"AI News That's Fit to Print"

The AI Newsletter

Bimonthly

Late Edition

K-water연구원 AI연구센터에서 정기적으로 발간하는 뉴스레터입니다. AI 분야의 뉴스 뿐 아니라 코딩실습과 기술정보 등 다양한 내용과 난이도로 담아냈습니다.

AI Newsletter No.14B

AI LAB, MONDAY, MAR 31, 2025

AI Tech

Tips

[파이썬 코딩] 효율적인 반복문 활용하기

for 루프는 리스트, 튜플, 문자열, 딕셔너리 등 반복 가능한(iterable) 객체의 요소를 하나씩 가져와 순차적으로 실행하는 반복문이다. 주어진 객체 내부를 순회하며 특정 작업을 반복적으로 수행할 때 사용된다. 머릿속으로 쉽게 떠올릴 수 있는 직관적인 구조 덕분에 초보자에게 유용하며, 데이터를 다룰 때 가장 먼저 떠오르는 반복문 방식이다. AI를 활용한 연구 및 데이터 분석에서는 보통 pandas 패키지를 기반으로 데이터프레임을 다룬다. 하지만 데이터프레임의 크기가 커질수록 for 루프는 비효율적인 방법이 된다. for 루프는 데이터프레임의 각 요소를 하나씩 인덱싱하여 접근하기 때문에 연산 속도가 느려지고, 코드 실행 시간이 길어질 수 있기 때문이다.

그렇다면, for 루프를 어떻게 효율적으로 개선할 수 있을까? 대표적인 해결책으로 리스트 컴프리헨션(List Comprehension)과 벡터 연산(Vectorized Operations)이 있다. 리스트 컴프리헨션은 반복문을 한 줄로 표현할 수 있도록 도와주는 파이썬 문법으로, 기존 for 루프보다 간결하면서도 실행 속도가 빠르다. 또한, pandas와 함께 가장 기본적으로 사용되는 라이브러리인 numpy에는 rolling 등 벡터 연산 기반 함수가 충실하게 내장되어 있기 때문에, 이를 잘 활용하면 데이터를 매우 빠르게 처리할 수 있다.

백만 개의 행을 가진 데이터프레임 df와 컬럼 feature를 가정해 보자. feature 컬럼의 값을 제곱하여 새로운 컬럼 feature_squared를 추가하는 과정을 for 루프와 리스트 컴프리헨션, 벡터 연산으로 구현하면 다음과 같다. 코드가 간소화될 뿐만 아니라, 데이터프레임이 커질수록 실행 시간도 유의미하게 개선될 것이다.

① for 루프 사용

```
feature_squared = [ ]
for x in df["feature"]:
feature_squared.append(x ** 2)
df["feature_squared"] = feature_squared
```

② 리스트 컴프리헨션

```
df["feature_squared"] = [x ** 2 for x in df["feature"]]
```

③ 벡터 연산

```
df["feature_squared"] = df["feature"] ** 2
```

최근 GPT 등 생성형 AI의 발전으로 인해, 코드 최적화도 AI에게 맡길 수 있는 시대가 되었다. 예를 들어, "이 for 루프를 numpy의 벡터 연산을 활용하도록 개선해줘"라고 요청하면 AI가 최적화한 코드를 바로 확인할 수도 있으니, 가능한 한 반복문을 줄이고, 리스트 컴프리헨션과 벡터 연산을 적극 활용해 보기 바란다!

딥시크, 제대로 알아보자!

딥시크의 기원

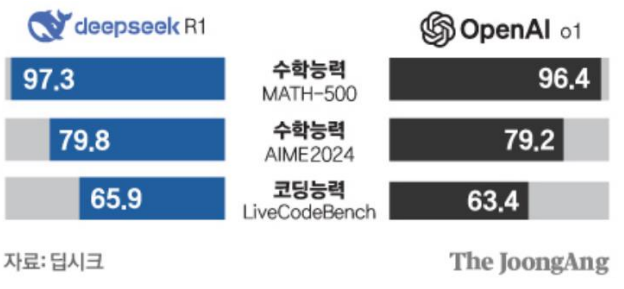
딥시크는 지난 2023년 량원펑(Liang Wenfeng)에 의해 설립된 중국의 AI 스타트업이다. 량원펑은 지난 2015년 대학 동창 2명과 함께 헤지펀드(Hedge Fund) 회사 '하이 플라이어'(환팡량화)를 세웠다. 하이 플라이어는 약 13억 달러(한화 약 1조 8천억 원) 이상을 조달 한 중국 최초의 헤지펀드가 되었다. 2019년부터는 투자 기법 정교화를 위해 하이 플라이어 내에 AI 전담 부서를 만들었다. 그의 팀은 엔비디아의 GPUR H800을 사용해 주식 거래에서 수익을 창출했다. 이후 량원펑은 2023년 5월 하이 플라이어의 AI 조직을 데리고 분사해 딥시크를 창업하였다.

