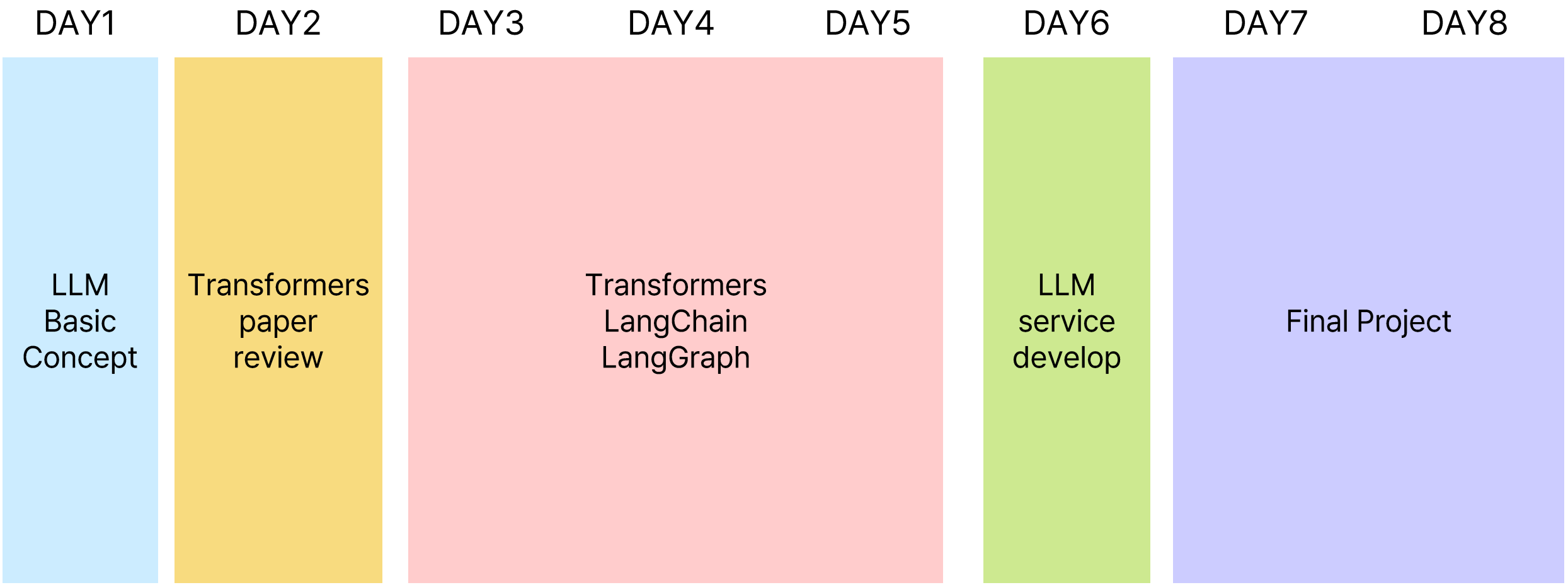
GEN AI 인텐시브 과정

강사 장철원

Section 0

코스소개



Section 0

목차

GEN AI 인텐시브 과정

Section 1. 트랜스포머

Section 1-1. 트랜스포머 라이브러리 소개

Transformers 라이브러리

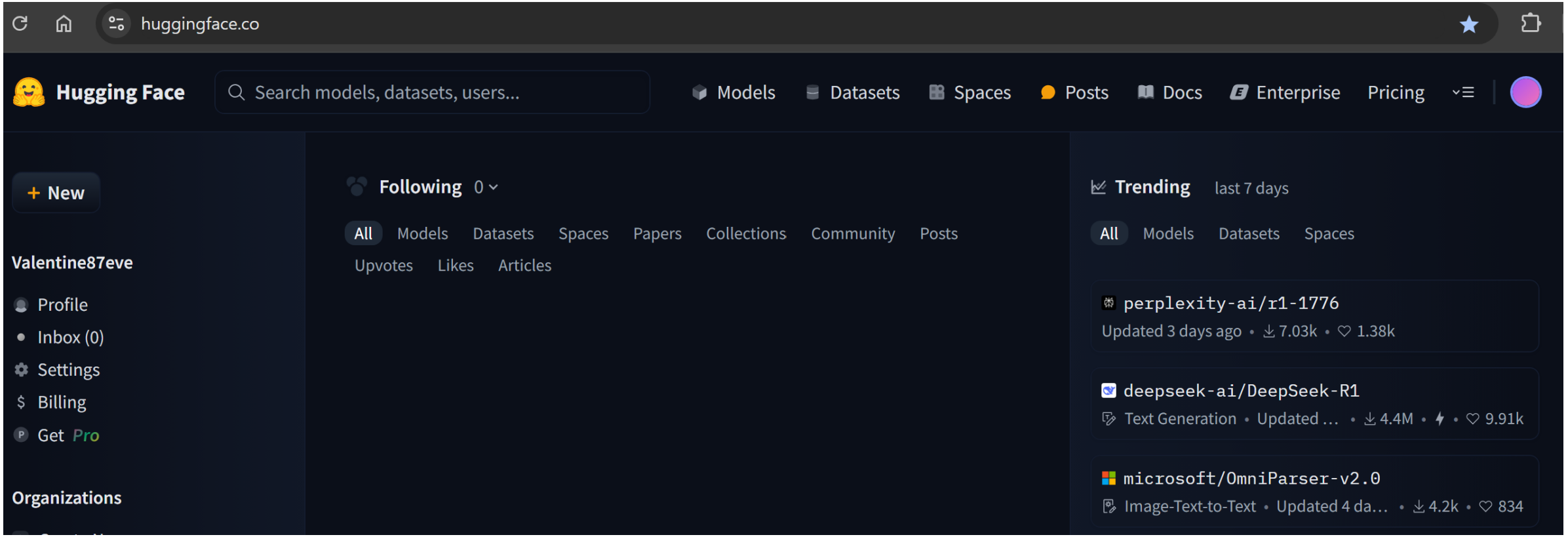
- transformers 라이브러리는 Hugging Face에서 만든 LLM 활용시 필수 라이브러리
- 사전 학습된 모델(pretrained models)을 쉽게 불러와서 텍스트 생성, 분류, 번역, 요약 등 다양한 작업 가능

복잡한 NLP 모델을 쉽게 가져와서 바로 사용하자!

