# AIFFEL_DLthon_DKTC_online16 : 위협 대화 탐지 모델링

## 팀장이 왕이다

**팀장 : 김태경　　팀원 : 안수호, 신대웅, 박성현, 서희망**

### 문제 정의

- 대화의 성격을 **위협 세부 클래스 4개 또는 일반 대화 중 하나**로 예측하는 과제
- **학습 데이터**는 '협박', '갈취', '직장 내 괴롭힘', '기타 괴롭힘' 등 4개 클래스 각 1,000개 내외로 구성
- **테스트 데이터**는 '협박', '갈취', '직장 내 괴롭힘', '기타 괴롭힘', '일반 대화' 등 5개 클래스 각 100개로 구성
  - train data에는 없지만, test data는 일반 대화 클래스가 존재합니다. 5개 문장을 분류할 수 있게 train data에 일반 대화 데이터셋을 추가합니다.
  - **4개의 위협 세부 클래스**는 Augmentation만 가능(새로운 데이터 추가/생성 불가)
  - **일반대화 클래스**는 **합성데이터**로 구성
    - 다양한 프롬프트로 문장을 생성하고 학습에 활용
    - 최종 결과물로 제출할 수 있는 건 합성데이터 기반의 성능뿐입니다.
      - 합성데이터 생성 및 활용(필수)
      - 기 확보된 데이터 활용(AI hub 등 활용, 추가실험)
  - 학습 결과를 확인하며 성능을 높이는데 영향을 미치는 요소는 무엇이 있는지, 어떻게 수정해야하는지 고민합니다.
    - 위 기준에서 벗어나지 않는 범위 내에서 데이터셋의 구성은 자유입니다. 성능을 비교/기록해보세요 :)
      - 실험 결과를 Ablation study형식으로 기록합니다

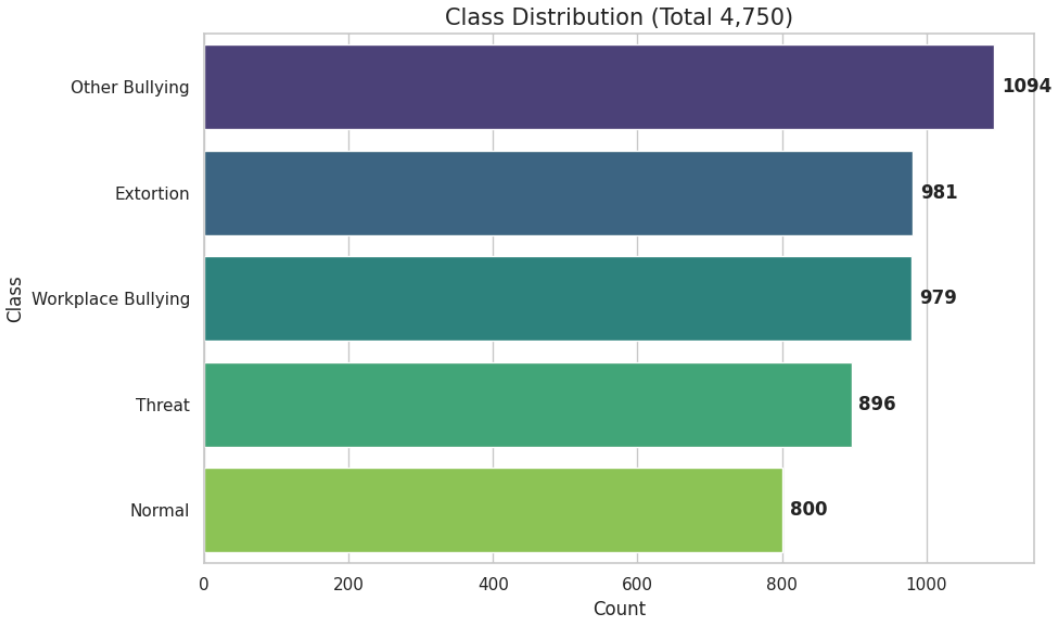
**프로젝트 평가 항목**

- 데이터 EDA와 전처리를 적절히 수행했는가?
- 모델 선정 근거가 타당한가?
- 모델의 성능/학습 방향을 판단하고 개선을 시도한 기준이 논리적인가?
- 결과 도출을 위해 다양한 시도를 했는가?
- 도출된 결론에 충분한 설득력이 있는가?
- 발표 자료가 청자의 입장에서 잘 정리되어있는가?
- 발표가 매끄럽게 진행되었고 발표시간을 준수하였는가?