딥시크의 특징

DeepSeek V3 모델은 671억 개의 파라미터를 가진 전문가 혼합(MoE) 모델을 사용하는 대형 언어 모델로, 각 토큰에서 37억 개의 파라미터를 활성화하여 낮은 에너지로 높은 효율을 제공한다. DeepSeek V3 모델은 14.8T 토큰으로 학습되어 고성능을 제공하고 사용자에게 낮은 전력을 소비하여 간결한 출력을 생성한다. 또한 멀티 토큰 예측(MTP) 방식을 사용하여 사용자의 쿼리를 이해하고 복잡한 작업을 간결하게 수행하며, 다중 토큰 예측(MTP)은 모델의 학습 효율을 높이고 더 빠르고 정확하게 결과를 생성할 수 있게 해준다.

가성비 비교

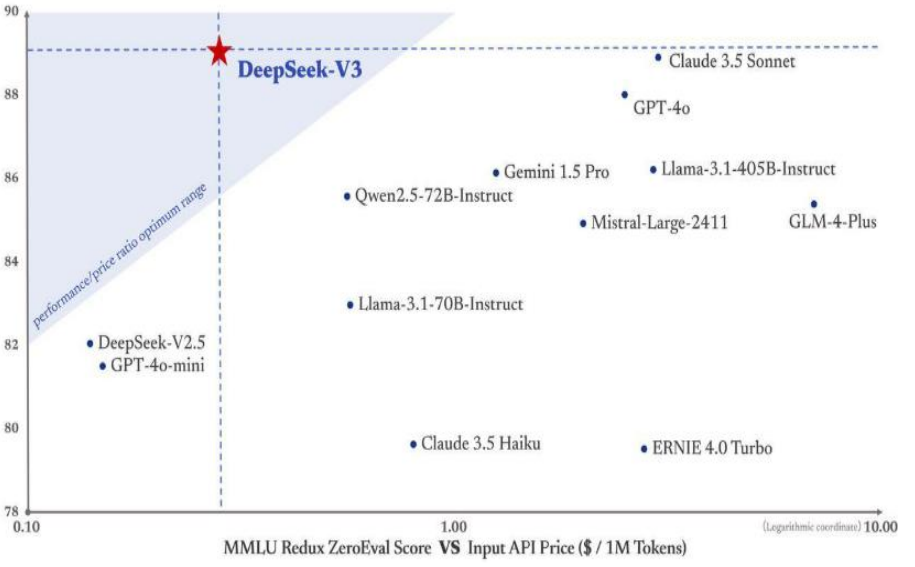
GPT-4o는 입력 캐시 히트 토큰 백만 개당 1.25달러, 입력 캐시 미스 토큰 백만 개당 2.50달러, 출력 토큰 백만 개당 10달러를 청구한다. 반면 딥시크 V3 모델은 입력 캐시 히트 토큰 100만 개당 0.07달러, 입력 캐시 미스 토큰 100만 개당 0.27달러, 출력 토큰 100만 개당 1.10달러를 청구한다. 두 가지 대형 언어 모델의 가격을 비교하면, DeepSeek V3 모델이 예산 친화적이고 성능이 뛰어나다고 말할 수 있다.



<deepseek R1과 ChatGPT o1 모델 비교>

deepseek 딥시크(DeepSeek)	구분	OpenAI 오픈AI(OpenAI)
량원펑(梁文峰)	창업자	샘 알트만(Sam Altman)
2023년	설립	2015년
180명	직원수(추정)	1,200명
R1	대표AI모델	o1
560만 달러(추정)	학습비용	1억 달러(추정)
오픈소스	모델공개	폐쇄형
엔비디아 저사양칩 H800	투입된 반도체	엔비디아 고성능칩 H100
90.20%	성능비교 (MATH-500)	74.60%

<딥시크·오픈AI 비교(출처: 이글루코퍼레이션)>



<여러 대형 언어 모델들의 가격 비교>

Tips

이용 가격

일반 사용자는 챗GPT를 자주 이용하지는 않기 때문에 이용료를 지급하면서까지 사용하기는 부담스러울 것이다. 딥시크는 이런 부담을 크게 완화했다. 입력토큰 100만 개당 0.55달러 출력토큰 100만 개당 2.19달러로 이용할 수 있게 하여, 일반 이용자들은 천 원도 안되는 저렴한 비용으로 AI 모델(딥시크 R1)을 사용할 수 있게 됐다.