Section

트랜스포머 라이브러리 소개

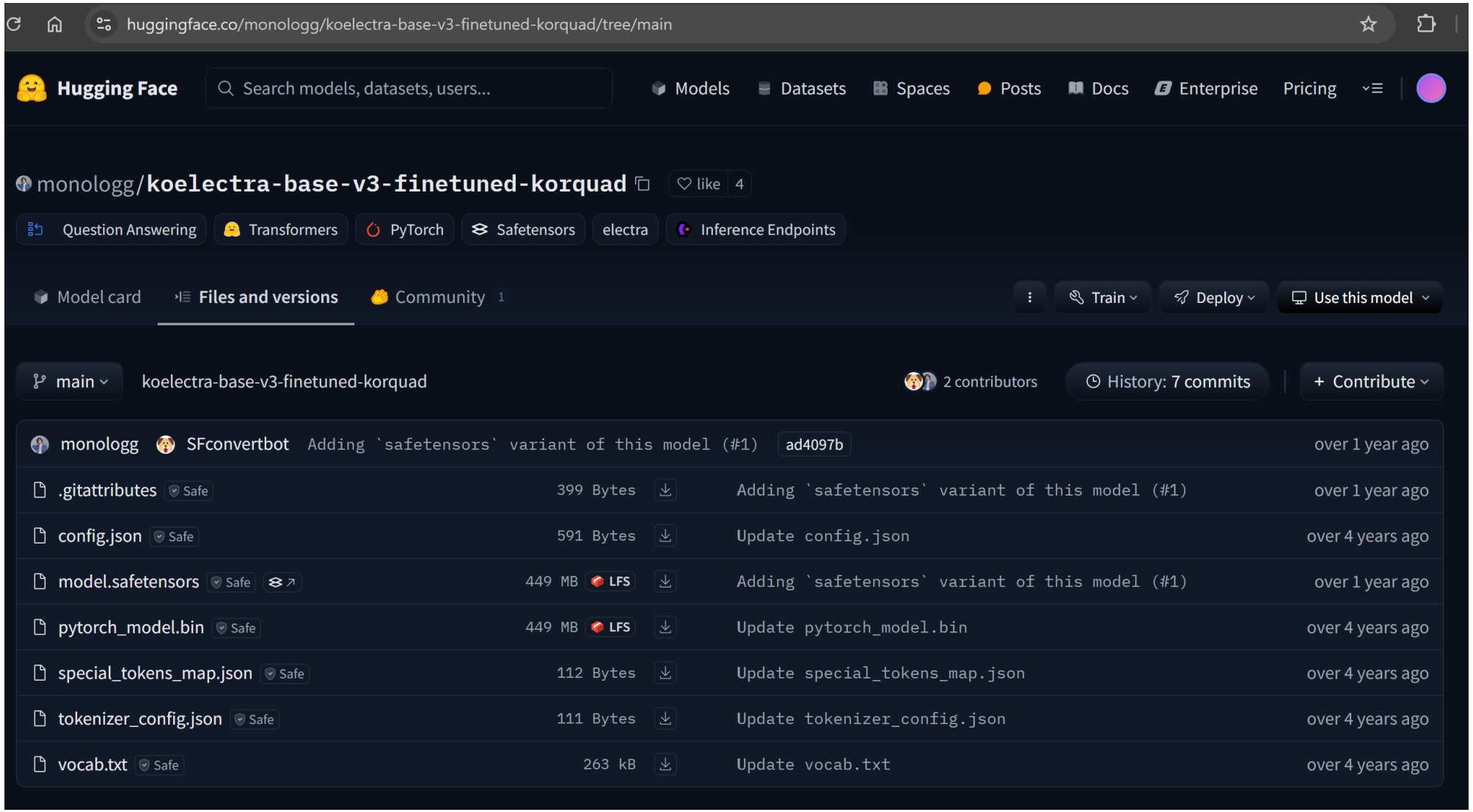
Hugging Face



Section

트랜스포머 라이브러리 소개

사용하고자 하는 모델 확인



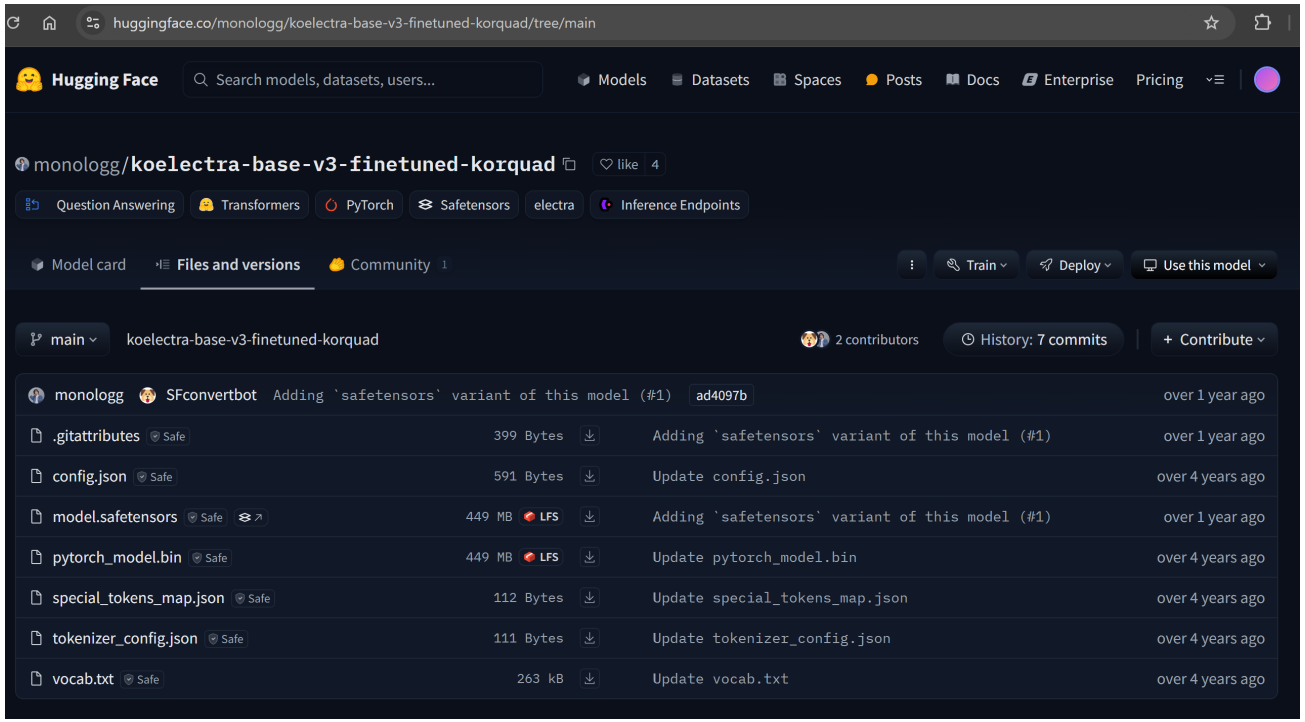
Section

트랜스포머 라이브러리 소개

Transformers

Local

Model Download



Hugging Face

GEN AI 인텐시브 과정

Section 1. 트랜스포머 라이브러리

Section 1-2. 파이썬 기초

딕셔너리 언패킹

```
hyper_parameter = {"a": 3, "b": "birthday", "c": [1,2,3,4,5]}
```

```
def explain1(a, b, c):
    res1 = a
    res2 = b
    res3 = c
    return res1, res2, res3
```

```
explain1(**hyper_parameter)
```

딕셔너리 언패킹(unpacking)

- 딕셔너리 내부의 key-value 쌍을 풀어서 함수에 전달

```
(3, 'birthday', [1, 2, 3, 4, 5])
```

```
def explain2(param):
    return {**param, "d": 30}
```

**param 딕셔니링 언패킹해서, d 추가

→ 기존 key-value 쌍에 새로운 key-value 추가

```
explain2(hyper_parameter)
```

```
{'a': 3, 'b': 'birthday', 'c': [1, 2, 3, 4, 5], 'd': 30}
```

EvalPrediction

```
import numpy as np
from transformers.trainer_utils import EvalPrediction
```