### 데이터 EDA 및 전처리

#### 📊 1. 데이터 분포 분석 (Data Distribution)

- 분석 목적: 클래스 간 불균형 정도를 파악하기 위함
- 분석 결과
  - 전체 데이터 수: 기존 Train 데이터(3950개) + 추가데이터(800개) = 4750개
  - 클래스별 분포
    - `기타 괴롭힘`: 1,094개 (23.0%)
    - `갈취`: 981개 (20.6%)
    - `직장 괴롭힘`: 979개 (20.6%)
    - `협박`: 896개 (18.9%)
    - `일반 대화`: **800개 (16.8%)**




→ 총 800건의 일반 대화 데이터를 직접 합성하여 학습 데이터셋에 추가

(Normal Class 외 Class 기준 각 200개)

→ 전체 데이터의 약 17%가 일반 대화로 구성

→ 총 10번 내외의 대화 턴으로 구성

→ `금전대화` `업무대화` `거절대화` `교우관계대화` 위주로 일반 대화 생성

1. 💰 금전 대화 (갈취와 대조)
   - **상황**: 돈을 빌리는 상황이지만, 강제성 없고 정중하게 부탁하고 거절/수락하는 흐름.
   - **포인트**: '**뒤져서 나오면**', '**내놔**' 대신 **'부탁해', '계좌이체'** 사용.
     ```
     A: 다름이 아니라 내가 지금 지갑을 두고 나와서.
     B: 저런, 많이 곤란하겠네. 식사는 했어?
     A: 아니 아직 못 했어. 혹시 만 원만 빌려줄 수 있어?
     B: 지금 당장 현금은 없는데 어떡하지.
     ```
2. 🏢 업무 대화 (직장 내 괴롭힘과 대조)
   - **상황**: 상사가 부하 직원의 실수를 지적하지만, 인격 모독 없이 업무적으로 피드백 함.
   - **포인트**: '**길동씨**', '**머리**', '**사표**' 대신 **'김 대리', '수정', '확인'** 사용.
   - **금지어 회피**: `정말 죄송합니다` (과도한 사과) → `아, 제가 놓쳤네요` (일반적 사과).
     ```
     A: 아까 제출한 기획안 검토해봤는데 수정이 좀 필요해요.
     B: 아, 혹시 어떤 부분이 부족했나요?
     ```

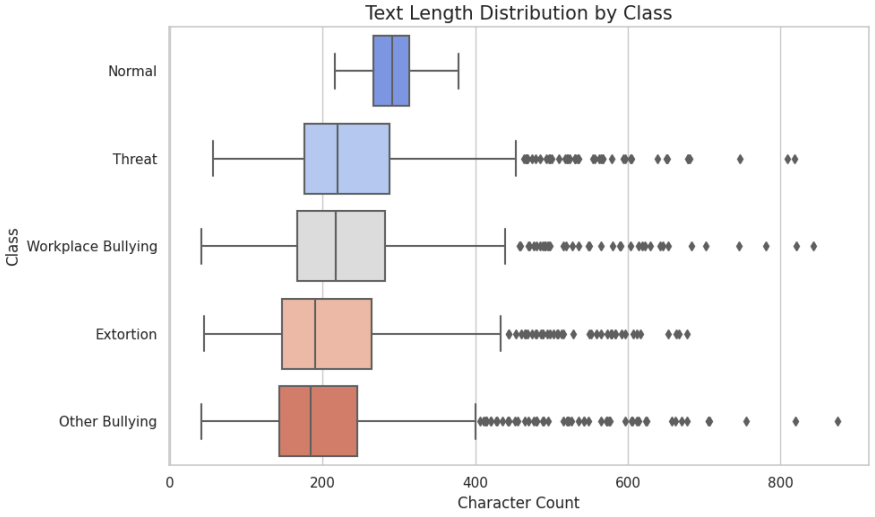
```
A: 3페이지 예산 데이터가 작년 기준으로 되어 있더라고요.
B: 아, 제가 최신 파일 업데이트를 놓쳤네요. 죄송해요.
```

