**문맥 기반 문장 순서 예측 AI 경진대회 결과 분석 리포트**

AI를 위한 딥러닝 01분반

# 

# Team 1

# 김대호, 박상춘, 서정민, 여서연, 이혜원

**목차**

1. Introduction

2. Method

2.1 문장 순서 이진 분류의 결합(pairwise) → BERT

2.2 소프트맥스 이용 다중 분류 → T5

2.3 Global Method

2.4 모델 선택 배경 및 Global 방식 도입 이유

3. Experiments

3.1 BERT 모델

3.1.1 모델 구성 방식

3.1.2 klue/bert-base 결과

3.1.3 klue/roberta-base 결과

3.1.4 monologg/koelectra-base-discriminator 결과

3.2 T5 기반 Sequence Method 실험 및 결과

3.2.1 전처리 과정

3.2.2 모델 구성

3.2.2.1 사전학습 모델 선택

3.2.2.2 구현 및 학습 구조

3.2.3 학습 설정

3.2.4 실험 결과

3.3 ELECTRA 하이퍼파라미터 튜닝

3.3.1 튜닝 방법 구체적 소개

3.3.2 monologg/koelectra-base-v3-discriminator 튜닝 결과

3.4 KoElectra 모델 구조 개선

3.4.1 성능 개선의 필요성

3.4.2 개선 방향

3.4.3 개선 방향

3.5 Global 기반 RoBERTa 실험 및 결과

3.5.1 데이터셋 구성

3.5.2 모델 구조

3.5.3 학습 및 튜닝 전략

3.5.4 결과 분석

4. Conclusion

# 1. **Introduction**

우리 팀이 이번에 참여한 컴피티션은 DACON이 주최한 ‘문맥 기반 문장 순서 예측 AI 경진대회’이다. 이 컴피티션은 무작위로 섞인 4개의 한국어 문장을 가장 정확하게 배열할 수 있는 예측 알고리즘을 개발하는 것을 최종 목표로 한다. 데이터에는 test, train, submission 데이터 세트가 있다. Test 데이터 세트에는 1800개의 논리적 순서 없이 섞인 4개의 한국어 문장 쌍이 있다. 그리고 이를 훈련할 수 있게끔 정답 인덱스도 존재한다. 해당 데이터 세트로 훈련한 모델을 직접 검증할 수 있는 train 데이터 세트에는 총 7,300개의 문장 쌍이 존재하며 이때 test 데이터 세트와 달리 정답 인덱스가 존재하지 않는다. 그리고 submission은 정답을 제출할 때 어떤 식으로 제출하면 되는지 예시를 보여주는 파일이다.

데이터 세트에서 알 수 있듯이 우리는 4개의 무작위로 섞인 문장들을 정확한 순서로 재배열해야 한다. 즉, 수업 시간에 배운 두 개의 문장을 배열하는 것보다 더 다양한 경우의 수를 고려하여 4개의 문장 쌍을 배열해야 한다. 따라서 문제에 접근하는 데 있어 이중분류와 다중 분류를 이용하기로 계획했다.

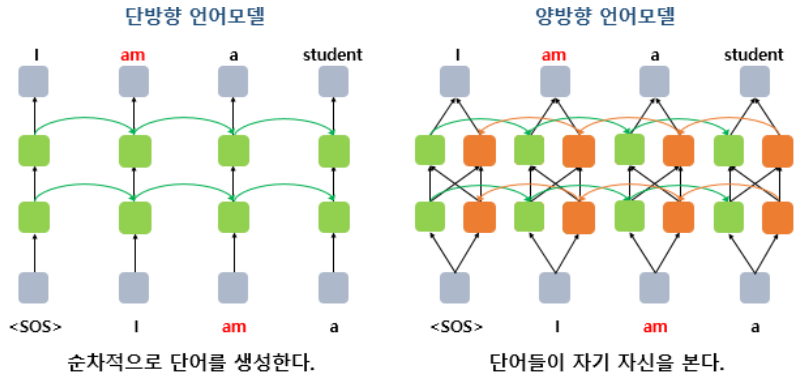
궁극적으로 우리는 이 데이터 세트를 이중분류와 다중 분류를 통해 받아 가장 자연스러운 문장 순서를 정답 인덱스 배열로 예측하는 AI 알고리즘을 개발하는데 최종 목표를 뒀다. 이 과정에서 우리는 정확도와 F1 점수를 최대한으로 올리고 과적합을 방지하고자 노력했다. 이는 다음으로 소개할 접근방법과 모델과의 조합, epoch 수 등을 세밀히 조합함으로써 목표에 근접하게 도달했다.

# **2. Method**

문장 순서 예측 문제는 일반적인 문장 분류나 생성 문제와 달리, 여러 문장 간의 논리적 연결성과 순차적 흐름을 동시에 고려해야 하는 복합적인 문제이다. 기존 사전학습 언어모델인 BERT와 T5는 각각의 특성을 가지고 있었지만, 본 문제에 직접적으로 대응하기 위해서는 문제 구조에 맞는 추가적인 접근 방식 설계가 필요하였다.

## **2.1 문장 순서 이진 분류의 결합(pairwise) → BERT**

문장 순서 예측 문제를 해결하기 위해, BERT 모델을 활용한 Pairwise 방식은 전체 순서를 직접 예측하기보다는 문장 쌍 간의 상대적 순서를 이진 분류로 판단하는 구조를 따른다. 4개의 섞인 문장으로부터 총 6개의 문장쌍을 생성하고, 각 쌍에 대해 문장 A가 문장 B보다 앞서는지를 예측한 뒤, 이 결과를 조합하여 24개의 문장 순열 중 가장 높은 점수를 갖는 순서를 최종 결과로 선택한다.

