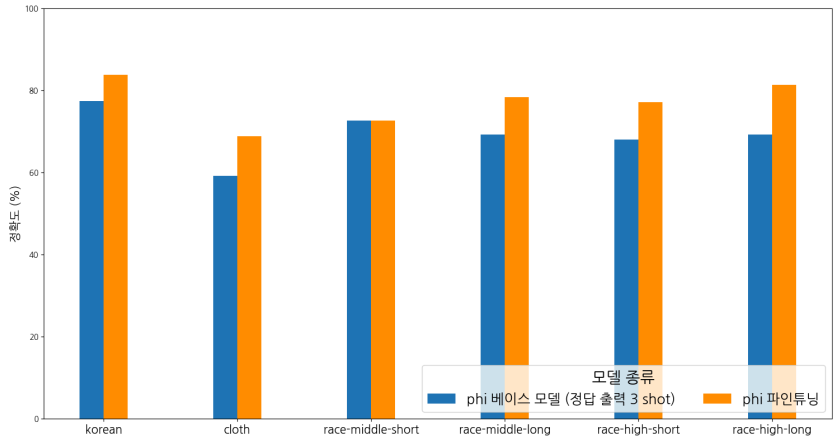
a. 레시피 1

Gemma



문제와 정답만을 프롬프팅하여 학습한 후, 기존 베이스 모델 (gemma 3 4b it)과 eval_2 데이터셋으로 비교 평가를 진행했다. 그 결과 모든 문제 유형에 대해 정확도가 소폭 향상되었으며. 최종 1002개의 문항 중 베이스 모델은 668문항(66.67%), 파인튜닝 모델은 741문항(73.95%)를 맞춰 전체 정확도가 7.28% 상승하였다.

Phi

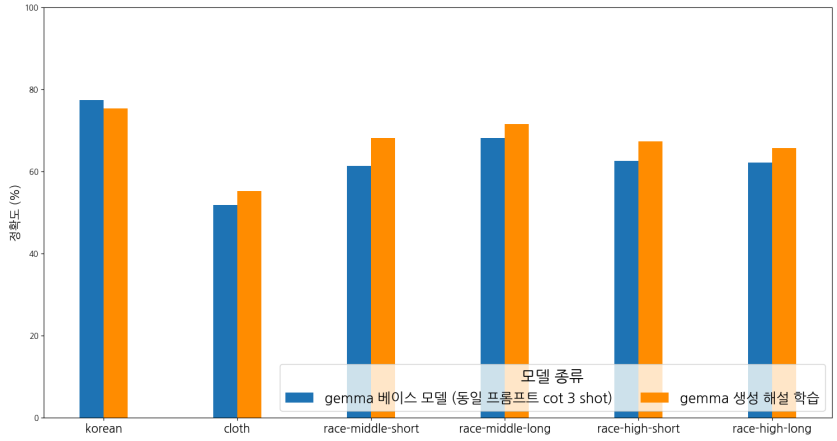


문제와 정답만을 프롬프팅하여 학습한 후, 기존 베이스 모델 (Phi-4-mini-instruct)과 eval_2 데이터셋으로 비교 평가를 진행했다. 그 결과 race 중학 짧은 지문 문제를 제외한 모든 문제 유형에 대해 정확도가 소폭

향상되었으며. 최종 1002개의 문항 중 베이스 모델은 675문항(67.37%), 파인튜닝 모델은 766문항(76.45%)를 맞춰 전체 정확도가 9.08% 상승하였다.

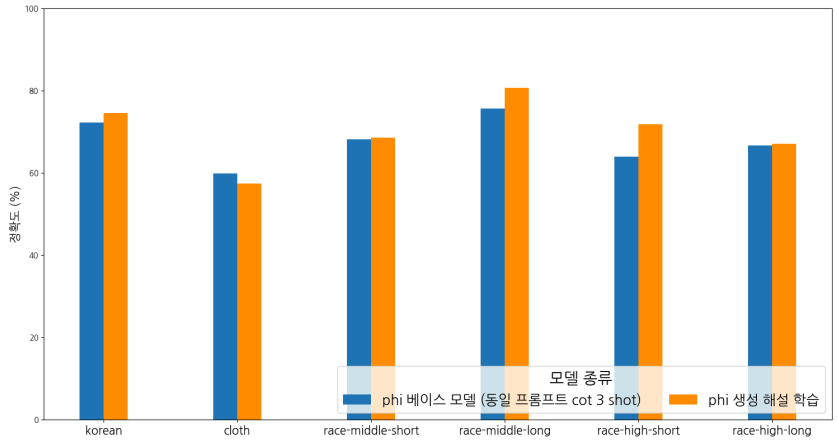
b. 레시피 2

Gemma



데이터셋에 대해 해설을 생성하고 이를 베이스 모델에 학습한 결과 모델이 정답을 낸 문제들 중 베이스 모델은 61.28%, 생성 해설 학습 모델은 65.00% 전체 정확도를 기록하였다.