3. 🚫 거절 대화 (협박과 대조)
   - **상황**: 만나자고 요구하지만, 상대방이 거절하고 제안자가 이를 쿨 하게 받아들이는 상황.
   - **포인트**: '**죽일거야**', '**가족**' 대신 **'아쉽네', '다음에 보자'** 사용.
```
A: 오랜만에 얼굴 보고 저녁이나 같이 먹을까 해서.
B: 아, 미안해서 어쩌지. 나 오늘 선약이 있어.
A: 그래? 아쉽네. 중요한 약속이야?
B: 응, 부모님이랑 식사하기로 했거든.
```

4. 🏫 교우 관계 (기타 괴롭힘과 대조)
   - **상황**: 물건을 빌리거나 부탁하지만, 비하 발언이나 강요가 없는 평범한 대화.
   - **포인트**: '**찐따**', '**냄새**', '**셔틀**' 대신 **'체육복', '다음 시간', '미안'** 사용.
```
A: 아, 깜빡하고 집에 두고 와서 빌릴 수 있나 했지.
B: 저런, 다음 교시 체육 선생님 무서운데 큰일이네.
A: 그러게 말이야. 다른 반 애들한테 물어봐야겠다.
B: 옆 반 준호한테 한번 연락해봐. 걔는 두 벌일걸?
```

## 📏 2. 텍스트 길이 분석 (Text Length Analysis)

- 분석 목적: 모델의 입력 시퀀스 길이 설정 및 데이터 특성 파악을 위함
- '일반 대화'가 위협 대화보다 평균 30% 이상 김
- 모델의 입력 길이를 너무 짧게 잡으면 →일반 대화의 핵심인 **'설명과 맥락'이 잘려나가 위협으로 오해**받을 수 있음
- 분석 결과
  - 전체 평균 길이: 235자 (최대: 874자, 최소: 41자)
  - 클래스별 평균 길이:
    - `일반 대화`: 301자
    - `협박`: 246자
    - `직장 괴롭힘`: 237자
    - `갈취`: 216자
    - `기타 괴롭힘`: 213자




- 특징:
  - `일반대화` 는 비교적 논리 정연함

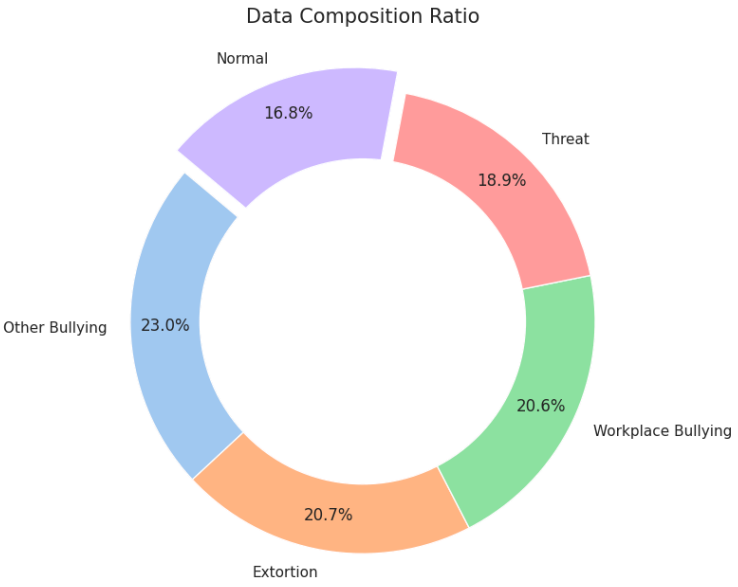
- `기타 괴롭힘` 이나 `갈취` 는 단답형 욕설이나 강압적 명령조로 인하여 길이가 짧음