각 문장쌍은 BERT 입력 형식인 [CLS] 문장 A [SEP] 문장 B [SEP] 형태로 구성되며, BERT 모델의 양방향 인코딩을 통해 문맥 정보를 추출하고, 0 또는 1의 클래스로 상대 순서를 분류한다. 이러한 방식은 단일 쌍 간의 관계를 잘 반영할 수 있어 국소적 판단에 강점을 가지며, NSP(Next Sentence Prediction) 사전학습 구조와 유사한 설정으로 인해 사전학습된 BERT 계열 모델의 성능을 효과적으로 활용할 수 있다.

이러한 접근은 비교적 간결하게 문제를 이진 분류 형태로 단순화할 수 있다는 장점이 있지만,

모든 문장쌍을 개별적으로 판단하기 때문에 문장 전체의 논리 흐름이나 맥락 정보를 온전히 반영하지 못할 수 있으며, 24개의 순열을 모두 평가해야 하므로 추론 시간이 길고 계산 비용이 높다는 단점도 존재한다.

### 

## **2.2 소프트맥스 이용 다중 분류 → T5**

T5(Text-to-Text Transfer Transformer) 모델은 입력과 출력을 모두 텍스트로 처리하는 구조로,

문장 순서 예측 문제를 자연어 생성 문제로 변환하여 해결할 수 있다. 본 프로젝트에서는 4개의 섞인 문장을 하나의 문자열로 연결한 후, 올바른 문장 순서를 [2 0 1 3]과 같은 인덱스 시퀀스로 예측하도록 구성하였다.

입력은 네 문장을 구분 기호 없이 공백으로 연결한 형태로 제공되며, 출력은 예측된 문장 순서를 나타내는 문자열이다. 예를 들어, "문장1 문장2 문장3 문장4"가 입력으로 들어가면,

출력은 "2 0 1 3"과 같은 형식으로 생성된다. 이는 T5의 decoder가 직접적으로 순서를 생성하게 하는 구조이며, 출력 결과는 softmax를 통해 선택된 각 인덱스에 대응하는 예측 값으로 간주된다.

이 방식은 입력 전체를 한 번에 모델에 전달하므로 문맥 전체의 흐름을 반영할 수 있다는 점에서 강점이 있으며, 출력 결과가 직관적이고 해석 가능하다는 장점이 있다. 또한, 사전학습된 T5 모델의 구조적 특성을 그대로 활용할 수 있다는 점에서도 구현 효율성이 높다.

하지만 모델이 직접 인덱스를 생성해야 하기 때문에 문장 간의 상대적 위치를 학습하는 데 어려움이 있고, 출력 형식이 조금만 달라져도 모델이 예측을 제대로 수행하지 못하는 경우가 발생했다. 또한 입력 문장이 길어지면 GPU 메모리 부담이 커지고, 학습이 불안정하거나 과적합이 나타나는 경우도 있었다.

## **2.3 Global Method → BERT**

문장 순서 예측 문제에 대해 Global 모델은 모든 문장을 동시에 입력 받아 각 문장의 올바른 위치를 예측하는 방식을 사용하였다. 기존의 pairwise 방식이 문장쌍 간 상대적 순서를 예측하는 것과 달리, 본 방식은 절대적인 문장 위치를 예측한다는 점에서 문맥 전반을 한 번에 파악하는 이점을 가진다.

모델 아키텍처로는 klue/roberta-large를 기반으로 하였다. 입력은 4개의 섞인 문장을 “[CLS] S0 [SEP] S1 [SEP] S2 [SEP] S3 [SEP]” 형태로 결합하여 하나의 문장 시퀀스로 구성하였으며, RoBERTa tokenizer를 통해 전처리하였다. 모델은 각 문장에 대해 위치를 0~3으로 예측하며, 총 16개의 로짓(4문장 × 4포지션)을 출력한다. 이 로짓을 기반으로 각 문장의 위치를 결정하고 전체 순서를 재구성한다.

출력층은 다음과 같이 구성하였다:

* 입력 시퀀스를 통해 생성된 [CLS] 토큰 임베딩과 전체 토큰의 평균 임베딩을 평균하여 pooled representation을 생성
* 해당 representation을 dropout 및 feedforward layer에 통과시켜 최종적으로 16차원 로짓 출력
* 손실 함수는 “CrossEntropyLoss()”를 사용하여 예측된 위치와 실제 위치 간 오차를 계산

이 방식은 모든 문장을 동시에 고려하기 때문에 문장 간 전반적인 논리 흐름과 전후 맥락 파악에 유리하다는 장점이 있다.