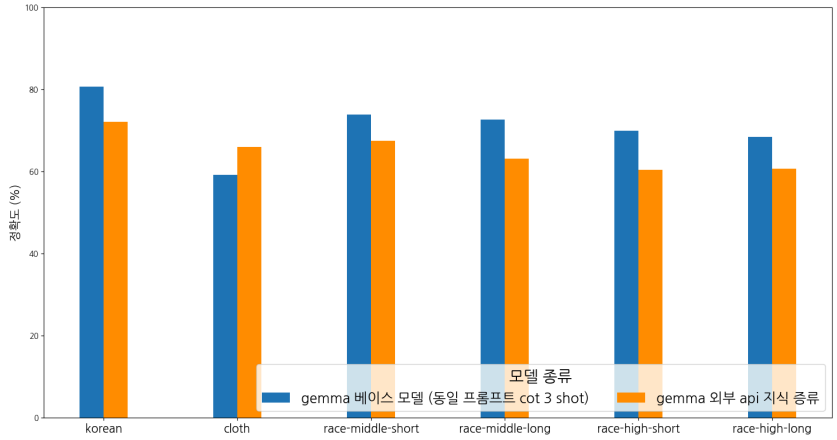
Phi



Phi 또한 마찬가지로 해설을 생성하고 이를 베이스 모델에 학습한 결과 모델이 정답을 낸 문제들 중 베이스 모델은 65.43%, 해설 학습 모델은 67.73% 전체 정확도를 기록하였다.

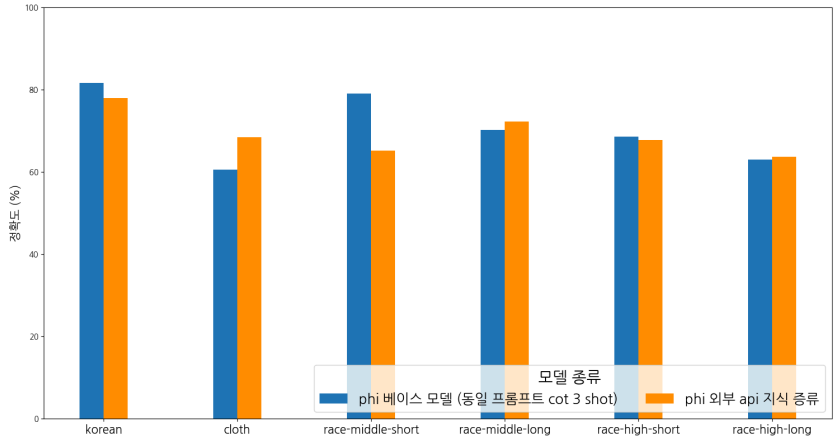
c. 레시피 3

Gemma



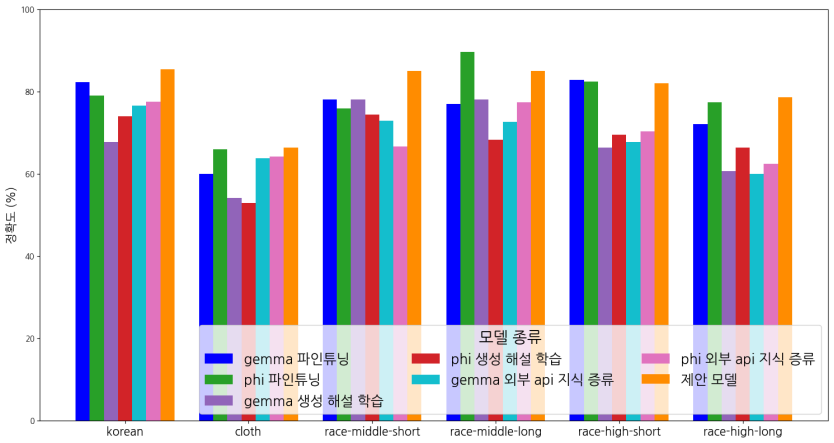
평가 결과 베이스 모델 few shot 프롬프팅에 비해 cloth를 제외한 모든 분야에 정확도가 하락하였다. 전체 정확도는 베이스 모델의 68.26% 에서 63.44%으로 하락하였지만 학습한 항목인 cloth 항목의 경우 베이스 모델의 정확도 59.24%에서 65.97%로 6.73% 상승하였다.