## 🧹 3. 데이터 정제 과정 (Data Cleaning)

- 중복 데이터 제거 (De-duplication)
  - 모델의 과적합 방지를 위해 전량 제거
- 결측치 제거 (Missing Values)
  - 별도 제거 작업 불필요(결측치 **0건** 확인)
- 정규화
  - 특수문자나 반복되는 문자 하나로 정규화
  - ㅋㅋㅋㅋ → ㅋ , !!!!! → ! 특수문자나 반복되는 문자 하나로 정규화

## 🔗 4. 외부 데이터 병합 (External Data Merging)

- 수행내용
  - 기존 학습 데이터(`train.csv`): **3,950개** (4개의 클래스)
  - 합성 데이터(normal_conversation.csv): **800개** (일반 대화)
- 방법
  - 두 데이터 셋을 `concat` 하여 통합 데이터셋 구성
  - `일반대화` 클래스를 `class: 4` 로 배정




## 모델 선정 방법

- BERT 기반 모델을 선택한 이유

| 모델명 | 구조 | 원거리 단서 결합 | 적은 데이터에서의 성능 | 학습 및 추론 속도 | 최종 선택 | 선정 이유 |
|---|---|---|---|---|---|---|
| **BERT** | Self-Attention 기반 | 매우 강함 | 높음 (Pretrain 있음) | 느림 | O | 문장 전체 토큰을 동시 학습하여 LSTM과 GRU보다 한국어 언어적 맥락을 더 잘 이해 |
| **LSTM** | 순차적 게이트 구조 | 점점 약해짐 | 낮음 | 빠름 | | |
| **GRU** | LSTM의 단순화 | 점점 약해짐 | 낮음 | 빠름 | | |

- 총 네 가지 모델을 고려했으며 최종적으로 F1 Score가 가장 높은 KoBigBird 모델을 선택

| 모델명 | 모델 특징 | F1 Score | 최종 선택 |
|---|---|---|---|
| **KoBigBird** | **학습 데이터:** 뉴스, 위키 등 정제된 표준어 말뭉치<br>**주요 강점**: **Sparse Attention** 기술로 BERT의 길이 제한을 극복한 장문 전문가 | 0.67 | O |
| **KoELECTRA** | **학습 데이터:** 뉴스, 위키 등 정제된 표준어 말뭉치<br>**주요 강점**: BERT보다 효율적인 학습 방식(RTD)으로 높은 정확도와 속도 제공 | 0.62 | |
| **KCELECTRA** | **학습 데이터:** 포털 뉴스 댓글 등 실제 사용자의 비정형 데이터**신조어,**<br>**주요 강점**: **오탈자, 줄임말** 등 거친 구어체 데이터 처리에 최적화 | 0.66 | |
| **KIUE-ROBERTa** | **학습 데이터:** 뉴스, 위키, 국민청원 등 정제된 한국어<br>**주요 강점**: 동적 마스킹(Dynamic Masking)**과 **NSP 제거**를 통해 BERT보다 훨씬 정교한 문맥 파악 가능 | 0.63 | |

## 모델 성능 개선 방법

1. 방법1 : Attention 설정

```
# ==========================================
# 4. 모델 & 토크나이저 (KoBigBird)
# ==========================================
# 🚨 [변경됨] KoBigBird 모델 사용
MODEL_NAME = "monologg/kobigbird-bert-base"
print(f"🔄 모델 로딩 중: {MODEL_NAME}")

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

config = BigBirdConfig.from_pretrained(MODEL_NAME)
config.attention_type = "original_full"
config.num_labels = 5

model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME, config=confi
g).to(device)

def preprocess_function(examples):
    return tokenizer(examples["conversation"], truncation=True, max_length=512)

tokenized_train = train_dataset.map(preprocess_function, batched=True)
tokenized_val = val_dataset.map(preprocess_function, batched=True)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

- Attention을 config.attention_type = "original_full"로 설정
- original_full(full attention)은 토큰간 연결성을 계산하여 입력 문장 내에서 단서들을 결합함으로써 특정 클래스에 분류하므로 적합한 설정으로 판단하였음

  Ex) "돈" + "안 하면" + "가족/회사에 알리겠다" → 갈취로 분류(협박을 수단으로 대가 요구)
- 짧은 문장(대부분 512 토큰 이하)이 많은 본 데이터셋 특성상 블록 sparse보다 안정적인 학습과 수렴으로 F1-Score 개선에 기여할 수 있음

2. 방법2 : 메모리 최적화 및 학습 안정화 설정

