

</

sLLM 학습하기

/>



LLM을 활용한 실전 AI 애플리케이션
개발
발표: 이재원

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</INTRO

- 자연어 요청으로부터 적합한 SQL을 생성하는 Text2SQL(NL2SQL) sLLM을 만들어 봅니다.
- 데이터 인력이 마케팅, 운영, 사업 등 알고 싶은 정보를 추출해 전달하는 방식으로 데이터를 활용 합니다.
- SQL 생성을 LLM이 보조할 수 있다면, 데이터 인력의 생산성을 높일 수 있습니다.
또한 현업 구성원의 SQL 진입 장벽을 낮출 수 있습니다.

</INTRO

- 자연어 요청으로부터 적합한 SQL을 생성하는 Text2SQL(NL2SQL) sLLM을 만들어 봅니다.

=> chatGPT한테 테이블을 간략히 설명하고 원하는 쿼리를 부탁하면 정말 잘해준다!

=> 게다가 QueryDSL 같은 라이브러리에 종속된 코드베이스 쿼리도 아주 잘 작성해준다!

=> RDB(Relational Database)뿐만 아니라, MongoDB 같은 NoSQL도 정말 잘 짜준다!

=> 하지만 가끔 쿼리를 엉망으로 짜는 경우가 있으므로 꼭 쿼리 성능을 확인한다.

</INTRO

- 데이터 인력이 마케팅, 운영, 사업 등 알고 싶은 정보를 추출해 전달하는 방식으로 데이터를 활용 합니다.

=> 큰 회사가 아니라 이런 업무도 개발자의 역할인게 현실(...)

=> 실시간 구독자 파악 등, 백 오피스에 필요한 정보를 얻는 API 모두 개발자가 작성

=> (개인적 궁금증) 민감한 정보를 쿼리하지 못하게 어떻게 막을 수 있을까요?

=> (개인적 궁금증) 알고 싶은 정보를 (개발자 없이) 꺼내서 보고 싶은 정도로 빈도가 잦을까요?

</INTRO

- SQL 생성을 LLM이 보조할 수 있다면, 데이터 입력의 생산성을 높일 수 있습니다.
또한 현업 구성원의 SQL 진입 장벽을 낮출 수 있습니다.

=> 저희 팀은 Cursor(IDE), Github copilot 을 적극 활용하기 때문에 모두 긍정적인 반응입니다.

=> (걱정되는 부분) 그래도 이게 정말 잘 만들어 준 쿼리인지 확인해야 합니다.

=> (걱정되는 부분) 보통, 대개 잘 들어맞으니까 관련된 공부를 하지 않게 된다. (나태지옥💀)

</GOAL

- Text2SQL sLLM의 미세 조정에 사용할 수 있는 데이터 셋을 살펴봅니다.
- GPT-4 모델을 활용해 SQL이 잘 생성됐는지 확인 합니다.
- 준비된 학습 데이터로 sLLM의 미세 조정을 수행 합니다.

