06-02-02. LLM Science Exam

LLM Science Exam

요즘 다양한 LLM이 경쟁적으로 나와 이슈가 되고 있다. 어느 모델이 뛰어난지 점수를 비교하던데 2023년 Kaggle - LLM Science Exam (https://kaggle.com/competitions/kaggle-llm-science-exam) 대회에서는 과학 문제를 풀게 했다. 그 솔루션 중 하나를 살펴보자. Chris Deotte님은 How To Train Open Book Model - Part 2 (https://www.kaggle.com/code/cdeotte/how-to-train-open-book-model-part-2) 노트북을 공유하고 있다. 노트북 앞부분에 언급한 내용과 링크를 살펴보자. JJ님은 오픈북 방식 (https://www.kaggle.com/code/jjinho/open-book-llm-science-exam)으로 문제를 풀었다. 위키피디아 데이터를 먼저 임베딩해서 인덱싱하고, 그 결과를 검색해서 RAG처럼 LLM모델에 같이 넣어주면 더 정확한 답을 찾을 수 있었다.

LLM 기반 RAG(Retrieval-Augmented Generation) 시스템의 전처리 과정은 대량의 데이터를 효율적으로 처리하고 검색 성능을 극대화하기 위해 여러 단계를 거친다. 주요 전처리 과정을 단계별로 살펴보자.

< 그림1. 전처리과정 >

여기에서는 전체 Wikipedia 문서를 문장으로 분할한다. 유사한 문장을 검색해 문제와 함께 맥락으로 제공하기 위해서다. 진호님의 Open Book LLM Science Exam (https://www.kaggle.com/code/jjinho/open-book-llm-science-exam)을 cuDF, cuVS, CAGRA로 바꿔보았다. 기존 코드의 실행 시간은 10분 20초였지만, 우리가 개선한 코드는 3분 20초로 단축됐다.