- transformers 라이브러리에서 모델 평가 시 사용되는 예측 결과와 실제 라벨을 담는 객체
- 이후 실습에서 허깅페이스에서 Trainer가 평가할 때 자동으로 이 함수를 호출하면서 EvalPrediction 객체로 만듬

```
def accuracy_score(predict):
    preds = np.argmax(predict.predictions, axis=1)
    acc = (preds == predict.label_ids).mean()
    return {"accuracy": acc}
```

정확도(accuracy)만드는 함수
- 입력값 predict는 EvalPrediction 객체에 해당

```
y_hat = np.array([[0.2, 0.8], [0.4, 0.6], [0.1, 0.9]])
y_true = np.array([1, 0, 1])
```

예측값
실제값

```
pred = EvalPrediction(predictions=y_hat, label_ids=y_true)
print(pred.predictions)
print('----------------')
print(pred.label_ids)
```

```
[[0.2 0.8]
 [0.4 0.6]
 [0.1 0.9]]
----------------
[1 0 1]
```

```
y_hat
```

```
array([[0.2, 0.8],
       [0.4, 0.6],
       [0.1, 0.9]])
```

0 1

```
np.argmax(y_hat, axis=1)
```

```
array([1, 1, 1], dtype=int64)
```

```
y_hat
```

```
array([[0.2, 0.8],
       [0.4, 0.6],
       [0.1, 0.9]])
```

```
np.argmax(y_hat, axis=0)
```

```
array([1, 2], dtype=int64)
```