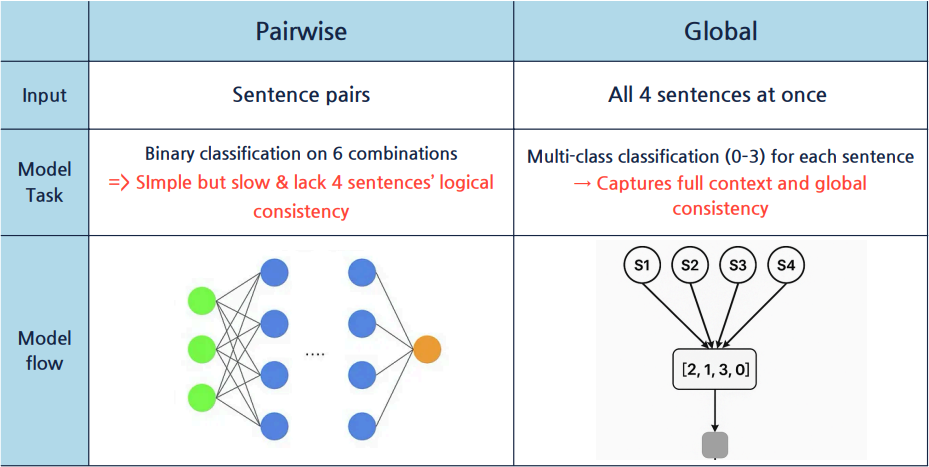
## **2.4 모델 선택 배경 및 Global 방식 도입 이유**

문장 순서 예측 문제에 대응하기 위해, 우리는 초기 단계에서 BERT 기반 Pairwise 방식과 T5 기반 Sequence 방식을 병행하여 실험하였다.

T5는 전체 문장을 입력받아 순서를 직접 생성하는 구조로, 이론적으로 문맥 순서를 맞추기에 가장 적합했지만, 실험 결과 상대 위치 학습의 어려움, 출력 포맷의 민감성, 학습 불안정성 등으로 인해 정확도와 실험 효율성에서 일관된 성능을 내기 어려웠다.

반면, BERT 기반 Pairwise 방식은 문장쌍을 이진 분류 문제로 구성해 보다 안정적인 학습 흐름과 높은 정확도를 기록하였다. 이로 인해 우리는 두 방식 중 BERT 계열 방식이 이 문제에 더 적합하다는 판단을 내리고, 후속 실험을 BERT 기반 구조에 집중하게 되었다.

그러나 Pairwise 방식은 각 문장쌍을 독립적으로 예측한 뒤, 24개의 가능한 순열을 평가하는 방식이기 때문에, 전체 문맥의 논리 흐름을 정합성 있게 반영하는 데에는 구조적 한계가 있었다. 이에 따라 우리는 BERT의 강점을 유지하면서도 전체 문장을 한 번에 입력받아 각 문장의 절대적 위치를 직접 예측하는 Global 방식을 새롭게 설계하였다. 이 방식은 문장 간 전후 관계와 맥락을 모두 반영할 수 있는 구조였으며, 실험 결과에서도 가장 높은 정확도와 일관된 성능을 보여주어 본 프로젝트의 최종 모델로 선정되었다.



# 

# **3. Experiments**

## **3.1 BERT 모델**

### **3.1.1 모델 구성 방식**

BERT 계통 모델은 pairwise(쌍별) 접근법을 사용하기 때문에, 기존 데이터 프레임에서 뽑아낸 문장쌍을 input으로 해야한다.

#### **3.1.1.1 pairwise\_preprocess 함수**

다음과 같은 순서를 통해 input 문장을 전처리하였다.

**1. 문장 리스트 추출**: 데이터 프레임의 sentences\_0 ~ sentences\_3 값들을 한 row씩 리스트로 저장

**2. 정답 순서대로 재배열**

: answer\_0 ~ answer\_3의 값을 인덱스로 이용하여 1.의 결과 리스트를 정답 순서에 맞게 배열

**3. 정답 순서 문장쌍(positive pair) 추출**예: [A, B, C, D] → (A,B), (B,C), (C,D)

**4. 모든 문장쌍 생성**: 4개 문장으로 만들 수 있는 모든 순서쌍 (4P2 = 12개)

**5. 각 문장쌍에 대해 label 부여**

* 정답 순서에 맞는 쌍이면 label = 1
* 그 외 모든 쌍은 label = 0

이 과정을 통해 학습 가능한 pairwise 데이터셋이 완성된다.

### **3.1.1.2 SentencePairDataset 클래스**

SentencePairDataset 클래스는 문장쌍과 라벨을 BERT 입력 포맷에 맞게 변환해주는 데이터 클래스이다. 주요 구성은 다음과 같다.

* \_\_init\_\_: 데이터 및 토크나이저 초기화
* \_\_len\_\_: 데이터 샘플 수 반환
* \_\_getitem\_\_: 특정 인덱스의 문장쌍과 라벨을 토큰화 및 패딩하여 BERT 입력값으로 변환

출력은 input\_ids, attention\_mask, labels (필요 시 token\_type\_ids)로 구성되며, 모델 입력 형식에 부합하도록 처리되었다.

#### **3.1.1.3 SentencePairModel 클래스**

모델 구조는 다음과 같다.

* Input Encoding: [CLS] 문장1 [SEP] 문장2 [SEP] 형태로 인코딩
* BERT Encoding: 총 13개 hidden state 중 마지막 4개를 평균
* Representation Extraction: 평균낸 [CLS] 벡터를 기반으로 Dropout → Linear Layer를 거쳐 클래스 확률 출력
* Classification & Training: CrossEntropyLoss 사용하여 이진 분류 모델 훈련 및 평가

이 구조는 직후 사용할 BERT 계열 모델 3가지에 공통적으로 적용 가능한 문장쌍 분류 아키텍처이다.

#### **3.1.1.4 하이퍼파라미터 설정**

모델 비교 실험의 객관성을 유지하기 위해, 모든 모델에 동일한 최소 수준의 하이퍼파라미터를 적용하였다. 설정값은 다음과 같다.

