1. 허깅페이스 BART 모델 개요 [BART모델.ipynb]

1-1. 기본 정보

- Transformer encoder-decoder (seq2seq) 모델 Hugging Face Hugging Face
- BERT-like 양방향 인코더 + GPT-like 자기회귀 디코더 facebook/bart-large-cnn · Hugging Face
- 텍스트 노이즈 추가 후 원문 복구 방식으로 사전훈련 facebook/bart-large-cnn · Hugging Face

1-2. 주요 모델 종류

facebook/bart-base	• 기본 BART 모델 • Mask filling 작업 가능 Bart — transformers 2.11.0 documentation
facebook/bart-large	• 대형 BART 모델, 텍스트 인필링용 facebook/bart-large · Hugging Face • Mask filling 지원 Bart — transformers 2.11.0 documentation
facebook/bart-large-cnn	• CNN Daily Mail 데이터로 요약 태스크에 파인튜닝 facebook/bart-large-cnn · Hugging Face • 요약 전용 모델 Summarizing Text Using Hugging Face's BART Model - DEV Community
facebook/bart-large-mnli	• 제로샷 분류 전용 facebook/bart-large-mnli · Hugging Face • 자연어 추론(NLI) 태스크용 Finetuning Hugging Face Facebook Bart Model | by Lidor ES | Medium

***참고: T5 (Text-to-Text Transfer Transformer)**

구글에서 발표한 범용 NLP 모델 (2019)./ 모든 태스크(분류, 요약, 번역 등)를 **"텍스트 입력 → 텍스트 출력"** 형태로 통일해서 학습 / 크기별로 T5-Base, T5-Large 등이 있으며, BART와 비슷한 시기에 **요약, 번역, QA** 등에서 비교 대상이 됨.

모델명	파라미터	레이어	학습 데이터	주요 특징	활용 태스크
facebook/bart-base	약 139M	인코더6 디코더6	영어 일반 코퍼스	- 범용 모델 (사전학습) - 파인튜닝 필수 - 가볍고 빠름	연구, 프로토타입, 소규모 태스크
facebook/bart-large 마스크채우기	약 406M (large모델 기반)	인코더12 디코더12 (large모델 기반)	영어 일반 코퍼스	- base보다 강력 - Mask filling 지원 - 복잡한 텍스트 처리	고성능 필요 범용 태스크
facebook/ bart-large-cnn 뉴스기사 요약특화			CNN/Daily Mail	- 뉴스 요약 특화 - Mask filling 불가 - 요약 품질 우수 (T5보다 강점)	문서/뉴스 요약
facebook/ bart-large-mnli 자연어 추론(NLI)을 통한 분류 특화			MNLI (자연어 추론)	- 분류 전용 - 제로샷 지원 - 새로운 라벨도 대응	제로샷 분류, 감정/주제 분류
facebook/ bart-large-xsum 한 문장 요약 특화			XSum (BBC 뉴스)	- 한 문장 요약 전용 - 핵심만 추출 - 극단적 요약 스타일	짧은 요약, 뉴스 브리핑

▶ 성능비교표 : CNN/DailyMail 데이터셋 (뉴스 기사 요약) /XSum 데이터셋 (BBC 뉴스 한 문장 요약)

모델명	파라미터 레이어	학습 데이터	주요 특징	활용 태스크
BART-Large-CNN	맥락 이해, 핵심 보존 탁월	영어 일반 코퍼스	- 범용 모델 (사전학습) - 파인튜닝 필수 - 가볍고 빠름	연구, 프로토타입, 소규모 태스크
	CNN/DailyMail 요약에서 **ROUGE-1, ROUGE-2, ROUGE-L** 모두 가장 높은 점수.장문의 맥락을 잘 이해하면서 핵심 요약에 강점			
T5-Large	유창성 뛰어나나 세부정보 손실	영어 일반 코퍼스	- base보다 강력 - Mask filling 지원 - 복잡한 텍스트 처리	고성능 필요 범용 태스크
	유창하고 매끄러운 문장을 잘 생성하지만, **세부 정보 손실**이 잦음. 즉, 읽기엔 부드럽지만 정확도가 떨어질 수 있음.			
T5-Base	단편적이고 부정확	CNN/Daily Mail	- 뉴스 요약 특화 - Mask filling 불가 - 요약 품질 우수 (T5보다 강점)	문서/뉴스 요약
	모델 크기가 작아 요약이 단편적이고 정보가 누락되는 경우가 많음.			

▶ (참고) 성능비교표 관련링크

- 아래 링크들에서는 주로 CNN/DailyMail, XSum, 기타 뉴스 요약 데이터셋에서의 ROUGE 점수를 기준으로 두 모델의 성능을 비교하고 있음

"Abstractive Text Summarization Using Pre-trained Models" (ArXiv 2023)
BART-Large-CNN, T5-base, Pegasus 등을 CNN/DailyMail, XSum, SAMSum 데이터셋에서 평가했습니다.
모델별 ROUGE 및 BLEU 점수를 제시함.
Hugging Face Forums+15arXiv+15arXiv+15 OSF

"Abstractive Text Summarization for Resumes With Cutting Edge NLP Transformers" (ArXiv 2023)
Resume 데이터셋과 오픈소스 뉴스 요약 데이터셋(XSum, CNN/DailyMail 등)에서 BART, T5, Pegasus, LSTM 성능 비교.
BART-Large가 가장 좋은 성능을 보였다고 보고.
arXiv+2arXiv+2

"Evaluating BART, T5, and PEGASUS for Effective…" (IETA 2024)
BART와 T5 비교: BART가 평균 F1 점수 33%, T5는 26%로, **BART가 더 높은 ROUGE/F1** 성능을 기록함.
Medium+14IIETA+14arXiv+14

"Make Lead Bias in Your Favor: Zero-shot Abstractive News…" (ArXiv 2019)
BART-LB 및 T5-LB와 기존 BART-Large, T5-Large의 요약 성능 비교 (zero-shot).
ROUGE-1 점수 기준으로 BART-LB가 BART-Large보다 최대 13.7% 더 높은 성능.
Medium+6arXiv+6Microsoft+6

Medium 블로그: "T5-Base, T5-Large, and BART – The Battle of the Summarization Models"
실험 예시와 함께 BART-Large-CNN이 T5-Large보다 더 읽기 좋은 요약을 생성한다는 인사이트 제시.