EvalPrediction

```
print(vars(pred))
```
객체 내부 정보 확인할 때 사용(1)

```
{'predictions': array([[0.2, 0.8],
       [0.4, 0.6],
       [0.1, 0.9]]), 'label_ids': array([1, 0, 1]), 'inputs': None, 'losses': None, 'elements': (array([[0.2, 0.8],
       [0.4, 0.6],
       [0.1, 0.9]]), array([1, 0, 1]))}
```
예측결과

실제 정답

```
print(pred.__dict__)
```
객체 내부 정보 확인할 때 사용(2)

```
{'predictions': array([[0.2, 0.8],
       [0.4, 0.6],
       [0.1, 0.9]]), 'label_ids': array([1, 0, 1]), 'inputs': None, 'losses': None, 'elements': (array([[0.2, 0.8],
       [0.4, 0.6],
       [0.1, 0.9]]), array([1, 0, 1]))}
```

```
accuracy_score(pred)
```

```
{'accuracy': 0.6666666666666666}
```

GEN AI 인텐시브 과정

Section 1. 트랜스포머 라이브러리

Section 1-3. 긍부정 분석

import 트랜스포머 라이브러리

- 모델마다 적합한 토크나이저가 다를 수 있음
- AutoTokenizer를 사용하면 모델 이름이 주어졌을 경우 적절한 토크나이저 자동 선택

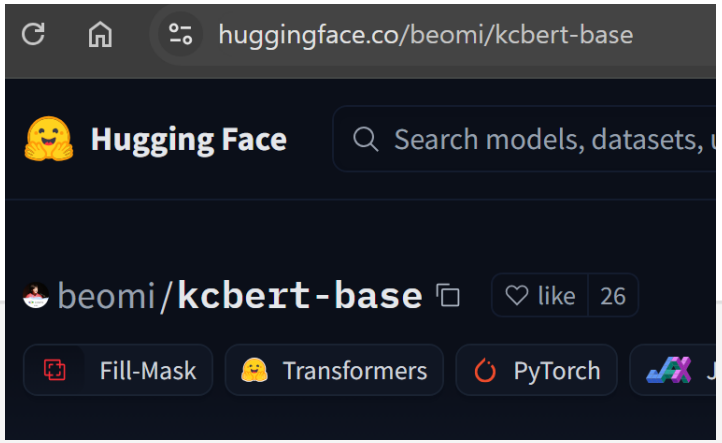
```
[6]: import torch
     from transformers import AutoTokenizer, AutoModelForSequenceClassification
```

- transformers 라이브러리에서 제공하는 자동 모델 로더 중 하나
- 분류(classification) 태스크를 수행하는 사전 훈련된 모델을 로드하는 역할

* 토크나이저: 모델이 문장을 이해할 수 있도록 텍스트를 숫자로 변환해 토큰으로 나타내는 역할

Section
긍부정 분석

모델 불러오기



BERT 기반으로 한국어로 사전 학습된 모델

```
model_name = "beomi/kcbert-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoin
t at beomi/kcbert-base and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predict
ions and inference.
```