① 라이브러리 임포트 및 환경 설정

```
!pip install -U /kaggle/input/faiss-cpu-173/faiss_cpu-1.7.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
!pip install "huggingface_hub<0.26.0" # Install an older version of huggingface_hub
!pip install -U sentence-transformers
#!pip install -U /kaggle/input/blingfire-018/blingfire-0.1.8-py3-none-any.whl
!pip install --upgrade blingfire

import os
import gc
#import pandas as pd
import cudf as pd
import numpy as np
import re
from tqdm.auto import tqdm
import blingfire as bf
from collections.abc import Iterable
import faiss
from faiss import write_index, read_index
from sentence_transformers import SentenceTransformer
import warnings
import cupy as cp
from cuvs.neighbors import cagra

warnings.filterwarnings("ignore")

# 설정
MODEL = '/kaggle/input/sentencetransformers-allminilml6v2/sentence-transformers_all-MiniLM-L6-v2'
DEVICE = 0
MAX_LENGTH = 384
BATCH_SIZE = 16
WIKI_PATH = "/kaggle/input/wikipedia-20230701"
NUM_SENTENCES_INCLUDE = 3
```

```
# 텍스트 문장 분할 함수 정의
def  process_documents(documents: Iterable[str],
                       document_ids: Iterable,
                       split_sentences: bool = True,
                       filter_len : int = 3,
                       disable_progress_bar: bool = False) -> pd.DataFrame:
  """
 EMR에서 문서를 처리하는 주요 도우미 함수입니다.

 :param documents: 문자열인 문서를 포함하는 반복 가능 객체
 :param document_ids: 문서 고유 식별자를 포함하는 반복 가능 객체
 :param split_sentences: 섹션을 문장으로 더 분할할지 여부를 결정하는 플래그
 :param filter_len: 문장의 최소 문자 길이(그렇지 않으면 필터링)
 :param disable_progress_bar: tqdm 진행률 표시줄을 비활성화하는 플래그
 :return: `document_id`, `text`, `section`, `offset` 열을 포함하는 Pandas DataF
rame
 """

    df = sectionize_documents(documents, document_ids, disable_progress_ba
r)

    if split_sentences:
        # cuDF Series를 Python 리스트로 변환하여 sentencize에 전달합니다.
        df = sentencize(df.text.to_arrow().to_pylist(),
                        df.document_id.to_arrow().to_pylist(),
                        df.offset.to_arrow().to_pylist(),
                        filter_len,
                        disable_progress_bar)
    return df

def  sectionize_documents(documents: Iterable[str],
                          document_ids: Iterable,
                          disable_progress_bar: bool = False) -> pd.DataFr
ame:
  """
 이미징 보고서의 섹션을 가져오고 선택한 섹션만 반환합니다(기본값은 FINDINGS, IMPRESSION 및 A
DDENDUM).

 :param documents: 문자열인 문서를 포함하는 반복 가능 객체
 :param document_ids: 문서 고유 식별자를 포함하는 반복 가능 객체
 :param disable_progress_bar: tqdm 진행률 표시줄을 비활성화하는 플래그
 :return: `document_id`, `text`, `offset` 열을 포함하는 Pandas DataFrame
 """
    processed_documents = []
    for document_id, document in tqdm(zip(document_ids, documents), total=l
en(documents), disable=disable_progress_bar):
        row = {}
```

```
        text, start, end = (document, 0, len(document))
        row['document_id'] = document_id
        row['text'] = text
        row['offset'] = (start, end)

        processed_documents.append(row)

    _df = pd.DataFrame(processed_documents)
    if _df.shape[0] > 0:
        return _df.sort_values(['document_id', 'offset']).reset_index(drop=
True)
    else:
        return _df

def  sentencize(documents: Iterable[str],
                document_ids: Iterable,
                offsets: Iterable[tuple[int, int]],
                filter_len : int = 3,
                disable_progress_bar: bool = False) -> pd.DataFrame:
  """
 문서를 문장으로 분할합니다. `sectionize_documents`와 함께 사용하여 문서를 더 관리하기 쉬운
조각으로 분할할 수 있습니다.
 분할 후 문장을 원본 문서의 위치와 일치시킬 수 있도록 오프셋을 사용합니다.

 :param documents: 문자열인 문서를 포함하는 반복 가능 객체
 :param document_ids: 문서 고유 식별자를 포함하는 반복 가능 객체
 :param offsets: 시작 및 끝 인덱스의 반복 가능 튜플
 :param filter_len: 문장의 최소 문자 길이(그렇지 않으면 필터링)
 :return: `document_id`, `text`, `section`, `offset` 열을 포함하는 Pandas DataF
rame
 """

    document_sentences = []
    for document, document_id, offset in tqdm(zip(documents, document_ids,
offsets), total=len(documents), disable=disable_progress_bar):
        try:
            _, sentence_offsets = bf.text_to_sentences_and_offsets(documen
t)
            for o in sentence_offsets:
                if o[1]-o[0] > filter_len:
                    sentence = document[o[0]:o[1]]
                    abs_offsets = (o[0]+offset[0], o[1]+offset[0])
                    row = {}
                    row['document_id'] = document_id
                    row['text'] = sentence
                    row['offset'] = abs_offsets
                    document_sentences.append(row)
        except:
```

```
            continue
    return pd.DataFrame(document_sentences)
```

② 데이터 로딩 (위키피디아, 훈련 데이터)

```
# 데이터 로드
wiki_files = os.listdir(WIKI_PATH)
trn = pd.read_csv("/kaggle/input/kaggle-llm-science-exam/train.csv")

# 모델 초기화
model = SentenceTransformer(MODEL, device='cuda')
model.max_seq_length = MAX_LENGTH
model = model.half()

# 위키피디아 인덱스 로드
sentence_index = read_index("/kaggle/input/wikipedia-2023-07-faiss-index/wi
kipedia_202307.index")

# 프롬프트 임베딩
prompt_embeddings = model.encode(trn.prompt.to_pandas().values, batch_size=
BATCH_SIZE, device=DEVICE, show_progress_bar=True, convert_to_tensor=True,
normalize_embeddings=True).half()
prompt_embeddings = prompt_embeddings.detach().cpu().numpy()
_ = gc.collect()

# 상위 3개 페이지 검색
search_score, search_index = sentence_index.search(prompt_embeddings, 3)

del sentence_index
del prompt_embeddings
_ = gc.collect()
```

③ 프롬프트(prompt) 임베딩 생성 및 관련 문서 검색

```
# 위키피디아 인덱스 파일 로드
df = pd.read_parquet("/kaggle/input/wikipedia-20230701/wiki_2023_index.parquet", columns=['id', 'file' ])

# 기사 및 관련 파일 위치 가져오기
wikipedia_file_data = []
for i, (scr, idx) in tqdm(enumerate(zip(search_score, search_index)), total=len(search_score)):
    scr_idx = idx
    _df = df.loc[scr_idx].copy()
    _df['prompt_id'] = i
    wikipedia_file_data.append(_df)

wikipedia_file_data = pd.concat(wikipedia_file_data).reset_index(drop=  True)
wikipedia_file_data = wikipedia_file_data[[  'id', 'prompt_id', 'file' ]].drop_duplicates().sort_values(['file' , 'id']).reset_index(drop=True)

del df
_ = gc.collect()
```