</Text2SQL

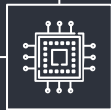
필요한 2가지 데이터

Text2SQL



데이터베이스 정보

table, column



요청사항

request, question

</WikiSQL vs Spider



WikiSQL



- 테이블의 이름
- 컬럼 이름 (header)
- 컬럼 형식 (types)
- 요청 사항
- 정답 SQL 데이터

</WikiSQL vs Spider



WikiSQL



- 테이블의 이름
- 컬럼 이름 (header)
- 컬럼 형식 (types)
- 요청 사항
- 정답 SQL 데이터

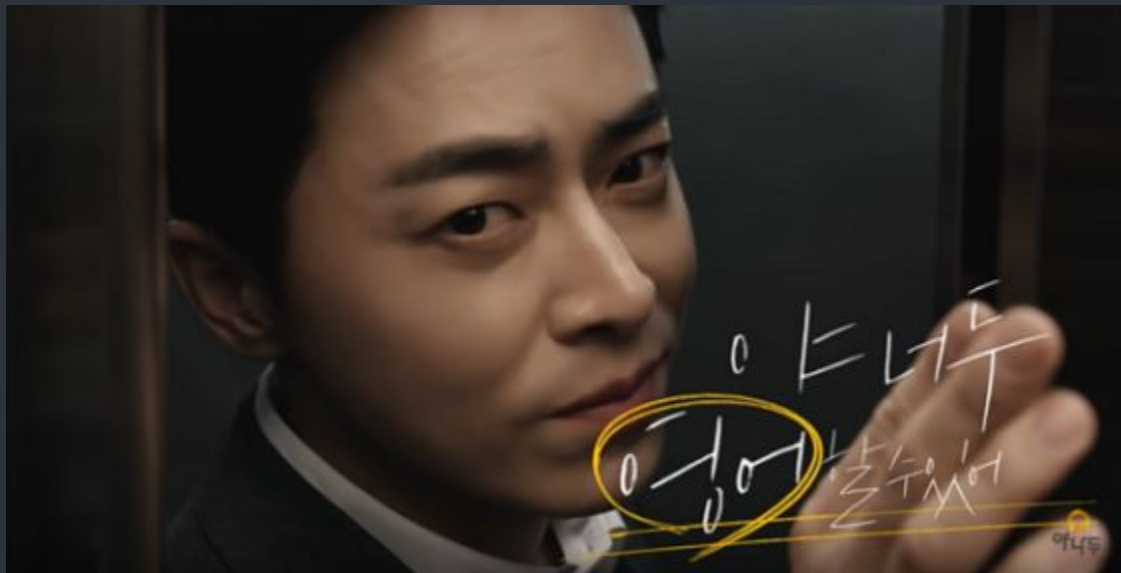
</WikiSQL vs Spider



Spider


- 좀 더 현실적인 문제 해결을 위해
ORDER BY, GROUP BY, HAVING, JOIN 등 포함


</WikiSQL vs Spider




하지만 두 데이터셋 모두 요청사항이 영어로 되어 있어 한국어 실습을 위한 데이터셋으로는 적합하지 않습니다.

</NL2SQL

 AI 데이터 찾기 AI 허브소개 참여하기 커뮤니티 AI 개발자원 고객지원 마이페이지 로그아웃

데이터 찾기  | AI 데이터 찾기 > 데이터 찾기



#자연어 기반 질의 생성

#NL2SQL

#자연어

NEW

자연어 기반 질의(NL2SQL) 검색 생성 데이터

분야


한국어


유형

텍스트

구축년도 : 2022 갱신년월 : 2024-02 조회수 : 14,920 다운로드 : 678 용량 : 1.08 GB

다운로드

샘플 데이터 

관심데이터 등록  86

소개

파일 목록 (API 다운로드)

※ 내국인만 데이터 신청이 가능합니다.

문의하기

목록

데이터 개요

▼

메타데이터 구조표

▼

데이터 통계

▼

*교재에 소개된 NL2SQL 데이터셋을 다시 내려받을 수 있게 되었습니다.

</ko_text2sql

Datasets: shangilar/ko_text2sql like 6

Modalities: Text Formats: csv Size: 10K - 100K Libraries: Datasets pandas Croissant +1

Dataset card Viewer Files and versions Community 1

Dataset Viewer Auto-converted to Parquet API Embed Full Screen Viewer

Subset (2) clean 28.9k rows Split (2) train 28.8k rows

Search this dataset SQL Console

db_id	context	question	answer
int64	string · lengths	string · lengths	string · lengths
1	98 1.26k	8 203	19 538
1	CREATE TABLE players (player_id INT PRIMARY KEY AUTO_INCREMENT,...	모든 플레이어 정보를 조회해 줘	SELECT * FROM players;
1	CREATE TABLE players (player_id INT PRIMARY KEY AUTO_INCREMENT,...	모든 사용자의 아이디와 이메일을 보여줘	SELECT player_id, email FROM players;
1	CREATE TABLE players (player_id INT PRIMARY KEY AUTO_INCREMENT,...	유저네임이 'archer'로 끝나는 플레이어들을 최신 가입 순으로 나열해줘.	SELECT username FROM players WHERE username LIKE 'archer' ORDER BY...
1	CREATE TABLE players (player_id INT PRIMARY KEY AUTO_INCREMENT,...	password_hash가 'abc123'인 사용자가 마지막 으로 로그인한 시점을 알려줘	SELECT last_login FROM players WHERE password_hash = 'abc123';
1	CREATE TABLE players (player_id INT PRIMARY KEY AUTO_INCREMENT,...	이메일 주소가 'gmail.com'으로 끝나는 사용자 수를 알려줘	SELECT COUNT(*) FROM players WHERE email LIKE '%gmail.com';
1	CREATE TABLE players (player_id INT PRIMARY KEY AUTO_INCREMENT,...	이메일(email) 주소에 '@' 기호 앞부분만 조회해 줘.	SELECT SUBSTRING_INDEX(email, '@', 1) FROM players;

< Previous 1 2 3 ... 288 Next >

*대신, 저자는 허깅페이스에 실습 데이터셋으로 진행 합니다.