2. BART Mask Filling

facebook/bart-base 와 facebook/bart-base-large모델에서 지원됨

BART는 사전학습(pretraining) 단계에서 마스크 언어 모델(MLM, Masked Language Model) 방식을 포함한 다양한 노이즈를 적용합니다.
따라서 BERT처럼 <mask> 토큰이 포함된 입력 문장을 주면, 빈칸 채우기(mask filling)를 수행할 수 있습니다. 대표적으로 facebook/bart-large 모델이 mask filling을 지원합니다.
(반면, bart-large-cnn 같은 요약 특화 모델은 mask filling 기능을 지원하지 않음 → mask_token_id 없음) 입력 문장에서 일부 단어나 구절을 <mask>로 치환

- BERT보다 유연함 : BERT도 mask filling이 가능하지만, 단순한 단어 예측에 가까움
- BART는 문장 수준에서 학습했기 때문에 더 자연스러운 문맥 복원이 가능
- 여러 개 마스크 처리 가능: 하나의 문장에 여러 <mask>가 있을 때, 순차적으로 또는 동시에 예측 가능
- 생성적 특성: GPT처럼 오토리그레시브 디코딩을 하기 때문에,
  단순 단어 치환이 아니라 문맥에 맞는 다양한 후보 생성 가능

▪ 영어 mask

```
from transformers import pipeline, AutoTokenizer
# BART-large는 영어 전용 모델
unmask = pipeline("fill-mask", model="facebook/bart-large")
tok = AutoTokenizer.from_pretrained("facebook/bart-large")
print("mask_token:", tok.mask_token)  # -> <mask>
# 영어 입력 예시
text = f"I went to the {tok.mask_token} today."
results = unmask(text, top_k=5)
for r in results:
    print("→", r["sequence"], round(r["score"], 4))
```

```
Device set to use cuda:0
mask_token: <mask>
→ I went to the doctor today. 0.1187
→ I went to the dentist today. 0.0778
→ I went to the gym today. 0.0483
→ I went to the grocery today. 0.0309
→ I went to the chirop today. 0.0247
```

▪ 한글 mask

```
from transformers import pipeline, AutoTokenizer
model_id = "klue/roberta-base"
unmask = pipeline("fill-mask", model=model_id, tokenizer=model_id)
tok = AutoTokenizer.from_pretrained(model_id)
text = f"나는 오늘 {tok.mask_token}에 갔다."
results = unmask(text, top_k=5)
for r in results:
    print("→", r["sequence"], round(r["score"], 4))
```

```
→ 나는 오늘 에 갔다. 0.0872
→ 나는 오늘? 에 갔다. 0.038
→ 나는 오늘 # 에 갔다. 0.0152
→ 나는 오늘. 에 갔다. 0.0104
→ 나는 오늘 제주도 에 갔다. 0.0096
```

- 마스킹전략을 이용한 텍스트 복원 -

텍스트 손상(예: OCR 오류, 철자 오타, 비정상 문자)은 단순한 규칙 기반 교정기로는 복원하기 어렵습니다. m@n@gement나 w0rd 같은 손상은 사전에 없는 형태이므로, 어떤 부분이 잘못되었는지 모델에게 **명확하게 알려주는** 전략이 필요합니다.

마스킹(Masking)은 바로 이 역할을 합니다. 손상된 부분을 [MASK]와 같은 특수 토큰으로 표시하면, 모델은 그 부분에 집중하여 문맥에 가장 잘 맞는 단어나 문구를 예측하고 복원할 수 있습니다. 즉, 모델의 강력한 언어 이해 능력에 **'어디를 고쳐야 할지'** 힌트를 주는 것이 마스킹 전략의 핵심입니다..

단순한 오타 몇 개를 고치는 작업은 BERT로도 충분할 수 있지만, OCR 결과물이나 심하게 파편화된 문장처럼 문맥 전체가 손상된 경우에는 **BART가 가장 안정적이고 효과적인 선택**입니다. BART는 손상된 문장을 '재구성'하는 데 최적화된 모델이기 때문입니다.

--> BART의 마스킹을 활용하는 이유

BART(Bidirectional and Auto-Regressive Transformers)는 **'Denoising AutoEncoder'** 구조를 가진 Seq2Seq(인코더-디코더) 기반 언어 모델입니다. 그 이름에서 알 수 있듯, **원래 목적 자체가 손상된 텍스트를 복원하는 것**입니다.

BART는 사전 학습 과정에서 의도적으로 문장을 손상시킨 후, 이를 원래대로 되돌리는 훈련을 반복합니다. 이 훈련에는 다음과 같은 다양한 손상 기법이 포함됩니다.

- **랜덤 마스킹**: 단어 일부를 가림
- **텍스트 셔플**: 문장 순서를 뒤섞음
- **삭제 후 복원**: 단어나 문장 일부를 지움

이러한 사전 학습 경험 덕분에, BART는 복잡하게 훼손된 텍스트를 입력받아도 자연스럽고 온전한 원문으로 재구성하는 데 탁월한 능력을 보입니다.