```
# ========================================
# 5. 학습 (Training)
# ========================================
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    acc = accuracy_score(labels, predictions)
    f1 = f1_score(labels, predictions, average='weighted')
    return {"accuracy": acc, "f1": f1}

training_args = TrainingArguments(
    output_dir="./results_bigbird",  # [변경됨] 저장 폴더
    learning_rate=2e-5,
    per_device_train_batch_size=8,   # [변경됨] 메모리 절약을 위해 8로 조정 (가능하면 16)
    per_device_eval_batch_size=8,
    gradient_accumulation_steps=2,   # 배치 8 * 2 = 실제 16 효과
    num_train_epochs=5,
    weight_decay=0.01,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    report_to="none"
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_val,
    data_collator=data_collator,
    compute_metrics=compute_metrics
)

print("\n🦅 KoBigBird 학습 시작! (시간이 조금 더 걸릴 수 있습니다)")
trainer.train()
```

- 최적의 지점인 5 Epochs를 최종 학습 단위로 선정 : 모델의 검증 정확도(Accuracy)가 1 Epoch 대비 5 Epoch에서 약 4.7%p 향상된 것을 확인(안정적으로 수렴하였음을 확인)한 바 에폭 중 가장 좋은 체크포인트를 최종 사용하게 해서 모델 성능(F1-Score) 개선에 기여
- F1-Score(Weighted) 함께 계산 : 정확도(Accuracy)뿐만 아니라 데이터 불균형이 있는 경우에도 모델의 성능을 객관적으로 파악할 수 있도록 설계

- 학습률을 “learning_rate=2e-5”설정 : 2e-5보다 크거나 작을 경우 발생할 수 있는 수렴과 학습이 느린 부작용을 개선하고 Loss의 진동(Loss가 출렁이거나 NaN 출력 또는 성능의 급격한 변화) 방지

3. 방법3 : 입력 길이 및 토크나이징

```
# ========================================
# 4. 모델 & 토크나이저 (KoBigBird)
# ========================================
# 🚨 [변경됨] KoBigBird 모델 사용
MODEL_NAME = "monologg/kobigbird-bert-base"
print(f"🔄 모델 로딩 중: {MODEL_NAME}")

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

config = BigBirdConfig.from_pretrained(MODEL_NAME)
config.attention_type = "original_full"
config.num_labels = 5

model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME, config=confi
g).to(device)

def preprocess_function(examples):
    return tokenizer(examples["conversation"], truncation=True, max_length=512)

tokenized_train = train_dataset.map(preprocess_function, batched=True)
tokenized_val = val_dataset.map(preprocess_function, batched=True)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

- 시퀀스 토큰 길이를 “512”로 설정
- 텍스트 길이 분석시 평균 길이가 300자 내외로 도출(“데이터 EDA 및 전처리 2. 텍스트 길이 분석” 참조)
- 텍스트 길이 분석의 결과를 바탕으로 너무 짧게 자르면 모델 성능(F1-Score)이 저하될 우려를 방지하기 위하여 “512” 설정이 적합하다고 가정하였음

## 결과 도출을 위한 다양한 시도

| 그루명 | 추가 실험 내용 | 결과 (F1-score) | 회고 |
|---|---|---|---|
| 대웅 | 라벨 토큰화를 통한 오분류쌍 최소화 및 클래스별 텍스트 증강(Augmentation) | 0.67→0.67 | **문제점**<br>• 협박 ↔ 갈취 : 요구적인 의미로 서로 경계 모호<br>• 기타 괴롭힘 ↔ 협박 : “죽여버리~”라는 표현이 둘다에서 등장<br>• 기타 괴롭힘 ↔ 직장 내 괴롭힘 : 선배, 후배 같은 단어가 학교, 알바, 직장에서도 등장<br><br>**해결 시도**<br>• 라벨별 태크 토큰을 부여하여 전처리에 적용<br>• 직장 라벨에 대하여 추가 후처리 적용<br><br>**개선방법**<br>• 일반 대화에 대한 혼입 가능성을 고려한 규칙 방법 적용 가능<br>• 증강 후 train과 val 분리시 노이즈 발생 가능성이 있으므로 모델 구현의 순서를 변경하는 방법 고려 |