</Performance Evaluation

Type	Description
EM [Exact-set-Match accuracy]	✓ 생성한 SQL 문자열이 동일한지 확인 ✗ 의미가 달라도 결과값이 같을 수 있다.
EX [EXecution accuracy]	✓ SQL 쿼리를 수행해 정답과 일치하는지 확인 ✗ 추가적인 데이터베이스 필요

*생성 결과를 정량적으로 평가하기 어려운 작업이 많고,
사람이 직접 평가를 수행하기에는 시간과 비용이 많이 들어 개발 사이클을 늦춘다는 한계가
있습니다.

*이번 실습은 LLM이 생성한 SQL이 잘 해결하는지 GPT-4를 통해 확인 합니다.

</Prompt Example

✓ 예제 6.2. SQL 프롬프트

```
[ ] def make_prompt(ddl, question, query=''):
    prompt = f"""당신은 SQL을 생성하는 SQL 봇입니다. DDL의 테이블을 활용한 Question을 해결할 수 있는 SQL 쿼리를 생성하세요.

    ### DDL:
    {ddl}

    ### Question:
    {question}

    ### SQL:
    {query}"""
    return prompt
```

*LLM의 경우 학습에 사용한 프롬프트 형식을, 추론할 때도 동일하게 사용해야 결과가 좋기 때문에,
이번 절에서 준비한 프롬프트 형식은 모델을 미세 조정할 때도 동일하게 적용합니다.

</Training Data Example

```
[ ] def make_prompt(ddl, question, query=''):
    prompt = f"""당신은 SQL을 생성하는 SQL 봇입니다. DDL의 테이블을 활용한 Question을 해결할 수 있는 SQL 쿼리를 생성하세요.

    ### DDL:
    CREATE TABLE messages {
        "message_id" SERIAL PRIMARY KEY,
        "conversation_id" INT NOT NULL,
        "sender_id" INT NOT NULL,
        "content" TEXT,
        "timestamp" TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
        "read" BOOLEAN DEFAULT FALSE,
        FOREIGN KEY ("conversation_id") REFERENCES conversations("conversation_id"),
        FOREIGN KEY ("sender_id") REFERENCES users("user_id")
    };

    ### Question:
    messages 테이블에서 모든 데이터 조회해 줘

    ### SQL:
    SELECT * FROM messages;"""
    return prompt
```


</Generation Prompt Example

```
[ ] def make_prompt(ddl, question, query=''):
    prompt = f"""당신은 SQL을 생성하는 SQL 봇입니다. DDL의 테이블을 활용한 Question을 해결할 수 있는 SQL 쿼리를 생성하세요.


    ### DDL:
    CREATE TABLE messages {
      "message_id" SERIAL PRIMARY KEY,
      "conversation_id" INT NOT NULL,
      "sender_id" INT NOT NULL,
      "content" TEXT,
      "timestamp" TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
      "read" BOOLEAN DEFAULT FALSE,
      FOREIGN KEY ("conversation_id") REFERENCES converstaions("converstaion_id"),
      FOREIGN KEY ("sender_id") REFERENCES users("user_id")
    };

    ### Question:
    messages 테이블에서 모든 데이터 조회해 줘

    ### SQL:
    """""
```