* 모델별 비교

	BERT	GPT	BART
장점	- 마스크된 단어를정확하게 예측 - 짧은 토큰 교정에 강함	- 주어진 문맥을 보고 자연스러운 문장 생성 - 손상 부위 주변 맥락을 보고 합리적 후보 생성	- 사전 학습 자체가 복원 목적 - 손상된 문장 전체를 '재구성' 가능 - 다양한 손상(마스킹, 순서 뒤섞임 등) 복원 경험
한계	- 문장 전체의 흐름을 자연스럽게 잇는 데 약함 - 긴 문맥 복원에 한계	- 손상 부분을 정확히 지정하기 어려움 - 원문에 없던 내용을 '창작'할 위험	- 모델 크기가 크고 계산 비용이 높음
적합한 상황	오타 몇 글자 수준의 단어 단위 복원	문맥을 이어 새로운 문장을 생성해야 할 때	OCR, 문장 파편화 등 복잡하고 다층적인 텍스트 복원

마스킹 전략들은 다음과 같이 **단순성**과 **정확성**이라는 두 가지 상충하는 목표를 조절하여 결정할수 있습니다.

- **단순성**: 기본 복원, 단일 마스킹
- **정확성**: 다중 마스킹, 반복적 마스킹

복원하려는 텍스트의 손상 정도와 처리 속도 요구사항에 따라 가장 적합한 전략을 선택하는 것이 중요합니다. 예를 들어, 빠른 처리가 필요한 경미한 오류에는 단일 마스킹이 적합하며, 정확도가 가장 중요한 문서 복원에는 반복적 마스킹이 최적의 선택이 될 수 있습니다.

다중 마스킹은 복원 성능이 뛰어나지만, 전처리 규칙 설계·복원 후보 관리·후처리 평가까지 요구되므로 난이도가 높습니다. 따라서 실제 복원 파이프라인에서는 보통 단일 마스킹 → 다중 마스킹 → 반복적 마스킹 순으로 점진적으로 적용합니다.

1. 기본 복원 (No Masking)

이 전략은 가장 단순하지만, BART의 핵심 능력인 '디노이징(Denoising)'을 가장 잘 보여주는 방식입니다. 전처리 없이 손상된 텍스트를 그대로 입력하면, BART는 사전 학습을 통해 익힌 문장 재구성 능력을 발휘해 스스로 오류를 찾아내고 고칩니다. 이 방식은 **경미한 오타나 단순한 노이즈**에는 효과적이지만, m@n@gement처럼 의미가 크게 훼손된 경우에는 BART의 디노이징 능력만으로는 한계가 있습니다.

2. 단일 마스킹 (Single Masking)

제시된 내용은 정확합니다. 오류가 명확한 단어에 &mask&를 적용해 모델이 **해당 위치에만 집중**하도록 유도하는 전략입니다. 이는 복원 정확도를 크게 높이고, 불필요한 연산을 줄여 효율적입니다. 하지만 여러 오류가 복합적으로 나타나는 경우에는 한 번의 마스킹만으로는 충분한 복원을 기대하기 어렵습니다. 예를 들어, w0rd와 h@llo가 한 문장에 동시에 있다면, 단일 마스킹으로는 두 오류를 모두 처리할 수 없습니다

3. 다중 마스킹 전략 (Multi-Strategy Masking)

이 전략은 단일 마스킹의 한계를 극복하기 위해 여러 마스킹 방식을 결합하는 고급 전략입니다. 제시된 세부 전략들은 다음과 같이 요약할 수 있습니다.

- Conservative: 가장 안전한 접근법. 확실한 오류만 마스킹하여 오버피팅을 방지합니다.
- Aggressive: 의심 가는 모든 부분을 마스킹하여 복원 후보를 최대한 많이 확보합니다. 정확도는 떨어질 수 있지만, 놓치는 오류를 최소화할 수 있습니다.
- Selective: 각 단어의 손상 정도를 정밀하게 평가하여 마스킹 강도를 조절합니다. 예를 들어, w0rd는 &mask&로, managemnt는 manag&mask&nt로 마스킹하는 방식입니다.

이러한 다중 전략은 복원 후보를 다양하게 만들 수 있지만, 결국 어떤 후보가 최적인지 결정하는 **후처리(post-processing) 과정**이 필요하다는 단점이 있습니다.

4. 반복적 마스킹 (Iterative Masking)

이 전략은 **점진적 개선**에 초점을 맞춥니다. 한 번에 모든 오류를 복원하는 것이 아니라, 오류의 종류나 심각도에 따라 **단계적으로** 마스킹 강도를 높여가며 여러 번의 복원 과정을 거칩니다.

- **1차 복원**: 가장 명확한 오류(예: 숫자 포함)만 마스킹 후 복원합니다.
- **2차 복원**: 1차 복원 결과에서 남아있는 다른 오류(예: 특수 문자)를 다시 마스킹하고 복원합니다.
- **3차 복원**: 모든 의심 패턴을 마스킹하여 최종 복원을 시도합니다.

이 방식은 **다층적인 손상이 혼재된 경우**에 특히 효과적입니다. 한 번에 모든 것을 해결하려다 실패하는 것보다, 단계별로 정확도를 높여나가므로 최종 복원 품질이 매우 높습니다. 다만, 여러 번의 연산을 거치기 때문에 **처리 속도가 느리다**는 단점이 있습니다.