* 학습 epoch: 3
* Train batch size: 16
* Evaluation batch size: 64
* Warmup step: 500
* Weight decay: 0.01
* Early Stopping: Accuracy 기준, patience=2

### **3.1.2 klue/bert-base 결과**

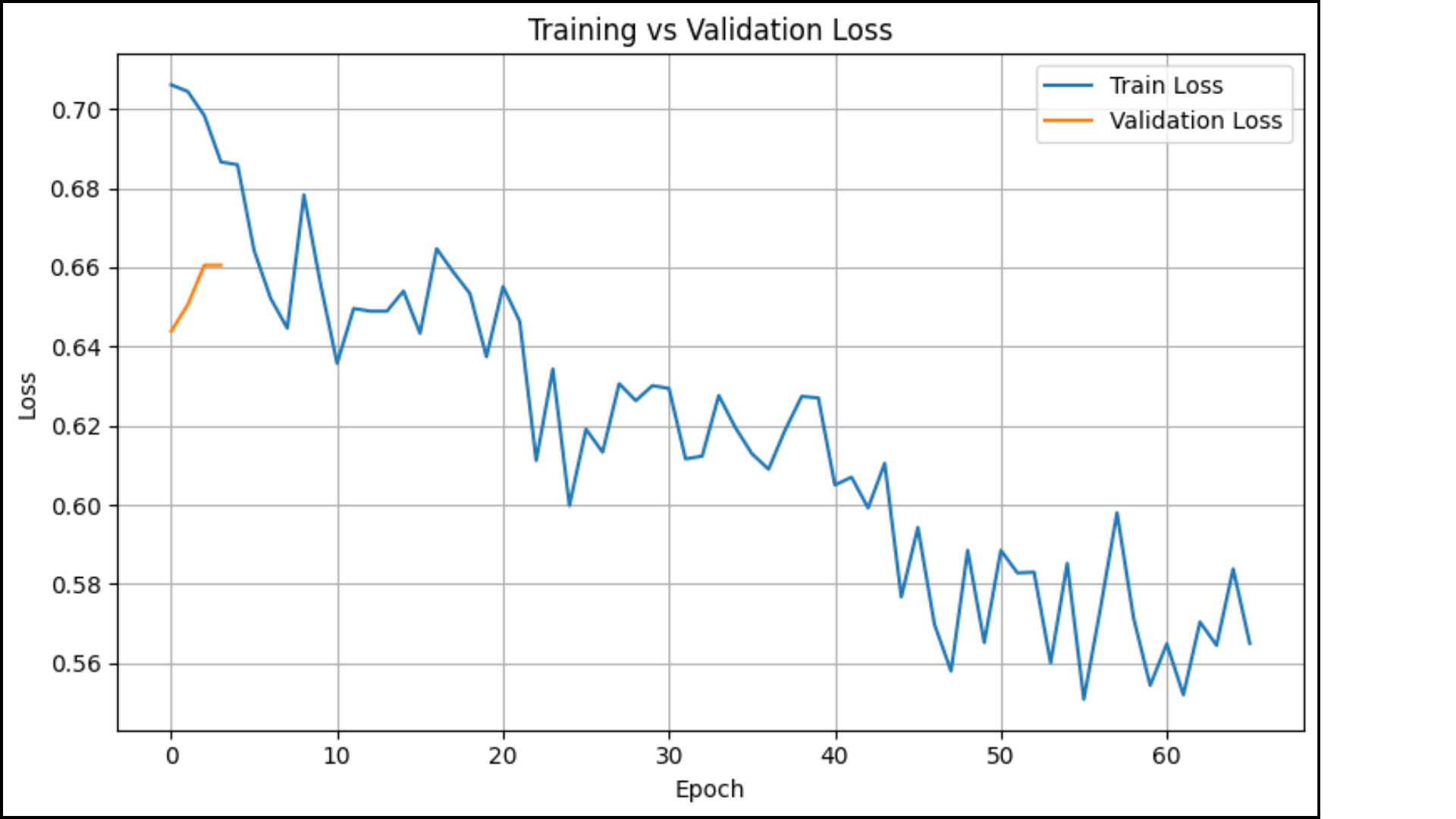
링크: <https://huggingface.co/klue/bert-base>  
  
 KLUE에서 제공하는 klue/bert-base 모델을 사용한 실험 결과는 다음과 같다.

* **Accuracy**: 66.53%
* **Eval Loss**: 0.6354

사전학습된 모델을 기반으로 문장 간 순서 관계를 어느 정도 반영하였으나, 다른 모델과 비교 시 상대적으로 낮은 성능을 보였다. 이는 NSP 기반의 사전학습 한계 또는 작은 batch size의 영향으로 해석될 수 있다.

### **3.1.3 klue/roberta-base 결과**

링크: <https://huggingface.co/klue/roberta-base>

klue/roberta-base 모델은 NSP 없이 문장 수준의 범용적 문맥 이해에 집중된 사전학습 모델로, 다음과 같은 성능을 나타냈다.

* **Accuracy**: 88.90%
* **Eval Loss**: 0.4405

가장 일반적인 BERT 모델 대비 약 22%p 향상된 정확도를 기록하였으며, 문장쌍의 순서를 예측하는 데 더 효과적인 구조임을 보여준다.

### 

### 

### **3.1.4 monologg/koelectra-base-discriminator 결과**

링크: <https://huggingface.co/monologg/koelectra-base-discriminator>  
  
monologg/koelectra-base-discriminator는 Generator-Discriminator 방식의 Pre-training을 거친 ELECTRA 구조이다. 실험 결과는 다음과 같다.