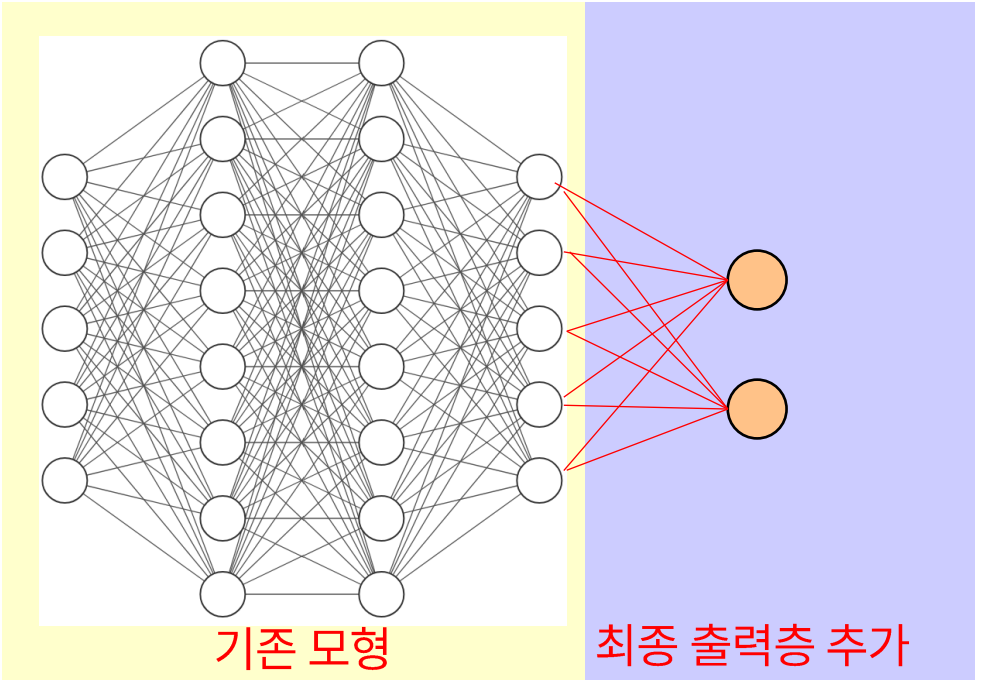
현재 불러온 모델 “beomi/kcbert-base”는 언어 모델이라 기본적인 문맥 이해 능력은 있으나

분류 모델이 아니므로 분류 레이어가 따로 없다는 뜻. 따라서 분류기로 사용하려면 파인튜닝 필요.

(다음장에 계속 설명)

Section
긍부정 분석

모델 불러오기




```
model_name = "beomi/kcbert-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