④ 검색된 문서의 텍스트 로딩

```
# 전체 텍스트 데이터 가져오기
wiki_text_data = []
for file  in tqdm(wikipedia_file_data.file.unique().to_arrow().to_pylist(), total=len(wikipedia_file_data.file.unique())):
    _id = [str(i) for i in wikipedia_file_data[wikipedia_file_data[  'file' ]==file][ 'id'].to_arrow().to_pylist()]
    _df = pd.read_parquet(f"{WIKI_PATH}/{file} ", columns=['id', 'text'])
    _df = _df[_df['id'].isin(_id)]
    wiki_text_data.append(_df)
    _ = gc.collect()

wiki_text_data = pd.concat(wiki_text_data).drop_duplicates().reset_index(drop=True)
_ = gc.collect()
```

⑤ 문서를 문장 단위로 분할하고 임베딩 생성

```
# 위키피디아 문서를 문장으로 분할
processed_wiki_text_data = process_documents(wiki_text_data.text.to_arro
w().to_pylist(), wiki_text_data.id.to_arrow().to_pylist())

# 문장 임베딩
wiki_data_embeddings = model.encode(processed_wiki_text_data.text.to_arro
w().to_pylist(), batch_size=BATCH_SIZE, device=DEVICE, show_progress_bar=Tr
ue, convert_to_tensor=True, normalize_embeddings=True).half()
wiki_data_embeddings = wiki_data_embeddings.detach().cpu().numpy()
_ = gc.collect()
```

⑥ 질문과 선택지 결합 후 임베딩 생성

```
# 답변 결합
trn['answer_all'] = trn['A'].str.cat([trn['B'], trn['C'], trn['D'], trn['
E']], sep=" ")
trn['prompt_answer_stem'] = trn['prompt'] + " " + trn['answer_all']

# 질문 임베딩
question_embeddings = model.encode(trn['prompt_answer_stem'].to_arrow().to_
pylist(), batch_size=BATCH_SIZE, device=DEVICE, show_progress_bar=True, con
vert_to_tensor=True, normalize_embeddings=True).half()
question_embeddings = question_embeddings.detach().cpu().numpy()
```

⑦ GPU 기반 유사도 검색을 통한 컨텍스트 추출

```
# 컨텍스트 검색 및 결합
prompt_contexts = []
contexts = []

# 위키 데이터 임베딩을 GPU에 올리고 인덱스 생성
index_params = cagra.IndexParams()
wiki_data_embeddings = cp.asarray(wiki_data_embeddings, dtype=cp.float32)
wiki_index = cagra.build(index_params, wiki_data_embeddings)

trn_pd = trn.to_pandas()

for r in trn_pd.itertuples():
    prompt_context = ""
    prompt_id = r.id
    context = ""
```

```
    if prompt_id >= len(question_embeddings):
        print(f"prompt_id {prompt_id} 가 question_embeddings 범위를 벗어납니다.
건너뜁니다.")
        continue

    query_vector = cp.asarray(question_embeddings[prompt_id], dtype=cp.floa
t32).reshape(1, -1)

    search_params = cagra.SearchParams(max_queries=100, itopk_size=64)

    try:
        distances, indices = cagra.search(search_params, wiki_index, query_
vector, NUM_SENTENCES_INCLUDE)
    except Exception as e:
        print(f"검색 실행 오류 (prompt_id {prompt_id}): {e}")
        continue

    try:
        prompt_text = trn_pd.at[prompt_id, 'prompt']
        choice_a = trn_pd.at[prompt_id, 'A']
        choice_b = trn_pd.at[prompt_id, 'B']
        choice_c = trn_pd.at[prompt_id, 'C']
        choice_d = trn_pd.at[prompt_id, 'D']
        choice_e = trn_pd.at[prompt_id, 'E']
    except Exception as e:
        print(f"질문 혹은 선택지 접근 실패 (prompt_id {prompt_id}): {e}")
        continue

    prompt_context += f"Question: {prompt_text}\n"
    prompt_context += "Choices:\n"
    prompt_context += f"(A) {choice_a}\n"
    prompt_context += f"(B) {choice_b}\n"
    prompt_context += f"(C) {choice_c}\n"
    prompt_context += f"(D) {choice_d}\n"
    prompt_context += f"(E) {choice_e}\n"

    if indices.shape[0] > 0:
        prompt_context += "Context:\n"
        distances_cpu = cp.asnumpy(distances)
        indices_cpu = cp.asnumpy(indices)
        for candidate_idx, candidate_distance in zip(indices_cpu[0], distan
ces_cpu[0]):
            if candidate_distance < 2:
                if candidate_idx < processed_wiki_text_data.shape[0]:
                    candidate_text = processed_wiki_text_data['text'].iloc[
candidate_idx]
                    context += "[*] " + candidate_text + "\n"
```

```
                else:
                    print(f"prompt_id {prompt_id}: 후보 인덱스 {candidate_idx}
가 processed_wiki_text_data 범위를 벗어났습니다.")
        prompt_context += context

    contexts.append(context)
    prompt_contexts.append(prompt_context)
```