Phi



Phi역시 대다수의 항목에서 정확도가 떨어졌으나 학습한 항목인 cloth 항목의 경우 베이스 모델의 정확도 60.50%에서 68.49%로 7.99% 상승하였다.

3. 앙상블을 활용한 최종 모델 성능 평가



앙상블 모델의 경우 과편향을 막기 위해 개별 모델 테스트에 사용된 데이터셋과 다른 데이터셋을 사용해 모든 모델을 테스트한다. 앙상블의 특성상 최종 제안 모델의 성능은 앙상블에 사용되는 모델 중 가장 좋은 모델의 정확도에 수렴할 것으로 예측하였고, 실제 측정 결과 최종 제안 모델의 성능이 타 모델에 비해 근소하게 높거나 가장 좋은 모델의 정확도에 근접하는 모습을 보였다. 일부 분야에서 가장 좋은 모델보다 근소하게 낮은 정확도를 기록했으나, 이는 모델들의 정확도 편차가 커서 발생한 것으로 차후 정확도가 떨어지는 모델에 대해 비중을 더 낮추거나 배제를 하면 정확도가 더 올라갈 것으로 기대된다.

5. 결론 및 향후 연구 방향

모델 개발 및 정확도를 올리기 위한 다양한 방법을 적용하였고, 베이스 모델에 비해 문제를 좀 더 잘 풀어내는 모델을 개발하여 문제 풀이에 접목할 수 있었다.

향후에는 모델의 정확도를 더 올리기 위한 방법들을 추가로 찾아보고, 이를 실제로 접목하여 정확도 향상을 얼마나 이뤄낼 수 있는가를 연구하는 방향으로 더 진행할 수 있겠다. 본 연구에서는 3가지 기법을 접목하여 모델들을 개량 시도하였으나, 이 외에도 성능을 올릴 수 있는 방법이 있다면 이를 접목할 수 있을 것이다.

또한 문제 분류에 있어서도 더 정확도가 높은 분류 방법을 연구하여 발전 시키는 방향으로도 연구를 진행할 수 있을 것이다. 본 연구에서는 BERT 계열 모델을 사용하여 문제를 분류하였는데, 다른 계열의 모델로도 시도하여 정확도를 높이고자 하는 시도를 해볼

수 있을 것이다. 방법적인 측면에서도 LOP 이외의 다른 방법을 시도하여 정확도 면에서의 성능 향상을 기대할 수 있을 듯하다.

6. 참고 문헌

[1] G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy, “RACE: Large-scale ReAding Comprehension Dataset From Examinations,” *arXiv preprint* arXiv:1704.04683, 2017.

[2] Q. Xie, G. Lai, Z. Dai, and E. Hovy, “Large-scale Cloze Test Dataset Created by Teachers,” *arXiv preprint* arXiv:1711.03225, Aug. 2018. (Presented at EMNLP 2018)

[3] Gemma Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, M. Rivière *et al.*, “Gemma 3 Technical Report,” *arXiv preprint* arXiv:2503.19786, Mar. 2025.

[4] Microsoft, A. Abouelenin, A. Ashfaq, A. Atkinson, H. Awadalla, N. Bach, J. Bao, A. Benhaim, M. Cai, V. Chaudhary *et al.*, “Phi-4-Mini Technical Report: Compact yet Powerful Multimodal Language Models via Mixture-of-LoRAs,” *arXiv preprint* arXiv:2503.01743, Mar. 2025.

[5] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “LoRA: Low-Rank Adaptation of Large Language Models,” *International Conference on Learning Representations (ICLR)*, 2022.

[6] E. Zelikman, Y. Wu, J. Mu, and N. D. Goodman, “STaR: Bootstrapping Reasoning With Reasoning,” *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 15476-15488, 2022.

[7] T. G. Dietterich, “Ensemble Methods in Machine Learning,” in *Multiple Classifier Systems*, Berlin, Heidelberg: Springer, 2000, pp. 1-15. doi: 10.1007/3-540-45014-9_1.

[8] T. Heskes, “Selecting Weighting Factors in Logarithmic Opinion Pools,” *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 10, pp. 267-273, 1997.