전략	1. 기본 복원 (No Masking)	2. 단일 마스킹 (Single Masking)	3. 다중 마스킹 전략 (Multi-Strategy Masking)	4. 반복적 마스킹 (Iterative Masking)
개념·아이디어	손상된 텍스트를 그대로 입력 → BART의 denoising 능력에 의존	명확히 손상된 부분을 &mask&로 지정, 집중 복원	여러 전략을 병행, 다수 복원 후보 생성	마스킹 강도를 점진적으로 ↑ 하며 단계별 개선
적용 방식 / 패턴	전처리 없음	- 숫자 포함 단어: w0rd → <mask> - 특수문자 섞임: h@llo → <mask> - - OCR 오류: m@n@gement → <mask>	[1] Conservative: 명확 오류만 마스킹 [2] Aggressive: 의심되는 모든 패턴 마스킹 [3] Selective: 단어별 정밀 평가 후 마스킹	1차: 숫자만 마스킹 2차: 숫자 포함 단어 마스킹 3차: 모든 의심 패턴 마스킹
장점	- 구현 간단 - 경미한 오타, 노이즈에 효과적	- 정확도 향상 - 특정 오류 패턴에 효과적 - 계산 비용 적음	- 다양한 후보 확보- 상황 맞춤 전략 가능- 복원 유연성 ↑	- 단계별 품질 향상 추적 - 정보 손실 최소화 - 복잡 손상에 효과적
한계·단점	- 심각한 손상에 취약- OCR 특수 오류 복원 한계	- 한 번에 하나의 마스킹만 적용 - 복잡 손상에 한계	- 후보 많아 후처리 필요- 계산량 증가	- 처리 속도 느림 - 반복 과정 관리 필요
적합한 시나리오	- 오타·경미한 노이즈- 빠른 처리 필요할 때	- 중간 정도 손상 - 오류 유형이 뚜렷할 때	- 품질 혼재 데이터 - 복원 후보 중 최적 선택 필요할 때	- 다층적 손상 텍스트 - OCR+오타 혼합 - 정확도 최우선 문서 복원

[참고자료 확인] 참고: BART의 마스킹을 이용한 복원전략.ipynb

3. 텍스트 요약 모델 : facebook/ bart-large-cnn

▶ **facebook/ bart-large-cnn 기본모델은 아래와 같이 단어길이가 긴 자료에 적합함.**

- 너무 짧은 입력(예: 1~2 문장)은 요약이 아니라 거의 "재진술(paraphrase)" 수준이 됨
- BART는 똑똑하지만, 적절한 길이 설정이 핵심임.
- 의료/AI 기사 (약 400단어) / 기후변화 뉴스 (약 350단어) / 양자컴퓨팅 기술 (약 380단어)

텍스트 길이	원본 텍스트가 너무 짧으면 요약이 제대로 되지 않음 최소 200-300단어는 되어야 함.
적절한 파라미터	max_length: 50-150 정도 (원본의 1/3-1/5 길이) min_length: max_length의 1/2
do_sample=False	일관된 결과를 위해 False (요약,번역등 정확성이 중요할 때) - 항상 가장 확률이 높은 단어를 선택 /같은 입력에 항상 같은 결과 생성 - 일관성 있고 예측 가능한 출력 *do_sample=True 이면 확률 분포에 따라 랜덤하게 단어 선택 (창작성) 같은 입력에도 매번 다른 결과 생성, 더 창의적이고 다양한 출력
권장 비율	원본 대비 요약본은 보통 20-30% 길이가 적절

[CodeN.1]

```
from transformers import pipeline

# 요약 파이프라인 생성
summarizer = pipeline("summarization", model="facebook/bart-large-cnn")

from transformers import pipeline, BartTokenizer

# 1. 모델 로드
def load_bart():
    summarizer = pipeline("summarization", model="facebook/bart-large-cnn")
    tokenizer = BartTokenizer.from_pretrained("facebook/bart-large-cnn")
    return summarizer, tokenizer

# 2. 길이별 요약 비교
def test_lengths(text, min_len, max_len):
    summarizer, tokenizer = load_bart()

    original_tokens = len(tokenizer.encode(text))

    result = summarizer(text, max_length=max_len, min_length=min_len, do_sample=False)
    summary = result[0]['summary_text']
    summary_tokens = len(tokenizer.encode(summary))

    display(f"원본 토큰: {original_tokens}")
    display(f"설정: {min_len}~{max_len} 토큰")
    display(f"요약: {summary}")
    display(f"실제 토큰: {summary_tokens}")
    display("-" * 30)

# 테스트
text = """Artificial intelligence is revolutionizing healthcare by transforming how doctors diagnose diseases and treat patients.
Machine learning algorithms can now analyze medical images like X-rays and MRIs with remarkable accuracy, often matching experienced radiologists. AI is also accelerating drug
discovery, with algorithms analyzing molecular structures much faster than traditional methods."""

# 실행
test_lengths(text, 20, 40)  # 짧게
test_lengths(text, 40, 80)  # 중간
test_lengths(text, 80, 120) # 길게
```

[CodeN.1_결과]