사전 학습된(pretrained) 모델, 임베딩 레이어 포함

모델에 맞는 토크나이저 자동 설정

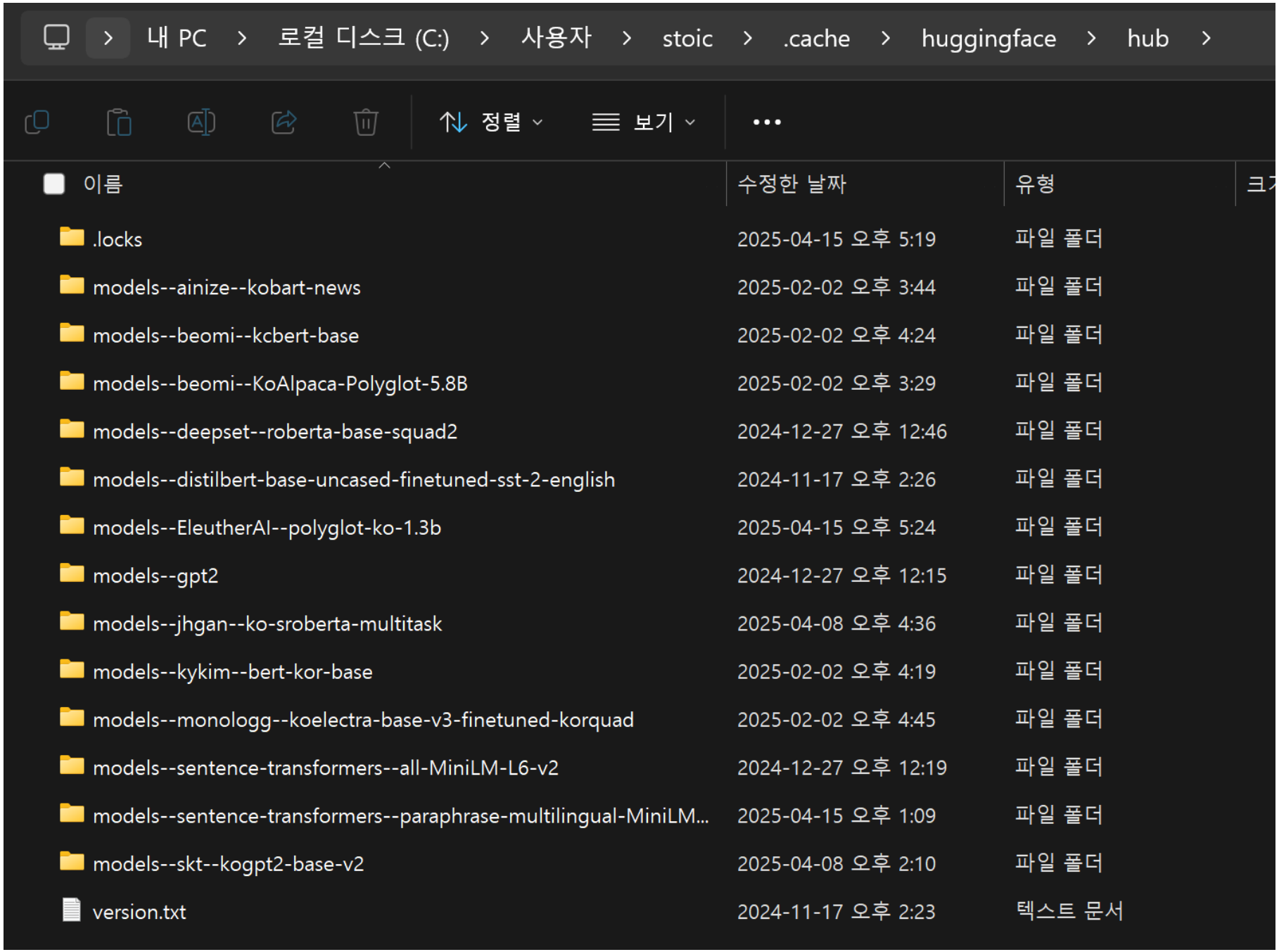
긍정/부정 두개로 분류할 예정이므로 2로 설정

- 분류 작업을 위해서 신경망에 마지막 출력층 추가(Adapter 아님)
- 이때 추가되는 출력층은 사전학습에 포함되어 있지 않으므로 가중치가 무작위로 부여됨
- 따라서 추후 추가학습 필요

Section
긍부정 분석

모델 저장 위치

사용자/.cache/huggingface/hub



Section
긍부정분석

분류할 텍스트 설정