| | | | |
|---|---|---|---|
| 태경 | Supcon loss 추가 | 0.67 → 0.67 | **문제점**<br>• TSNE 시각화에서 클래스 간 중첩 발생 문제를 해결하기 위해 사용<br>• Class imbalance 문제 발생<br><br>**해결 시도**<br>• **Supcon loss 사용**<br>• 배치(Batch) 내에서 **같은 라벨**을 가진 데이터들은 서로 가까워지도록 당기고 (Pull), **다른 라벨**을 가진 데이터들은 서로 멀어지도록 할 수 있음<br>• `Extortion(갈취)`과 `Threat(협박)`처럼 문맥이 비슷한 클래스들을 임베딩 공간에서 강제로 분리하여 경계를 명확하게 할 것으로 예상<br>• weighted loss 사용<br>• 일부 클래스에 더 큰 페널티를 주도록 가중치를 적용한 CrossEntropyLoss 구성<br><br>**개선방법**<br>• **SupCon loss와 Weighted CrossEntropy의 결합 학습**<br>• SupCon loss로 임베딩 공간에서 클래스 간 분리를 먼저 강화 한 이후 weighted CrossEntropyLoss로 불균형을 고려한 분류 성능 최적화 하도록 함<br>• 표현 학습과 분류 목적을 동시에 만족하도록 두가지 loss를 사용해 학습 |
| 희망 | 일반대화(Normal Class) 합성 및 역번역(Back-Translation) 증강 | 0.67→0.66 | **문제점**<br>•복잡한 모델 튜닝이 성능 향상에 큰 도움이 되지 않을 수 있다는 걸 확인.<br>•단순 랜덤 생성이 아니라, 모델이 헷갈릴 만한 데이터를 의도적으로 학습한 전략이 유효했음.<br>•증강을 수행했음에도 불구하고 여전히 데이터 총량이 부족했음.<br><br>**해결 시도**<br>•모델 중심에서 데이터 중심으로 전환하여 데이터 EDA와 전처리에 시간을 투자했다.<br>•데이터 품질 확보의 차원에서 텍스트 길이를 분석하여 max_length를 최적화하고, 중복 데이터를 제거하였다.<br><br>**개선방법**<br>•다양한 페르소나와 상황을 구체적으로 부여하여 중복률을 낮추고 더 현실적인 데이터를 더욱 많이 생성하면 좋을 것 같다.<br>•쉬운 데이터부터 학습시키고 점차 어려운 데이터를 학습시키는 방식을 도입하여 학습 효율을 극대화하면 좋을 것 같다. |
| 성현 | 협박 및 갈취 클래스 대화 역번역 증강 | 0.67→ 0.31 | **문제점**<br>• tsne 시각화 데이터를 확인했을때 저 두개의 클래스가 엮여있어서 거리를 두어야할 것 같단 생각을 함<br><br>**해결 시도**<br>• 두 개의 클래스를 역번역 후 모델에 학습 했음.<br><br>**개선방법**<br>• 학습 후 혼동행렬과 Validation Loss 및 F1 점수는 기존 BaseLine 모델보다 높게 나옴.<br>• Accuracy를 확인하지 못한 실수를 했던것 같습니다. 그래서 Test 결과가 안좋게 나온것같습니다. |
| 수호 | LLM을 활용한 추가 데이터 생성(일반 대화 외 4개 항목에 대한 700개 데이터) | 0.67→0.66 | **문제점**: 모델이 학습할 때 과적합되어서 결과가 좋지 않았다고 생각함.<br><br>**해결 시도**: 추가 데이터(700개)를 만들어 학습시킨 후 모델 성능 평가.<br><br>**개선방법**<br>1. **Early Stopping 강화**: 검증 손실(Validation Loss)이 최저점일 때 학습을 멈춰, 모델이 추가 데이터의 전형적인 패턴에 과하게 빠지는 것을 방지 필<br>2. **데이터 증강(Augmentation)의 고도화**: 단순한 데이터 추가보다는 기존의 거친 데이터를 역번역(Back-translation)하거나 **AEDA**를 통해 변형하여 실전 대응 능력 향상 |

| | | | |
|---|---|---|---|
| | | | 3. **학습률(Learning Rate) 하향**: 데이터가 정교해질수록 더 낮은 학습률로 세밀하게 조정(Fine-tuning)하는 과정필요 |