원본 토큰 67	Artificial intelligence is revolutionizing healthcare by transforming how doctors diagnose diseases and treat patients. Machine learning algorithms can now analyze medical images like X-rays and MRIs with remarkable accuracy, often matching experienced radiologists. AI is also accelerating drug discovery, with algorithms analyzing molecular structures much faster than traditional methods (인공지능은 의사가 질병을 진단하고 환자를 치료하는 방식을 변화시켜 의료에 혁명을 일으키고 있습니다. 이제 기계 학습 알고리즘은 X선이나 MRI와 같은 의료 이미지를 놀라운 정확도로 분석할 수 있으며, 경험이 풍부한 방사선 전문의와 일치하는 경우도 많습니다. AI도 약물 개발 가속화 기존 방법보다 훨씬 빠르게 분자 구조를 분석하는 알고리즘을 통해 발견할 수 있습니다)
->설정: 20~40 -> 출력: 37	Machine learning algorithms can now analyze medical images like X-rays and MRIs with remarkable accuracy. AI is also accelerating drug discovery, with algorithms analyzing molecular structures much faster than traditional method (이제 기계 학습 알고리즘은 X선, MRI와 같은 의료 이미지를 놀라운 정확도로 분석할 수 있습니다. AI는 또한 기존 방법보다 훨씬 빠르게 분자 구조를 분석하는 알고리즘을 통해 약물 발견을 가속화하고 있습니다.)
-> 설정: 40~80 -> 출력: 44	Machine learning algorithms can now analyze medical images like X-rays and MRIs with remarkable accuracy, often matching experienced radiologists. AI is also accelerating drug discovery, with algorithms analyzing molecular structures much faster than traditional methods. (이제 기계 학습 알고리즘은 X-ray 및 MRI와 같은 의료 이미지를 놀라운 정확도로 분석할 수 있으며 종종 숙련된 방사선 전문의와 일치합니다. AI는 또한 기존 방법보다 훨씬 빠르게 분자 구조를 분석하는 알고리즘을 통해 약물 발견을 가속화하고 있습니다.)
-> 설정:80~120 -> 출력: 86 원본 67보다 긴 토큰을 요구해서 할루시네이션발생 (빨간색글자)	Machine learning algorithms can now analyze medical images like X-rays and MRIs with remarkable accuracy, often matching experienced radiologists. AI is also accelerating drug discovery, with algorithms analyzing molecular structures much faster than traditional methods. It is revolutionizing healthcare by transforming how doctors diagnose diseases and treat patients. For more information on how to use artificial intelligence in your healthcare, visit the National Institute of Health's website (이제 기계 학습 알고리즘은 X선이나 MRI와 같은 의료 이미지를 놀라운 정확도로 분석할 수 있으며, 경험이 풍부한 방사선 전문의와 일치하는 경우도 많습니다. AI는 또한 기존 방법보다 훨씬 빠르게 분자 구조를 분석하는 알고리즘을 통해 약물 발견을 가속화하고 있습니다. 이는 의사가 질병을 진단하고 환자를 치료하는 방식을 변화시켜 의료에 혁명을 일으키고 있습니다. 의료에 인공지능을 활용하는 방법에 대한 자세한 내용을 보려면 국립보건원 (National Institute of Health) 웹사이트를 방문하세요.)

-> BART의 지능적 길이 조절

짧은 설정 (20~40): 핵심만 추출 → 37토큰

중간 설정 (40~80): 좀 더 상세히 → 44토큰

긴 설정 (80~120): 최대한 확장 → 86토큰

➡ BART는 설정된 길이에 맞춰 내용의 디테일을 조절함

-> 코드를 실행하면 나타나는 경고메세지 - max_length=80, but input_length=66 / 원본(67토큰)보다 긴 요약(80토큰)을 요청하면 경고 , 요약은 보통 원본보다 짧아야 하는게 정상

-> 긴 설정에서의 문제점 (할루시네이션)

마지막 요약에서 이상한 부분: "For more information... visit: www.nhs.uk/machine-learning"

[추가작업 해보세요]

-> do_sample 효과 (가장 중요!) / -> 토큰 vs 단어 차이 (개념 이해)

-> 다른 모델 비교 (facebook/bart-large와도 비교해보세요)

-> 영어 vs 한국어 텍스트로 토큰화 차이 확인

4. 텍스트 분류 모델 : facebook/bart-large-mnli

BART-MNLI는 이 추론 능력을 이용해 제로샷 분류를 수행함으로서 "이 텍스트는 {라벨}에 관한 것이다" 형태로 변환하여 -> 각 라벨별로 참/거짓 확률 계산한후 -> 가장 높은 확률의 라벨 선택
즉) MNLI 학습으로 BART가 텍스트 간의 논리적 관계를 이해하게 되어, 다양한 분류 태스크에 응용 가능!

이 모델의 효과성을 높이려면
(1)관련성 높은 라벨들로 구성 (2)애매한 경우 멀티 라벨 모드 활용 (3)신뢰도 0.5 이하는 재검토 필요 함

기본정보	- MNLI: Multi-Genre Natural Language Inference 데이터 학습 - 제로샷 분류: 학습하지 않은 라벨도 분류 가능 - NLI 원리: "이 텍스트는 {라벨}에 관한 것이다" 형태로 추론
	MNLI (Multi-Genre Natural Language Inference) https://huggingface.co/datasets/nyu-mll/multi_nli https://huggingface.co/datasets/nyu-mll/multi_nli/blob/main/README.md 총 크기: 433k 문장 쌍 | 장르: 10개 (정부, 소설, 여행, 9/11, 전화 통화 등) 언어: 영어만 | 라벨: 3개 클래스 (균등 분포 목표) - 자연어 추론 태스크용 데이터셋 - 전제(Premise) + 가설(Hypothesis) 쌍으로 구성 - 가설이 전제로부터 참/거짓/중립 중 어느 관계인지 판단 ▪ 레이블1: Entailment: 전제가 가설을 지지함 (참) ▪ 레이블2: Contradiction: 전제가 가설과 모순됨 (거짓) ▪ 레이블3: Neutral: 전제만으로는 가설을 판단 불가 (중립)
	예) 전제: "사람이 개를 산책시키고 있다" 가설: "사람이 애완동물과 함께 있다" → Entailment (참) 가설: "사람이 고양이를 키운다" → Neutral (중립) 가설: "사람이 혼자 있다" → Contradiction (거짓)
주요 특징	- 유연한 라벨 설정: 미리 정의된 카테고리 없이 원하는 라벨 사용 - 멀티 라벨 지원: 하나의 텍스트에 여러 카테고리 동시 할당 가능 - 신뢰도 점수: 각 라벨별 확률 점수 제공
핵심파라미터	classifier(text, labels, multi_label=False) text: 분류할 텍스트 labels: 후보 라벨 리스트 multi_label: False: 하나의 라벨만 선택 (기본값) True: 여러 라벨 동시 선택 가능
신뢰도 해석	높음 (0.7+): 매우 확실한 분류 중간 (0.4~0.7): 적당한 신뢰도 낮음 (0.4↓): 불확실한 분류
주요활용분야	뉴스 카테고리 분류: technology, business, sports 등 감정 분석: positive, negative, neutral 의료 분야 분류: cardiology, neurology 등 콘텐츠 태깅: 블로그, 문서 자동 태그 생성
주의사항	영어 특화: 한국어는 토큰 수가 많아지고 정확도 감소 라벨 품질: 명확하고 구체적인 라벨일수록 좋은 결과 텍스트 길이: 너무 짧거나 길면 성능 저하