⑧ 컨텍스트를 훈련 데이터에 추가하고 저장

```
trn['context'] = contexts
trn.to_csv("./train_context.csv", index=False)
```

⑨ 마지막으로, 결과를 확인한다.

```
# 결과 출력 (상위 10개 질문)
for i, p in enumerate(prompt_contexts[:10]):
    print(f"Question {i}")
    print(p)
    print()
```

Question 0
Question: Which of the following statements accurately describes the impact of Modified Newtonian Dynamics (MOND) on the observed "missing baryonic mass" discrepancy in galaxy clusters?
Choices:
(A) MOND is a theory that reduces the observed missing baryonic mass in galaxy clusters by postulating the existence of a new form of matter called "fuzzy dark matter."
(B) MOND is a theory that increases the discrepancy between the observed missing baryonic mass in galaxy clusters and the measured velocity dispersions from a factor of around 10 to a factor of about 20.
(C) MOND is a theory that explains the missing baryonic mass in galaxy

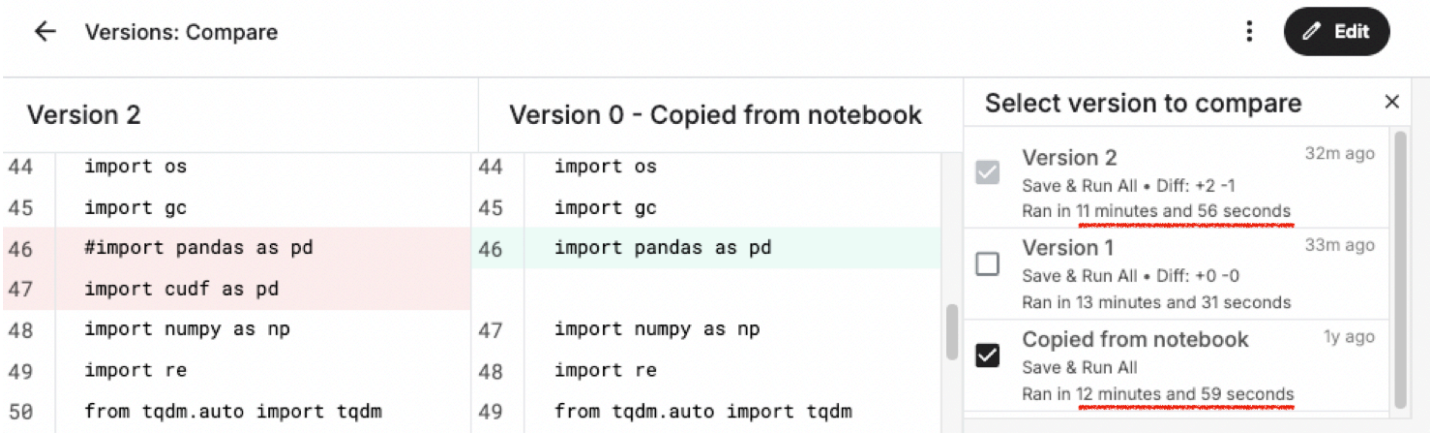
clusters that was previously considered dark matter by demonstrating that the mass is in the form of neutrinos and axions.
(D) MOND is a theory that reduces the discrepancy between the observed missing baryonic mass in galaxy clusters and the measured velocity dispersions from a factor of around 10 to a factor of about 2.
(E) MOND is a theory that eliminates the observed missing baryonic mass in galaxy clusters by imposing a new mathematical formulation of gravity that does not require the existence of dark matter.
Context:
[*] While almost all astrophysicists today reject MOND in favor of dark matter, a small number of researchers continue to enhance it, recently incorporating Brans–Dicke theories into treatments that attempt to account for cosmological observations.
[*] Alternatives to the dark matter hypothesis include a modification of gravity at small accelerations (MOND) or an effect from brane cosmology.
[*] The homogeneously distributed mass of the universe would result in a roughly scalar field that permeated the universe and would serve as a source for Newton's gravitational constant; creating a theory of quantum gravity. === MOND === Modified Newtonian Dynamics (MOND) is a relatively modern proposal to explain the galaxy rotation problem based on a variation of Newton's Second Law of Dynamics at low accelerations.
...
Question 9
Question: What is the role of axioms in a formal theory?
Choices:
(A) Basis statements called axioms form the foundation of a formal theory and, together with the deducing rules, help in deriving a set of statements called theorems using proof theory.
(B) Axioms are supplementary statements added to a formal theory that break down otherwise complex statements into more simple ones.
(C) Axioms are redundant statements that can be derived from other