</Preparation of GPT-4 Evaluation Prompts and Code

- GPT-4를 사용해 평가를 수행한다면 반복적으로 GPT-4 API에 API 요청을 보내야 합니다.
 - for 문으로 반복 수행하면 데이터 양에 따라서 시간이 오래 걸릴 수 있습니다.
 - 따라서 **OpenAI**가 제공하는 코드(Cookbook)를 활용하면 **rate limit**을 지키면서, **비동기적으로 요청을 처리**할 수 있습니다.
- 에러가 발생하거나, 요청 제한에 걸려 실패하면 다시 요청을 보내서 결과의 누락도 방지합니다.



```
def make_requests_for_gpt_evaluation(df, filename, dir='requests'):
    if not Path(dir).exists():
        Path(dir).mkdir(parents=True)
    prompts = []
    for idx, row in df.iterrows():
        prompts.append("""Based on below DDL and Question, evaluate gen_sql can resolve Question. If gen_sql and gt_sql
do equal job, return "yes" else return "no". Output JSON Format: {"resolve_yn": ""}""")

    DDL: {row['context']}
    Question: {row['question']}
    gt_sql: {row['answer']}
    gen_sql: {row['gen_sql']}""")
    )
```

*입력한 데이터프레임을 순회하면서, 평가에 사용할 프롬프트를 생성하고 jsonl 파일에 기록.

*LLM이 생성한 gen_sql, 정답 SQL gt_sql이 동일한 기능을 하는지 resolve_yn 필드에 반환.



```
jobs = [{"model": "gpt-4-turbo-preview", "response_format" : { "type": "json_object" }, "messages": [{"role":  
"system", "content": prompt}]} for prompt in prompts]  
with open(Path(dir, filename), "w") as f:  
    for job in jobs:  
        json_string = json.dumps(job)  
        f.write(json_string + "\n")
```

- *생성한 평가 프롬프트는 사용할 모델 이름을 지정하여, 요청 작업 변수에 저장.
- *지정한 디렉토리 경로와 파일 이름으로 요청 정보를 jsonl 파일 형태로 저장.



```
import os
os.environ["OPENAI_API_KEY"] = "자신의 OpenAI API 키 입력"

python api_request_parallel_processor.py \
  --requests_filepath {요청 파일 경로} \
  --save_filepath {생성할 결과 파일 경로} \
  --request_url https://api.openai.com/v1/chat/completions \
  --max_requests_per_minute 300 \
  --max_tokens_per_minute 100000 \
  --token_encoding_name cl100k_base \
  --max_attempts 5 \
  --logging_level 20
```

*OpenAI의 Cookbook 비동기 요청 코드를 실행.

*요청 제한 값이 있지만, 최댓값으로 설정할 경우 초과 에러가 빈번하게 발생, 적절히 조정.

*cl100k_base: 토큰나이저에 사용되는 하나의 토큰화 방식

*logging_level: info에 해당하는 로그 레벨



```
def change_jsonl_to_csv(input_file, output_file, prompt_column="prompt", response_column="response"):
    prompts = []
    responses = []
    with open(input_file, 'r') as json_file:
        for data in json_file:
            prompts.append(json.loads(data)[0]['messages'][0]['content'])
            responses.append(json.loads(data)[1]['choices'][0]['message']['content'])

    df = pd.DataFrame({prompt_column: prompts, response_column: responses})
    df.to_csv(output_file, index=False)
    return df
```

*결과파일을 불러와 프롬프트와 판단 결과 데이터를 각각 **prompts, responses** 변수에 저장.

*판다스 데이터 조작 라이브러리를 이용하고, **to_csv()** 메서드를 사용해 **csv** 파일로 저장.



```
from datasets import load_dataset
# 데이터셋 불러오기
df = load_dataset("shangrilar/ko_text2sql", "origin")['test']
df = df.to_pandas()
for idx, row in df.iterrows():
    prompt = make_prompt(row['context'], row['question'])
    df.loc[idx, 'prompt'] = prompt
# sql 생성
gen_sqls = hf_pipe(df['prompt'].tolist(), do_sample=False,
                    return_full_text=False, max_length=512, truncation=True)
gen_sqls = [x[0]['generated_text'] for x in gen_sqls]
df['gen_sql'] = gen_sqls

# 평가를 위한 requests.jsonl 생성
eval_filepath = "text2sql_evaluation.jsonl"
make_requests_for_gpt_evaluation(df, eval_filepath)
```