[신뢰도 출력의 원리]

```
# CodeN.2에서의 신뢰도 비교함수

def compare_confidence():
"""신뢰도 높음 vs 낮음 비교"""
# 높은 신뢰도: 명확한 금융 텍스트
high_result = classifier("Apple stock increased 5% after earnings.", ["finance", "sports"])

# 낮은 신뢰도: 관련없는 라벨들
low_result = classifier("Nice weather today.", ["technology", "politics"])
```

```
출력
신뢰도 비교:
높음: 0.916
낮음: 0.702
```

[BART-MNLI의 내부 작동 방식]

A. NLI(자연어 추론) 원리

```
# 원본: "Apple stock increased 5% after earnings." + ["finance", "sports"]
# 내부적으로 이렇게 변환:
premise = "Apple stock increased 5% after earnings."
hypothesis1 = "This text is about finance."
hypothesis2 = "This text is about sports.“

# 각각에 대해 참/거짓 판단:
# premise가 hypothesis1을 지지하는가? → 참 (0.916)
# premise가 hypothesis2를 지지하는가? → 거짓 (0.084)
```

B. 구체적인 추론 과정

단어 매칭이 아니라 문맥적 연관성 판단 하여서
"Apple stock earnings" → 금융 문맥으로 이해
"Nice weather" → 기술과 억지로 연결하려니 애매함

C. 제로샷을 사용할수 있음

수많은 "전제-가설" 쌍으로 추론 능력 학습한 모델임(특정 도메인이 아닌 일반적인 논리 추론 학습)

미리 정해진 카테고리가 없어도 됨
새로운 라벨을 즉석에서 만들어 사용 가능
"이 텍스트가 {라벨}과 관련있나?"라는 추론 문제로 변환
결론: 단순한 키워드 매칭이 아니라 문장 전체의 의미와 라벨 간의 논리적 연관성을 추론하는 것임.

높은 신뢰도 케이스	전제: "Apple stock increased 5% after earnings." 가설: "This text is about finance." → BART 판단: "Apple", "stock", "earnings" = 명백히 금융 관련 → 결과: 참 (0.916)
낮은 신뢰도 케이스	전제: "Nice weather today." 가설: "This text is about technology." → BART 판단: "weather"와 "technology" = 관련성 애매 → 결과: 약간 참? (0.702)

[CodeN.1] - 데이터셋 형태보기

```
from datasets import load_dataset

# MNLI 데이터셋 로드
dataset = load_dataset("nyu-mll/multi_nli")

def explore_mnli():
    """MNLI 데이터셋 기본 정보 탐색"""

    # 1. 데이터셋 구조 확인
    print("=== 데이터셋 구조 ===")
    print(f"분할: {list(dataset.keys())}")
    print(f"훈련 데이터 크기: {len(dataset['train']):,}")
    print(f"검증 데이터 크기: {len(dataset['validation_matched']):,}")
    print()

    # 2. 데이터 컬럼 확인
    print("=== 데이터 컬럼 ===")
    print(f"컬럼들: {dataset['train'].column_names}")
    print()

    # 3. 라벨 정보
    print("=== 라벨 정보 ===")
    labels = ["entailment", "neutral", "contradiction"]
    for i, label in enumerate(labels):
        print(f"{i}: {label}")
    print()

    # 4. 샘플 데이터 확인
    print("=== 샘플 데이터 ===")
    for i in range(3):
        sample = dataset['train'][i]
        print(f"샘플 {i+1}:")
        print(f"  전제: {sample['premise']}")
        print(f"  가설: {sample['hypothesis']}")
        print(f"  라벨: {labels[sample['label']]}")
        print(f"  장르: {sample['genre']}")
        print()
explore_mnli()
```

```
=== 데이터셋 구조 ===
분할: ['train', 'validation_matched', 'validation_mismatched']
훈련 데이터 크기: 392,702
검증 데이터 크기: 9,815[
=== 데이터 컬럼 ===
컬럼들: ['promptID', 'pairID', 'premise', 'premise_binary_parse', 'premise_parse', 'hypothesis', 'hypothesis_binary_parse', 'hypothesis_parse', 'genre', 'label']
=== 라벨 정보 ===
0: entailment
1: neutral
2: contradiction
=== 샘플 데이터 ===
샘플 1:
  전제: Conceptually cream skimming has two basic dimensions - product and geography.
  가설: Product and geography are what make cream skimming work.
  라벨: neutral
  장르: government
샘플 2:
  전제: you know during the season and i guess at at your level uh you lose them to the next level if if they decide to recall the the parent team the Braves decide to call to recall a guy from triple A then a double A guy goes up to replace him and a single A guy goes up to replace him
  가설: You lose the things to the following level if the people recall.
  라벨: entailment
  장르: telephone

샘플 3:
  전제: One of our number will carry out your instructions minutely.
  가설: A member of my team will execute your orders with immense precision.
  라벨: entailment
  장르: fiction
```

```
def check_genres():
    """장르별 분포 확인"""

    from collections import Counter

    # 장르 분포 계산
    genres = [sample['genre'] for sample in dataset['train']]
    genre_counts = Counter(genres)

    print("=== 장르별 분포 ===")
    for genre, count in genre_counts.most_common():
        print(f"{genre}: {count:,}")
check_genres()
```

```
=== 장르별 분포 ===
telephone: 83,348
government: 77,350
travel: 77,350
fiction: 77,348
slate: 77,306
```

```
def check_labels():
    """라벨별 분포 확인"""

    from collections import Counter

    # 라벨 분포 계산
    labels = [sample['label'] for sample in dataset['train']]
    label_counts = Counter(labels)

    print("=== 라벨 분포 ===")
    label_names = ["entailment", "neutral", "contradiction"]
    for label_id, count in sorted(label_counts.items()):
        print(f"{label_names[label_id]}: {count:,}")
check_labels()
```

```
=== 라벨 분포 ===
entailment: 130,899
neutral: 130,900
contradiction: 130,903
```

```
def find_examples(keyword):
    """특정 키워드가 포함된 예시 찾기"""

    print(f"=== '{keyword}' 포함 예시 ===")
    count = 0
    for sample in dataset['train']:
        if keyword.lower() in sample['premise'].lower() or keyword.lower() in sample['hypothesis'].lower():
            if count >= 3:  # 3개만 출력
                break
            labels = ["entailment", "neutral", "contradiction"]
            print(f"전제: {sample['premise']}")
            print(f"가설: {sample['hypothesis']}")
            print(f"라벨: {labels[sample['label']]}")
            print("-" * 30)
            count += 1
find_examples("dog")  # 'dog' 키워드 예시 찾기
```

```
=== 'dog' 포함 예시 ===
전제: it really is our kids are all grown and gone and away from home so our our new family is the you know our two cats and our dog we never really well we had we did have some time to devote to them you know but not nearly as much time as we have now so they've really become children they're they're real characters they really well all of them are
가설: Since our kids are all grown and gone, we have the pets to fill their spots.
라벨: entailment
------------------------------
전제: make sure that they didn't have to do it again  make hot dogs or some potato chips or
가설: Make sure to have steaks and potatoes.
라벨: contradiction
------------------------------
전제: They also regularly hold sheep dog and sheep-shearing demonstrations, all in a covered auditorium, which allows you to watch the shepherds at work without having to stand out on the hillsides.
가설: If you want to watch the shepherds at work, you can go to the auditorium.
라벨: entailment
------------------------------
```