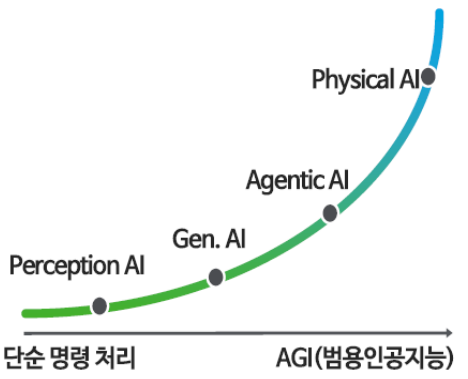
<딥시크-오픈AI AI 모델 가격 정책 비교(출처: Indian Startup News)>

'25.02.01부 가격 정책

MODEL	Input Tokens ($ per 1M)	Output Tokens ($ per 1M)	Total Cost (1M Tokens)
Deepseek-chat(V3)	$0.07~$0.27	$1.10	$1.17~$1.37
Deepseek-reasoned	$0.14~$0.55	$2.19	$2.33~$2.74
OpenAI GPT-4o	$5.00	$15.00	$20.00
OpenAI GPT-4 Turbo	$10.00	$30.00	$40.00
OpenAI GPT-3.5 Turbo	$0.50	$1.50	$2.00

시사점

딥시크의 AI 모델은 AI 주도권 변화와 도메인 특화모델의 확산을 이끌고, AI 시장확대 및 반도체 시장의 다변화를 촉진하며, 중장기적으로 에이전틱 AI와 물리적 AI의 상용화를 앞당길 것으로 기대된다. AI개발 비용 구조의 변화로는 고비용 인프라 구축과 빅테크 중심의 개발에서 스타트업 중심으로 이동할 것으로 예상되며, 미국 중심의 AI 기술 리더십이 위협받게 될 것으로 생각된다. 또한, 저가 및 중저가형 칩의 수요가 증가하게 되어 NVIDIA에서 AMD, Intel, ASIC 등의 중저가 칩 공급업체 들에게 새로운 기회가 될 것이다. 중장기적으로는 에이전틱 AI로의 전환이 가속화되어 실시간 데이터 처리와 의사결정 품질 강화를 기하고, 저사양 하드웨어 활용을 통한 물리적 AI(예: 로봇) 개발 비용이 절감 될 것으로 기대된다.



Hands-on AI Project

온라인 쇼핑의 유사 상품 추천, 어떻게 이루어질까?

온라인 쇼핑을 하면서 한번쯤 이런 경험을 한 적이 있지 않은가? 내가 관심 있는 제품을 검색하면, 그 아래에 '방금 본 상품과 유사한 제품'과 같은 추천 목록이 나타나는 것을 본 적이 있을 것이다. 이러한 추천 시스템의 핵심 기술 중 하나가 바로 **이미지 유사도 분석**이다. 단순히 상품 이름이나 설명이 비슷해서 추천하는 것이 아니라, 상품의 이미지를 분석하여 시각적으로 유사한 상품을 찾아주는 것이라고 볼 수 있다.

이번 Hands-on에서는 지난 호에서 다뤘던 CLIP을 다시 한번 사용하여 이미지 유사도를 평가해보는 실습을 해보도록 하겠다. CLIP을 통해 여러 이미지를 생성 한 후 ① 생성된 이미지가 내가 원하는 실제 이미지와 얼마나 비슷한지, ② 다른 주제로 생성된 이미지들이 얼마나 명확히 구분되는지를 알아볼 예정이다. 이미지 유사도의 분석 과정은 아래를 참조하길 바란다. 이제 본격적인 코딩 실습으로 들어가보자!