```
text = "이 영화 너무 감동적이었어! 최고야!"
inputs = tokenizer(text, return_tensors="pt")
```

출력 결과물의 형태

return_tessors="pt" → pytorch
return_tessors="tf" → tensorflow
return_tessors="np" → numpy

```
inputs
```

```
{'input_ids': tensor([[    2,  2451,  9376,  8069, 13912,  8805, 11320,     5,  8619,  4144,
             5,     3]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]),
'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

주요 토큰

토큰	의미
1	패딩 토큰
2	문장의 시작
3	문장의 끝 혹은 문장 구분

```
inputs.input_ids
```

→ 텍스트를 토큰화 한 후 각 토큰을 숫자 ID로 변환

```
tensor([[    2,  2451,  9376,  8069, 13912,  8805, 11320,     5,  8619,  4144,
             5,     3]])
```

```
inputs.token_type_ids
```

→ 문장 구분 정보, 따옴표 한 쌍당 문장 1개로 인식

```
tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
inputs.attention_mask
```

→ 실제 입력 토큰은 1, 패딩 토큰은 0, 텐서 내부에서 주목(attention)해야할 부분, 1이면 주목, 0이면 무시

```
tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

Section
긍부정분석

추론

```
model.eval()
with torch.no_grad():
    outputs = model(**inputs)
```

이 코드를 생략하는 경우도 많지만, 코드 안정성을 위해 쓰는게 좋음

```
outputs
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[0.1845, 0.4710]]), hidden_states=None, attentions=None)
```

```
outputs.logits
```

```
tensor([[0.1845, 0.4710]])
```

1번째가 높으므로 1번째 결과로 출력할건데, 문제는 1번째가 긍정인지 부정인지 모르는 상태.

```
logits = outputs.logits
sentiment = torch.argmax(logits).item()
```

```
print("감성 분석 결과:", "긍정" if sentiment == 1 else "부정")
```

```
감성 분석 결과: 긍정
```

임의로 1을 긍정으로 설정

추가학습 필요

감사합니다.

Q & A