statements in a formal theory, providing additional perspective to theorems derived from the theory.
(D) The axioms in a theory are used for experimental validation of the theorems derived from the statements in the theory.
(E) The axioms in a formal theory are added to prove that the statements derived from the theory are true, irrespective of their validity in the real world.
Context:
[*] A formal system is an abstract structure used for inferring theorems from axioms according to a set of rules.
[*] The explication of the particular axioms used in a theory can help to clarify a suitable level of abstraction that the mathematician would like to work with.
[*] A formal theory is an axiomatic system (usually formulated within model theory) that describes a set of sentences that is closed under logical implication.

< code 1. Open Book LLM Science Exam-GPU >

Chris Deotte님의 솔루션 (https://www.kaggle.com/competitions/kaggle-llm-science-exam/discussion/446318)을 보면 Part 2에서는 RAPIDS TF-IDF를 사용해 점수를 향상시켰다.

Pandas를 cuDF로 바꾸는 것 만으로도 실행 시간을 단축 시킬 수 있었다.



다양한 RAG기법 뿐만아니라 RAPIDS 24.10 버전부터 사용 가능한 cuVS/CAGRA를 활용하여 GPU 가속을 통해 성능 개선에 도전해 보자.

참고문헌

- QwQ-32B (https://modelscope.cn/models/Qwen/QwQ-32B)
- Benchmark performance of DeepSeek-R1 (https://github.com/deepseek-ai/DeepSeek-R1/blob/main/DeepSeek_R1.pdf)
- Will Lifferth, Walter Reade, and Addison Howard. Kaggle - LLM Science Exam. https://kaggle.com/competitions/kaggle-llm-science-exam, 2023. Kaggle. (https://kaggle.com/competitions/kaggle-llm-science-exam)
- JJ, Open Book LLM Science Exam. https://www.kaggle.com/code/jjinho/open-book-llm-science-exam, 2023. Kaggle. (https://www.kaggle.com/code/jjinho/open-book-llm-science-exam)
- Sung Hur, Open Book LLM Science Exam-cuVS CAGRA. https://www.kaggle.com/code/kmmmvs/open-book-llm-science-exam-cuvs-cagra, 2025. Kaggle. (https://www.kaggle.com/code/kmmmvs/open-book-llm-science-exam-cuvs-cagra)
- Chris Deotte, How To Train Open Book Model - Part 2. https://www.kaggle.com/code/cdeotte/how-to-train-open-book-model-part-2, 2023. Kaggle. (https://www.kaggle.com/code/cdeotte/how-to-train-open-book-model-part-2)
- RAFT (https://github.com/rapidsai/raft)
- rapids.ai/cuVS (https://rapids.ai/cuvs/)
- github-cuVS (https://github.com/rapidsai/cuvs)
- CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs (https://ar5iv.labs.arxiv.org/html/2308.15136)

마지막 편집일시 : 2025년 8월 16일 1:38 오전

댓글 0 피드백

- **이전글** : 06-02-01. cuVS를 활용한 GPU 최적화 CAGRA 인덱스 구축 및 검색
- **다음글** : 07. NVIDIA RAPIDS 응용

(/)