* **Accuracy**: 89.27%
* **Eval Loss**: 0.3371

세 모델 중 가장 높은 정확도를 보였으며, 입력 문장의 섬세한 의미 차이를 구별하는 데에 매우 효과적인 성능을 보였다. 이는 ELECTRA 구조가 학습 효율성 및 문맥 이해에서 강점을 지닌다는 점을 뒷받침한다.

## 

## **3.2 T5 기반 Sequence Method 실험 및 결과**

### **3.2.1 전처리 과정**

Sequence 방식에서는 4개의 문장을 하나의 입력으로 결합하여 모델이 문장 순서를 예측할 수 있도록 학습시킨다. 이를 위해 다음과 같은 전처리 과정을 거쳤다. 우선 각 데이터 샘플의 4개 문장을 [SEP] 토큰 없이 단순 문자열로 이어붙여 하나의 문장열로 구성하였다. 정답 레이블은 해당 문장들의 올바른 순서를 나타내는 인덱스 순서 문자열로 구성하였다(예: "3 1 4 2"). 그 후, 입력 문장과 정답 순서 문자열을 T5 모델의 형식에 맞춰 tokenization 하였고, padding은 손실 계산에서 제외되도록 -100으로 설정하였다. 이를 위한 커스텀 SentenceOrderDataset 클래스를 정의하여 \_\_init\_\_, \_\_len\_\_, \_\_getitem\_\_ 메서드를 구현하였다. \_\_getitem\_\_에서는 입력 문장과 라벨을 토크나이즈하고, 이를 tensor로 변환하여 반환한다.

### **3.2.2. 모델 구성**

본 프로젝트의 Sequence Method에서는 T5 계열의 사전학습 모델을 기반으로 하여 문장 순서를 예측하는 모델을 구성하였다. T5(Text-to-Text Transfer Transformer)는 모든 자연어 처리 과제를 텍스트 입력-텍스트 출력 형식으로 변환할 수 있도록 설계된 범용 모델이며, 본 과제처럼 주어진 문장들의 순서를 예측해야 하는 구조에도 자연스럽게 적용 가능하다.

#### **3.2.2.1 사전학습 모델 선택**

실험에는 다음 두 개의 한국어 기반 T5 사전학습 모델을 사용하였다:

* **paust/pko-t5-large** 
  + 링크:<https://huggingface.co/paust/pko-t5-large>
  + 한국과학기술원(KAIST)의 박건우 연구팀이 공개한 모델로, 원래 T5 구조를 한국어 데이터에 맞춰 대형 사이즈로 사전학습시킨 버전이다.
  + 대규모 파라미터 수(약 770M)를 바탕으로 높은 표현력을 가지며, 다양한 한국어 NLP 태스크에서 우수한 성능을 보인 바 있다.
* **wisenut-nlp-team/KoT5-base**
  + 링크:<https://huggingface.co/wisenut-nlp-team/KoT5-base>
  + 와이즈넛 NLP 팀에서 공개한 KoT5 시리즈 중 베이스 모델로, T5 구조를 기반으로 한국어 위키, 뉴스, 웹 문서 등을 대상으로 사전학습되었다.
  + 파라미터 수는 약 220M 수준으로, 비교적 경량이며 속도와 자원 효율 면에서 강점을 가진다.

두 모델은 모두 Huggingface Transformers 라이브러리에서 제공하는 T5ForConditionalGeneration 클래스를 통해 로딩 및 fine-tuning이 가능하다. 실험에서는 두 모델의 성능(정확도 및 손실 값)을 비교하여, 최종적으로 성능이 더 우수한 모델을 선택하였다.

#### **3.2.2.2 구현 및 학습 구조**

두 모델 모두 Huggingface의 기본 학습 구조(forward() 함수 및 Trainer API)를 그대로 활용하였다. 입력은 하나의 문자열로 결합된 문장열, 출력은 문장 순서 인덱스를 문자열 형태로 제공하였다(예: "3 1 4 2"). 입력과 출력은 tokenizer를 통해 tokenization되며, 토큰화 과정에서 최대 길이를 초과하는 경우 truncation을 적용하고, 부족한 경우에는 padding을 수행하였다. Padding token을 모델이 명시적으로 인식하지 못하는 경우, tokenizer.pad\_token\_id 및 model.config.pad\_token\_id를 수동으로 설정하였다. 손실 함수는 cross-entropy loss를 사용하며, padding 토큰은 손실 계산에서 제외하기 위해 라벨 상에서 -100으로 마스킹 처리하였다. 이러한 구조 덕분에 특별한 모델 구조 수정 없이도 텍스트 기반 문장 순서 예측 문제를 효과적으로 해결할 수 있었다. 두 모델의 비교 실험 결과는 다음 절에서 기술한다.

### **3.2.3 학습 설정**

T5 모델의 학습은 다음과 같은 설정으로 진행하였다.

* 학습 epoch: 10
* Train batch size: 4
* Warmup step: 1000
* Early Stopping: eval loss 기준, patience=2
* Learning rate: 2e-5
* Gradient accumulation step: 4
* Optimizer: AdamW
* Learning rate scheduler 사용
* Mixed precision training 사용
* Gradient clipping: max norm 1.0

### **3.2.4 실험 결과**

두 T5 모델의 성능을 비교한 결과는 다음과 같다.

| 모델 | Accuracy | Eval Loss |
| --- | --- | --- |
| paust/pko-t5-large | 0.7391 | 0.0941 |
| wisenut-nlp-team/KoT5-base | 0.7052 | 0.1444 |

paust/pko-t5-large 모델이 정확도와 손실 측면에서 더 우수한 성능을 보여 이후 실험에서는 해당 모델을 채택하였다.