[CodeN.2] 추론

```
from transformers import pipeline, AutoTokenizer

# 모델 로드 (한 번만 실행)
classifier = pipeline("zero-shot-classification", model="facebook/bart-large-mnli")
tokenizer = AutoTokenizer.from_pretrained("facebook/bart-large-mnli")

def test_classification(text, labels, multi_label=False):
    """기본 분류 테스트"""
    tokens = len(tokenizer.encode(text))
    result = classifier(text, labels, multi_label=multi_label)

    print(f"텍스트: {text}")
    print(f"토큰 수: {tokens}, 멀티라벨: {multi_label}")
    print("결과:", result['labels'][0], f"({result['scores'][0]:.3f})")
    print("-" * 30)

def compare_confidence():
    """신뢰도 높음 vs 낮음 비교"""
    # 높은 신뢰도: 명확한 금융 텍스트
    high_result = classifier("Apple stock increased 5% after earnings.", ["finance", "sports"])

    # 낮은 신뢰도: 관련없는 라벨들
    low_result = classifier("Nice weather today.", ["technology", "politics"])

    print("신뢰도 비교:")
    print(f"높음: {high_result['scores'][0]:.3f}")
    print(f"낮음: {low_result['scores'][0]:.3f}")
    print("-" * 30)

def compare_languages():
    """영어 vs 한국어 토큰 수 비교"""
    en_tokens = len(tokenizer.encode("AI transforms healthcare."))
    ko_tokens = len(tokenizer.encode("AI가 의료를 변화시킨다."))

    print("언어별 토큰 수:")
    print(f"영어: {en_tokens}, 한국어: {ko_tokens}")
    print(f"비율: {ko_tokens/en_tokens:.2f}배")
    print("-" * 30)

def compare_modes():
    """단일 라벨 vs 멀티 라벨 비교"""
    text = "AI fitness app monitors health data."
    labels = ["technology", "health", "fitness"]

    # 단일 라벨: 하나만 선택
    single = classifier(text, labels, multi_label=False)
    print(f"단일: {single['labels'][0]}")

    # 멀티 라벨: 임계값 이상 모두 선택
    multi = classifier(text, labels, multi_label=True)
    selected = [l for l, s in zip(multi['labels'], multi['scores']) if s > 0.5]
    print(f"멀티: {selected}")
    print("-" * 30)

# 실행
if __name__ == "__main__":
    test_classification("iPhone has better camera.", ["technology", "sports"])
    compare_confidence()
    compare_languages()
    compare_modes()
```

[CodeN.2_추론결과]

- BART-MNLI는 영어에 최적화됨 (한국어 5배 비효율)
- 구체적 키워드가 있을수록 높은 신뢰도
- 멀티 라벨은 실제 업무에서 더 유용할 수 있음
- 0.7 이상의 신뢰도에서 실무 활용 권장
- 결론: 영어 텍스트 분류에는 매우 우수하지만, 한국어는 심각한 한계가 있음

코드결과	해석
Device set to use cuda:0 텍스트: iPhone has better camera. 토큰 수: 7, 멀티라벨: False 결과: technology (0.987)	매우 높은 신뢰도 (98.7%) - BART가 확신을 가지고 분류 iPhone이라는 명확한 기술 제품명으로 인해 확실한 분류 교훈: 구체적인 제품명/브랜드가 있으면 분류 정확도 상승
신뢰도 비교: 높음: 0.916 낮음: 0.702	높은 신뢰도: "stock", "earnings" 등 금융 전문용어로 명확함 낮은 신뢰도: 날씨는 기술/정치와 관련성이 낮아 애매함 실무 기준: 0.7 이상은 신뢰할 만, 0.5 이하는 재검토 필요
언어별 토큰 수: 영어: 6, 한국어: 30 비율: 5.00배	한국어의 심각한 비효율성: 같은 의미인데 5배 많은 토큰 원인: 영어 기반 토크나이저가 한국어를 잘게 쪼갬 실무 영향: 한국어는 처리 속도 느리고 정확도 감소 결론: 한국어 전용 모델이 필요한 이유
단일: technology 멀티: ['technology', 'health', 'fitness']	단일 모드: 가장 확률 높은 하나만 강제 선택 멀티 모드: 0.5 이상인 모든 라벨 선택 가능 "AI fitness app"의 특성: 실제로 3개 영역 모두에 해당 실무 활용: 블로그 태그, 상품 카테고리 등은 멀티 모드가 적합