📌 이미지 유사도 분석 📌

- 이미지는 사람 눈에는 직관적이지만, 컴퓨터 입장에서는 단순한 픽셀의 배열
- 컴퓨터가 이미지를 이해하고 비교하려면 이미지를 수치화하여 표현하는 과정이 필요, 이러한 과정을 이미지 임베딩(Image Embedding)이라고 함
- 이미지가 딥러닝 모델(CLIP, ResNet 등)을 통과하면, 이미지 임베딩을 통해 고차원 벡터 형태로 변환되며 이 벡터는 이미지의 핵심 특징을 표현하는 압축 정보
- 벡터 간 거리(유사도)를 측정하는 다양한 방법이 있지만 본 실습에서는 가장 대중적인 방법인 코사인 유사도 사용

※ 전체 Source 코드 및 결과는 분량상 생략, 아래 링크를 참고해주세요.
구글 코랩에서 열기(인터넷망)

① 관련 라이브러리 선언

```
# 필요한 라이브러리 설치
!pip install git+https://github.com/openai/CLIP.git
!pip install torch torchvision Pillow
!pip install diffusers transformers accelerate torch torchvision numpy matplotlib
googletrans==4.0.0-rc1
```

```
# 해당 라이브러리에서 필요한 모듈 참조
from diffusers import StableDiffusionPipeline
from googletrans import Translator
import torch
import clip
from numpy import dot
from numpy.linalg import norm
from matplotlib import pyplot as plt
import requests
from PIL import Image
from io import BytesIO
```

② GPU 선택 및 번역기 세팅 (지난 호와 동일한 관계로 생략)

③ 텍스트(프롬프트) 입력

```
# 텍스트 입력
# 한국어 프롬프트 입력
korean_prompt = "귀여운 포메라니안" # 그리고 싶은 그림의 텍스트 입력

# 한국어 -> 영어 번역
english_prompt = translator.translate(korean_prompt, src="ko", dest="en").text
print(f"한국어 프롬프트: {korean_prompt}")
print(f"번역된 프롬프트: {english_prompt}")
```

④ 이미지 생성 및 저장

```
# 이미지 생성
num_images = 1 # 생성할 이미지 수
guidance_scale = 7.5 # 텍스트의 중요도를 조정

# 이미지 생성 실행
images = pipeline(english_prompt, num_inference_steps=50,
guidance_scale=guidance_scale, num_images_per_prompt=num_images).images

# 생성된 이미지 시각화
for idx, image in enumerate(images):
    plt.imshow(image)
    plt.axis("off")
    plt.title(f"Generated Image {idx+1}")
    plt.show()

# 이미지 저장
for idx, image in enumerate(images):
    image.save(f"dog_{idx+1}.png")
    print(f"Image {idx+1} saved as 'generated_image_{idx+1}.png'")
```

Hands-on AI Project

⑤ 실제 이미지 불러오기 및 저장

```
# 이미지 URL
image_url =
"https://postfiles.pstatic.net/20121226_18/qelih406_1356508246895R4uFT_PNG/
%B1%CD%B0%AD13.png?type=w1" #복사한 이미지 URL 붙여넣기

# URL에서 이미지 불러오기
response = requests.get(image_url)
img = Image.open(BytesIO(response.content)).convert("RGB")

# 이미지 시각화
plt.imshow(img)
plt.axis("off")
plt.show()

# 이미지 저장
img.save("dog_real.png")
```

⑥ 이미지 유사도 분석

```
# 이미지 임베딩 추출 함수
def get_embedding(img_path):
    image =
preprocess(Image.open(img_path).convert("RGB")).unsqueeze(0).to(device)
    with torch.no_grad():
        embedding = model.encode_image(image)
        embedding /= embedding.norm(dim=-1, keepdim=True)  # 정규화
    return embedding.cpu().numpy().flatten()

# 코사인 유사도 계산 함수
def cosine_similarity(vec1, vec2):
    return dot(vec1, vec2) / (norm(vec1) * norm(vec2))

# 이미지 경로 지정
img_paths = {
    "dog_fake": "dog_1.png",
    "dog_real": "dog_real.png",
    "tiger_fake": "tiger_1.png",
    "tiger_real": "tiger_real.png",
}

# 임베딩 벡터 생성
embeddings = {name: get_embedding(path) for name, path in img_paths.items()}

# 유사도 계산 및 출력
pairs = [("dog_fake", "dog_real"), ("dog_fake", "tiger_fake"), ("tiger_fake",
"tiger_real")]

for img1, img2 in pairs:
    sim = cosine_similarity(embeddings[img1], embeddings[img2])
    print(f"유사도({img1} ↔ {img2}): {sim:.4f}")
```