## **3.3 ELECTRA 하이퍼파라미터 튜닝**

문장 순서 예측 성능을 극대화하기 위해, 본 프로젝트에서는 monologg/koelectra-base-v3-disc riminator 모델을 대상으로 **랜덤 서치(Random Search)** 기반의 하이퍼파라미터 튜닝을 수행하기로 결정하였다. 실험은 파라미터 간 상호작용 효과를 균형 있게 살피기 위해 팀원별로 탐색 범위를 분할하고, 각자 지정된 범위 내에서 무작위 조합 실험을 수행하는 방식으로 진행되었다.

### **3.3.1 튜닝 방법 구체적 소개**

#### **실험 설계 및 수행 방식**

하이퍼파라미터 튜닝은 사전에 정의된 범위 내에서 무작위로 조합을 샘플링하는 Random Search 방식을 기반으로 하였다. 각 팀원은 사전에 정의된 범위 내에서 무작위로 파라미터 조합을 추출하고 실험을 수행하였으며, 실험마다 다음과 같은 조건이 공통적으로 적용되었다.

* + **모델 구조**: monologg/koelectra-base-v3-discriminator
  + **최적화 목표**: Accuracy (accuracy 기준 Early Stopping)
  + **EarlyStoppingCallback** 적용  
    - load\_best\_model\_at\_end=True
    - metric\_for\_best\_model='accuracy'
    - greater\_is\_better=True

#### **팀원별 파라미터 설정 범위**

#### 총 5명의 팀원이 동일한 하이퍼파라미터 세트를 다루되, 범위를 분할하여 다양성을 확보하였다. 다음 표는 각 팀원별로 설정한 탐색 범위를 요약한 것이다.

| **팀원** | **Learning Rate** | **Weight Decay** | **Warmup Steps** | **Epochs** | **Train Batch Size** | **Eval Batch Size** | **Gradient Accumulation** | **Scheduler Type** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | [1e-5, 1e-4] | [0.0001, 0.1] | [200, 1000] | [3, 10] | [4, 8] | [8, 16] | [4, 8] | 'linear' |
| 2 | [1e-5, 1e-4] | [0.0001, 0.1] | [200, 1000] | [3, 10] | [8, 16] | [16, 32] | [2, 4] | 'linear', 'cosine' |
| 3 | [1e-5, 1e-4] | [0.0001, 0.1] | [200, 1000] | [3, 10] | [16, 32] | [32, 64] | [1, 2] | 'cosine', 'polynomial' |
| 4 | [1e-5, 1e-4] | [0.0001, 0.1] | [200, 1000] | [3, 10] | [32, 64] | [64, 128] | [1, 2] | 'cosine', 'polynomial' |
| 5 | [1e-5, 1e-4] | [0.0001, 0.1] | [200, 1000] | [3, 10] | [64, 128] | [128, 256] | [1, 1] | 'polynomial' |

* + **Learning Rate**와 **Weight Decay**는 loguniform 분포로부터 연속값을 샘플링
  + **Warmup Steps**, **Epochs**, **Batch Size** 등은 균등 분포 기반 정수형 샘플링
  + **Scheduler Type**은 각 팀원의 탐색 목적에 따라 고정 또는 2개 후보 중 무작위 선택

#### **평가 기준**

#### 각 trial의 성능은 **validation accuracy**를 기준으로 평가하였고, 그 외 보조 지표로 **eval loss**도 함께 기록하였다. 모든 trial에 대해 성능이 자동 기록되며, 가장 성능이 우수한 조합은 후속 분석에서 별도 적용되었다.

### **3.3.2 monologg/koelectra-base-v3-discriminator 튜닝 결과**

각 팀원이 수행한 실험 중 가장 성능이 우수한 trial의 결과는 다음과 같다.

* **최적 하이퍼파라미터 조합**
  + Learning Rate: 1.1921e-05
  + Weight Decay: 0.0219
  + Warmup Steps: 550
  + Epochs: 8
  + Train Batch Size: 63
  + Eval Batch Size: 98
  + Gradient Accumulation Steps: 1
  + Scheduler Type: cosine
* **Validation 성능**
  + Accuracy: **0.8943**
  + Eval Loss: **0.2992**

이는 기본 BERT 계열 모델 실험에서의 성능(accuracy: 0.8927, eval loss: 0.3371) 대비 정확도와 손실값 모두 개선된 결과이다. 이로써 하이퍼파라미터 튜닝이 모델 성능 향상에 실질적인 기여를 했음을 확인할 수 있었으며, ELECTRA 구조가 본 태스크에 매우 효과적인 사전학습 모델임을 검증하였다.

## **3.4 KoElectra 모델 구조 개선**

### **3.4.1 성능 개선의 필요성**

KoElectra는 기존 pairwise 실험에서 가장 높은 정확도를 보였지만, 성능이 일정 수준에서 정체되는 현상을 보였다. 이를 해결하기 위해 모델의 구조를 직접 수정하여 더 복잡하고 강력한 표현 능력을 갖도록 개선하였다.

### **3.4.2 개선 방향**

(1) Hidden Layer 추가 및 확장

* 기존 linear layer 하나만 사용하던 구조에 대해, 여러 개의 hidden layer를 도입하고 hidden size를 증가시켜 비선형 표현력을 강화하였다.