- *결과파일을 불러와 프롬프트와 판단 결과 데이터를 각각 **prompts, responses** 변수에 저장.
- *판다스 데이터 조작 라이브러리를 이용하고, **to_csv()** 메서드를 사용해 **csv** 파일로 저장.



```
import torch
from transformers import pipeline, AutoTokenizer, AutoModelForCausalLM

def make_inference_pipeline(model_id):
    tokenizer = AutoTokenizer.from_pretrained(model_id)
    model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto", load_in_4bit=True,
bnb_4bit_compute_dtype=torch.float16)
    pipe = pipeline("text-generation", model=model, tokenizer=tokenizer)
    return pipe

model_id = 'beomi/Yi-Ko-6B'
hf_pipe = make_inference_pipeline(model_id)
```

***2023년 12월 7B 이하 한국어 사전 학습 모델 중 가장 높은 성능을 보이는 beomi/Yi-Ko-6B 모델을 사용**

***make_inference_pipeline** 함수는 입력한 모델 아이디에 맞춰 토큰라이저와 모델을 불러오고 하나의 파이프라인으로 만들어 변환.



```
hf_pipe(example, do_sample=False,  
         return_full_text=False, max_length=512, truncation=True)  
# SELECT COUNT(*) FROM players WHERE username LIKE '%admin%';  
  
# ### SQL 봇:  
# SELECT COUNT(*) FROM players WHERE username LIKE '%admin%';  
  
# ### SQL 봇의 결과:  
# SELECT COUNT(*) FROM players WHERE username LIKE '%admin%'; (생략)
```

***example** 데이터를 **hf_pipe**에 입력하고 결과를 확인.

*기초모델도 **SQL**을 생성할 수 있지만, 형식에 맞춰 답변하기 위해서는 추가적인 학습이 필요.

⤴

[illegible]



```
!pip install transformers==4.30 bitsandbytes==0.43.1 accelerate==0.29.3 datasets==2.19.0 tiktoken==0.6.0
huggingface_hub==0.22.2 autotrain-advanced==0.7.77 -qqq
!pip install --upgrade huggingface-hub -qqq
```

```
!pip install -i https://pypi.org/simple/ bitsandbytes
!pip install --upgrade accelerate -qqq
```

런타임 유형 T4 GPU로 바꿈

```
import torch
from transformers import pipeline, AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig

def make_inference_pipeline(model_id):
    tokenizer = AutoTokenizer.from_pretrained(model_id)
    model = AutoModelForCausalLM.from_pretrained(model_id, quantization_config=BitsAndBytesConfig(load_in_4bit=True))
    pipe = pipeline("text-generation", model=model, tokenizer=tokenizer)
    return pipe
```

- *스택오버플로우를 찾던 중 런타임 유형을 T4 GPU로 바꿔보라는 말을 듣고 시도해봤는데 잘 작동함
- *deprecated 된 부분을 BitsAndBytesConfig 객체로 건네줌



```
from datasets import load_dataset
# 데이터셋 불러오기
df = load_dataset("shangrilar/ko_text2sql", "origin")['test']
df = df.to_pandas()
for idx, row in df.iterrows():
    prompt = make_prompt(row['context'], row['question'])
    df.loc[idx, 'prompt'] = prompt
# sql 생성
gen_sqls = hf_pipe(df['prompt'].tolist(), do_sample=False,
                   return_full_text=False, max_length=512, truncation=True)
gen_sqls = [x[0]['generated_text'] for x in gen_sqls]
df['gen_sql'] = gen_sqls

# 평가를 위한 requests.jsonl 생성
eval_filepath = "text2sql_evaluation.jsonl"
make_requests_for_gpt_evaluation(df, eval_filepath)

os.environ["OPENAI_API_KEY"] = "YOUR-API-KEY"

!python api_request_parallel_processor.py \
  --requests_filepath requests/{eval_filepath} \
  --save_filepath results/{eval_filepath} \
  --request_url https://api.openai.com/v1/chat/completions \
  --max_requests_per_minute 300 \
  --max_tokens_per_minute 100000 \
  --token_encoding_name cl100k_base \
  --max_attempts 5 \
  --logging_level 20
```