.BART-MNLI에서 중요한 옵션들

do_sample 같은 생성 옵션은 없고, 분류 방식과 라벨 설계가 핵심입니다

multi_label (가장 중요!)
업무 특성에 맞는 모드 선택
	# 단일 라벨: 하나만 강제 선택
	multi_label=False # 기본값
	# 멀티 라벨: 여러 개 동시 선택 가능
	multi_label=True

hypothesis_template
도메인에 맞는 문장 형태
- 다음페이지에 샘플있음
	# 기본 템플릿
	"This example is {}."
	# 커스텀 템플릿
	hypothesis_template="This text is about {}."
	hypothesis_template="The sentiment is {}."
	hypothesis_template="This belongs to {} category."

임계값 (threshold)
멀티 라벨에서 품질 관리
	# 멀티 라벨에서 몇 점 이상만 선택할지
	# 코드에서 직접 필터링
	selected = [l for l, s in zip(labels, scores) if s > 0.5] # 0.5가 임계값

candidate_labels (라벨 설계,품질) **명확하고 구체적인 카테고리**	# 좋은 라벨: 구체적, 명확 ["technology", "healthcare", "finance"] # 나쁜 라벨: 추상적, 애매 ["good", "bad", "other"]

max_length (간접 영향)	# 토크나이저 최대 길이 (텍스트 잘림 방지)
	tokenizer(..., max_length=512)

.

(참고) hypothesis_template은 NLI 가설 문장의 형태를 바꾸는 옵션

같은 내용도 질문 방식에 따라 답이 달라질 수 있음.
-> 도메인에 맞는 자연스러운 질문 형태로 바꿔서 정확도를 높이는 기법입니다

(템플릿참고1: 분류결과 변화)

text = "This product costs $500" 는 템플릿 구성에 따라 결과가 다르게 나올수 있음.

1. 기본 템플릿	test_template(text, labels, None, "기본")
	"This product costs $500" + "This example is business" → AI 생각: "가격($500) 얘기하니까 business 맞네!" → business (0.812) ← 확신
2. 뉴스 템플릿 뉴스에서 "product + 가격" = 보통 신제품 출시 뉴스 신제품 출시 뉴스 = 주로 기술 관련 (스마트폰, 노트북 등) "This news is about technology"가 더 자연스러운 문장 실제 뉴스 패턴: "iPhone 14 costs $999" → 기술 뉴스 "New Tesla costs $50,000" → 기술 뉴스 "MacBook costs $1,200" → 기술 뉴스	test_template(text, labels, "This news is about {}.", "뉴스")
	"This product costs $500" + "This news is about technology" → AI 생각: "뉴스에서 product 얘기하면 보통 새로운 기술제품이지!" → technology (0.507) ← 절반 확신
3. 카테고리 템플릿	test_template(text, labels, "This belongs to {} category.", "카테고리")
	"This product costs $500" + "This belongs to business category" → AI 생각: "가격만으로 비즈니스 카테고리라고 하기엔... 좀 애매한데?" → business (0.458) ← 확신 없음

```
def test_template(text, labels, template, name):
    """템플릿별 분류 테스트"""
    if template:
        result = classifier(text, labels, hypothesis_template=template)
    else:
        result = classifier(text, labels)

    print(f"{name}: {result['labels'][0]} ({result['scores'][0]:.3f})")

# 사용 예시
text = "This product costs $500"
labels = ["business", "technology", "sports"]

test_template(text, labels, None, "기본")
test_template(text, labels, "This news is about {}.", "뉴스")
test_template(text, labels, "This belongs to {} category.", "카테고리")
```

텍스트: This product costs $500

기본: business (0.812)
뉴스: technology (0.507)
카테고리: business (0.458)

(템플릿참고2: 도메인 특화)- 템플릿이 AI의 해석 관점을 변경

"I've seen worse products" 문장의 미묘함 ("더 나쁜 제품들을 본 적 있다")
의미: "이 제품이 그나마 낫다" (간접적 긍정?)

템플릿별 해석:

1. 일반템플릿: test_template(text, labels, "This example is {}.", "일반")
   결과: "This example is negative" : "worse"라는 단어에 집중 → 부정적 (0.924)

2. 감정템플릿: test_template(text, labels, "The sentiment is {}.", "감정")
   결과: "The sentiment is negative": 감정적으로는 부정적이지만 확신 떨어짐 (0.570)

3. 리뷰템플릿: test_template(text, labels, "This review is {}.", "리뷰")
   결과: "This review is positive": 리뷰 맥락에서는 "상대적으로 좋다"는 의미로 해석 → 긍정! (0.440)

--> 이 코드에서 알수 있는건 같은 애매한 문장도 템플릿이 바뀌면:
다른 라벨 선택 (negative → positive)하며, 신뢰도가 크게 변화 (0.924 → 0.440)하면서
완전히 다른 해석을 할수 있다.

```
def test_template(text, labels, template, name):
    """템플릿별 분류 테스트"""
    if template:
        result = classifier(text, labels, hypothesis_template=template)
    else:
        result = classifier(text, labels)

    print(f"{name}: {result['labels'][0]} ({result['scores'][0]:.3f})")

# 부적절한 vs 적절한 템플릿
text = "The weather is sunny and warm today."
labels = ["pleasant", "unpleasant"]

print("\n=== 템플릿 적절성 비교 ===")
test_template(text, labels, "This person is {}.", "부적절")
test_template(text, labels, "This weather is {}.", "적절")
```

```
=== 도메인 특화 효과 ===
일반: negative (0.924)
감정: negative (0.570)
리뷰: positive (0.440)
```