(2) Multi-Head Attention 도입

* 문장쌍 간의 다양한 관계를 병렬적으로 파악할 수 있도록 multi-head attention mechanism을 추가하였다. 이를 통해 모델이 각 문장의 다른 부분에 대해 동시에 다양한 시각으로 집중할 수 있도록 하였고, 문장 내 의미적 연결성 이해가 향상되었다.

(3) Layer Normalization 적용

* 각 layer 출력의 분포를 정규화함으로써 학습 안정성과 수렴 속도를 개선하였다. 이는 특히 deep한 구조에서 gradient vanishing/exploding 문제를 줄여주고 일반화 성능을 높이는 데 도움이 되었다.

### **3.4.3 개선 방향**

개선된 모델은 기존 KoElectra 모델보다 성능이 향상되어 다음과 같은 결과를 얻었다:

| 모델 | Accuracy | Eval Loss |
| --- | --- | --- |
| 기존 KoElectra | 0.8943 | 0.2992 |
| 개선된 KoElectra | 0.9019 | 0.2826 |

정확도는 90.19%까지 상승하였지만, 이 이상의 향상은 구조 개선만으로는 어렵다고 판단되어 이후 global 방식으로 전환하였다.

## **3.5 Global 기반 RoBERTa 실험 및 결과**

Global 모델 실험은 PyTorch를 기반으로 진행되었으며, klue/roberta-large 사전학습 모델을 사용하였다. 전체 문장 4개를 하나의 시퀀스로 결합한 후, 이를 입력으로 하여 각 문장이 최종 문서 내에서 어느 위치(0~3)에 위치해야 하는지를 예측하는 구조로 구성하였다.

### **3.5.1 데이터셋 구성**

4개의 문장을 무작위로 섞은 후, 이를 문자열로 이어붙여 모델 입력으로 사용하였다. 각 문장은 문서 내 위치(0~3)를 정답으로 갖는 **position vector**로 라벨링되며, 총 4개의 위치에 대한 정수형 라벨을 예측하도록 학습되었다. 이 과정을 처리하는 커스텀 Dataset 클래스는 다음과 같은 흐름으로 구성되었다:

* \_\_init\_\_: CSV 또는 리스트 형태의 데이터를 받아 tokenizer와 함께 저장
* \_\_getitem\_\_: 4개의 문장을 하나의 텍스트로 결합한 후 tokenizer를 적용
* 라벨이 있을 경우 “label = [2, 0, 1, 3]”과 같은 위치 벡터를 반환
* 최종적으로 input\_ids, attention\_mask, labels를 딕셔너리 형태로 출력

이 구조를 통해 각 배치마다 모델 입력에 필요한 정보들을 일관되게 구성할 수 있었으며, 문장 순서 정보를 효과적으로 학습할 수 있는 기반이 되었다.

### **3.5.2 모델 구조**

모델은 사전학습된 RoBERTa 모델의 마지막 hidden state 출력을 기반으로 두 가지 벡터를 추출하였다: [CLS] 토큰 벡터와 나머지 토큰들의 평균 벡터. 이 둘을 평균하여 pooled representation을 구성하고, 이를 분류기(classifier)에 입력하여 최종 예측을 수행하였다.

분류기는 총 세 층의 feedforward neural network로 구성되며, 각 층은 다음을 포함한다:

* Linear layer → LayerNorm → ReLU → Dropout
* 마지막 출력은 16차원 (4문장 × 4개 위치) 벡터

출력된 로짓은 shape (batch\_size, 4, 4)로 reshape되어 각 문장별 위치를 예측하며, 손실 함수로는 CrossEntropyLoss가 사용되었다. 학습 시 각 문장의 위치 예측을 독립적인 다중 분류 문제로 간주하여 처리하였다.

### **3.5.3 학습 및 튜닝 전략**

모델 학습에는 HuggingFace의 Trainer와 TrainingArguments를 사용하였다.

학습 전략은 다음과 같다:

* Optimizer: AdamW
* 손실 함수: CrossEntropyLoss (4문장 각각의 위치 예측에 적용)
* Evaluation Metric: Sentence Accuracy, Full Order Accuracy
* EarlyStoppingCallback: validation loss 기준, patience=2

하이퍼파라미터 튜닝은 scipy.stats.loguniform을 이용한 랜덤 탐색 기반으로 수행되었으며, 학습률, 배치 사이즈, warmup step, weight decay 등의 다양한 조합이 실험되었다. 최종적으로 선택된 최적 조합은 다음과 같다:

* Learning Rate: 1.5e-5 ~ 3.5e-5 구간에서 loguniform 분포를 사용해 샘플링
* Weight Decay: 0.01 ~ 0.07 구간에서 샘플링
* Epochs: 정수형으로 9 ~ 12 범위에서 무작위 선택
* Warmup Step: 전체 학습 step 수의 약 2%로 설정
* Batch Size: 16(train), 64(eval)
* Scheduler: linear
* 모델 저장 기준: full\_order\_accuracy를 기준으로 가장 우수한 모델 저장

튜닝은 trial 번호 기준으로 자동으로 이어지며, 중간에 오류가 발생할 경우 즉시 중단되도록 설계되었다. 모든 결과는 로그로 누적되며, 가장 성능이 좋은 trial은 자동으로 출력된다.