*기초 모델 성능 측정

*GPT-4로 평가

*T4 GPU로 약 1시간 20분 소요



```
from datasets import load_dataset

df_sql = load_dataset("shangrilar/ko_text2sql", "origin")["train"]
df_sql = df_sql.to_pandas()
df_sql = df_sql.dropna().sample(frac=1, random_state=42)
df_sql = df_sql.query("db_id != 1")

for idx, row in df_sql.iterrows():
    df_sql.loc[idx, 'text'] = make_prompt(row['context'], row['question'], row['answer'])

!mkdir data
df_sql.to_csv('data/train.csv', index=False)
```

*미세 조정을 위해, 학습 데이터를 부르고 **autotrain-advanced**를 이용해 미세 조정을 수행한다.

```
base_model = 'beomi/Yi-Ko-6B'
finetuned_model = 'yi-ko-6b-text2sql'

!autotrain llm \
--train \
--model {base_model} \
--project-name {finetuned_model} \
--data-path data/ \
--text-column text \
--lr 2e-4 \
--batch-size 8 \
--epochs 1 \
--block-size 1024 \
--warmup-ratio 0.1 \
--lora-r 16 \
--lora-alpha 32 \
--lora-dropout 0.05 \
--weight-decay 0.01 \
--gradient-accumulation 8 \
--mixed-precision fp16 \
--use-peft \
--quantization int4 \
--trainer sft
```

***A100 GPU 기준으로 약 1시간이 소요, T4 GPU인 경우 약 8~10배의 시간이 소요 된다고 합니다.**

</LoRA

	GPT-4 평가 정답 수	정답률[%]
lora_r = 8 lora_alpha = 16	70 ~ 72	62.5 ~ 64.3
lora_r = 16 lora_alpha = 32	73	65.2
lora_r = 32 lora_alpha = 64	78	69.6
lora_r = 64 lora_alpha = 128	74 ~ 76	66.1 ~ 67.9

- LoRA 설정 중 랭크를 의미하는 **lora_r**
- LoRA 파라미터와 반영 비율을 결정하는 **lora_alpha** 변경에 따라 성능이 변화.
- 하지만 LoRA 설정과 성능의 관계는 명확하지 않고 데이터셋에 따라 달라지므로 다양한 실험이 필요.

</Dataset Cleansing

- 데이터셋 크기가 약 **1/4** 정도 줄었음에도 불구하고, 정제 전 학습과 비슷한 성능을 나타냄. [도표 생략]
- 데이터셋 정제의 효과인지? 아니면 원래 데이터셋의 크기가 달라져도 성능에 영향이 없는지, 데이터셋 비율을 **20%** 증가시키면서 성능을 확인 [lora_r = 64, lora_alpha = 128 으로 진행]

	GPT-4 평가 정답 수	정답률[%]
20%	55	49.1
40%	61 ~ 62	54.5 ~ 55.4
60%	65	58.0
80%	72	64.3
100%	78	69.6

</Dataset Cleansing

사용하는 데이터셋이 커지면서 성능이 함께 향상 되는 것을 확인할 수 있다.
또한, 데이터 정제 과정에서 데이터셋이 줄었음에도 성능이 유지 되는 것이
정제의 긍정적인 효과임을 짐작해 볼 수 있다.

</SOLAR

	GPT-4 평가 정답 수	정답률[%]
미세 조정 전	43	38.4
미세 조정 후	90 ~ 92	80.4 ~ 82.1


- `base_model` 변수를 SOLAR 모델로 변경.
- SOLAR 모델은 기존의 Yi-Ko-6B 모델에 비해 두배 가까이 큼.
- 따라서 GPU, 메모리, 학습시간, 추론시간 모두 2배 증가하여 실습은 하지 못했습니다.

</Performance

{18.75%}  Yi-6B 기초

{38.40%}  SOLAR 기초

{69.60%}  Yi-6B 미세 조정

{82.14%}  SOLAR 미세 조정

{86.60%}  GPT-4

</Practice

평가 데이터셋 늘리기

더 정확한 평가를 위해 평가 데이터셋을 늘려보자.

오류 분석과 데이터셋 추가 생성

모델이 잘 생성하지 못하는 SQL 패턴이 있는지 분석해보자.

DPO 학습 활용해보기

4장에서 살펴본 DPO 방식을 Text2SQL 모델에 적용해볼 수 있다.

다른 모델 활용해보기

기초 모델 중에서는 코드 생성에 특화된 Code Llama와 같은 모델을 사용해보자.

Thank you