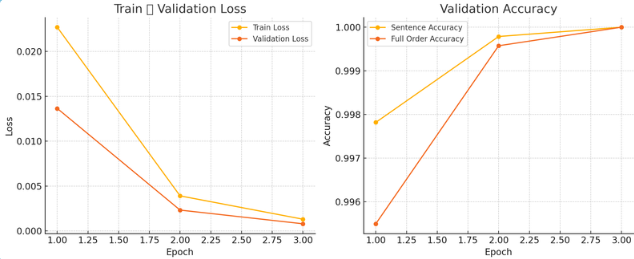
### **3.5.4 결과 분석**

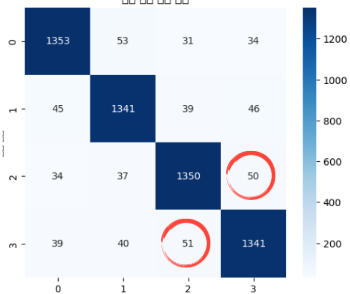
Global 모델은 전체 문장 시퀀스를 동시에 고려하는 방식으로 설계되어, 문장 간의 전반적인 논리적 일관성과 문맥을 효과적으로 포착할 수 있었다. 실제로 validation 데이터셋에서는 거의 완벽에 가까운 Full Order Accuracy를 달성하며 뛰어난 fitting 성능을 보였고, 모델의 아키텍처 및 학습 구조가 목적에 부합함을 확인할 수 있었다.

이 모델의 강점은 다음과 같다:

* 문장 전체를 한 번에 인식하므로순서 간 전후관계, 문맥 흐름을 전체적으로 파악 가능
* 입력 구조가 단순하며, 라벨도 명확하여 모델 구조가 직관적이고 확장성도 높음
* 검증셋에서 거의 100%에 가까운 성능으로, 학습 데이터 구조에 대한 적응 능력이 뛰어남

그러나 다음과 같은 한계도 확인되었다:

* 테스트셋에서의 성능이 84.27%로 하락, 명백한 **과적합(overfitting)** 현상이 발생함
* 문장 간 순서가 \*\*애매한 경우 (특히 2번 ↔ 3번 위치)\*\*에서 오류가 빈번하게 나타났으며, 이는 confusion matrix 분석을 통해도 확인됨



* 모델이 확률적 불확실성을 표현하지 못하고, 항상 확정적인 위치를 예측해야 한다는 구조적 제약이 존재

향후에는 다음과 같은 보완 방안이 고려될 수 있다:

* 순서 혼동이 잦은 특정 위치쌍(예: 2↔3)을 중심으로 한 Focused Fine-Tuning 또는 curriculum learning 전략 적용
* Pairwise 방식과의 하이브리드 구조를 통해 local-consistency와 global-consistency를 모두 강화
* 데이터 증강(augmentation)을 통해 다양한 문장 조합 패턴을 학습하도록 유도
* 앙상블 기법을 도입하여 다양한 관점에서의 순서 예측을 통합하는 방식

# 

# **4. Conclusion**

이번 프로젝트에서는 문장 순서 복원 문제를 해결하기 위해 세 가지 접근법을 실험하였다: 문장쌍을 비교하는 Pairwise 방식(BERT 기반), 전체 문장 시퀀스를 직접 생성하는 Sequence 방식(T5 기반), 그리고 문장들의 절대적 위치를 동시에 예측하는 Global 방식(RoBERTa 기반)이 그것이다.

**Pairwise** 방식은 문장 간의 로컬 관계를 정밀하게 포착하는 데 강점을 보였으며, 특히 KoElectra와 같은 한국어 특화 모델을 사용할 경우 안정적인 성능을 기록하였다. 다만 문장 전체의 논리적 흐름을 반영하기에는 구조적으로 한계가 있었고, inference 과정에서 가능한 모든 순서를 조합해 평가해야 하는 계산량 부담도 존재하였다.

**Sequence** 방식은 입력 문장을 하나의 시퀀스로 보고 직접 순서를 생성하는 방식으로, 자연어 처리 태스크와 유사한 직관적 구조를 제공하였다. 그러나 **T5** 모델이 긴 입력에 취약하고, 문장 간 미묘한 순서 차이를 구분하는 데 어려움이 있어 가장 낮은 성능을 기록하였다.

**Global** 방식은 문장 4개를 동시에 입력하여 각각의 위치를 예측하는 구조로, 전체 문맥을 고려한 포지셔닝에 매우 효과적이었다. validation 데이터셋에서는 Full Order Accuracy 기준 거의 100%에 가까운 정확도를 보였고, DACON 리더보드 기준 최고 성능(0.8427)을 기록하였다. 하지만 테스트셋에서는 2-3번 문장 위치 혼동이 자주 발생했고, overfitting 문제도 관찰되었다.

| **접근 방식** | **모델** | **특징** | **Score (DACON 기준)** |
| --- | --- | --- | --- |
| Pairwise | BERT / KoElectra | 문장쌍 비교, 로컬 순서 판단 우수 | 0.7809 |
| Sequence | T5 | 문장 전체를 시퀀스로 예측, 직관적 | 0.7258 |
| Global | RoBERTa | 전체 문장을 한 번에 보고 포지션 예측 | **0.8427** |

종합적으로 볼 때, Global 방식이 현 시점에서 가장 효과적인 접근으로 확인되었으며, 향후에는 이 구조를 기반으로 pairwise 정보 보완, 순서 혼동 영역에 대한 집중 학습, 앙상블 및 증강 전략을 접목하는 것이 주요 개선 방향이 될 것이다. 동시에 각 접근법이 갖는 구조적 특성과 한계를 명확히 이해하고, 문제 특성에 따라 적절히 조합해나가는 설계 전략이 필요함을 확인하였다.