## 대응 try)

Trial 1) 오분류 쌍들에 대한 해결을 통하여 성능 개선을 시도하였으나 개선 효과 없었음

- 오분류 해결에 집중하여 "일상 대화" 클래스로의 혼입 등을 방지하지 못한 것으로 판단됨
- "일상 대화"의 의미 훼손이나 혼입이 없도록 보수적으로 증강비율(0.1)을 적용하는 것이 필요할 수 있음

```
# 태그 토큰을 붙여서 전처리에 적용

import re

money_re = re.compile(r"(만원|원|계좌|송금|입금|이체|현금|비트코인|이더리움|코인)")
demand_re = re.compile(r"(줘|내놔|보내|가져와|입금해|송금해|할인|빼줘|도장찍|넘겨|사와)")
blackmail_re = re.compile(r"(사진|영상|증거|유포|폭로|공개|카톡 캡쳐|퍼뜨)")
work_re = re.compile(r"(회사|팀장|부장|과장|상사|인사|보고서|회의|출근|퇴근|야근|업무|프로젝
트|점장|매장)")
threat_re = re.compile(r"(죽이|해치|찾아가|가만안두|신고|고소|유포하|공개하)")
cond_re = re.compile(r"(안\s*하면|하면|그러면|기한|며칠|오늘까지|내일까지)")

def add_tags(text: str) -> str:
    tags = []
    if money_re.search(text): tags.append("[MONEY]")
    if demand_re.search(text): tags.append("[DEMAND]")
    if blackmail_re.search(text): tags.append("[BLACKMAIL]")
    if work_re.search(text): tags.append("[WORK]")
    if threat_re.search(text) and cond_re.search(text): tags.append("[COND_THREA
T]")
    return " ".join(tags) + " " + text

# dataset.map에서 conversation에 적용
# example["text"] = add_tags(example["conversation"
```

```
# 토크나이저에 스페셜토큰 추가

specials = ["[MONEY]","[DEMAND]","[BLACKMAIL]","[WORK]","[COND_THREAT]"]
tokenizer.add_special_tokens({"additional_special_tokens": specials})
model.resize_token_embeddings(len(tokenizer))
```

```
# 후처리 적용(직장 라벨의 오탐 최소화)

import numpy as np

WORK_LABEL = label2id["직장 내 괴롭힘 대화"]

def apply_work_gate(texts, logits):
    probs = logits.copy()
    for i, (t, logit) in enumerate(zip(texts, probs)):
        pred = int(np.argmax(logit))
```

```
        if pred == WORK_LABEL and not work_re.search(t):
            # 직장 단서 없으면 2순위 클래스로 내리기
            logit[WORK_LABEL] = -1e9
            probs[i] = logit
    return probs
```

Trial 2) 클래스별 Augmentation(클래스별 텍스트 증강)을 적용하여 표현의 다양성을 확보하고 일반대화의 의미 훼손이나 혼입 위험을 방지하려 하였으나 개선 효과 없었음

- "기타 괴롭힘" 같이 포괄적인 의미(직장에서 괴롭힘, 학교에서 괴롭힘 모두 해당)를 담고 있는 라벨이 증강시 노이즈가 발생(클래스간 경계가 흐려짐)한 것으로 판단됨
- 라벨별로 보존 규칙 필터(라벨을 깨뜨릴 가능성이 큰 증강 샘플은 버리도록 하는 규칙 기반 필터)를 적용한다면 개선할 수 있을 것으로 사료됨

  Ex) 갈취 : 금액/기한/요구("보내", "내놔") 유지

```
# ========================================
# 3.2. 클래스별 Augmentation (라벨 보존)
# ========================================
# 현재 형태(데이터 로드→라벨링→split→학습)는 그대로 유지하면서,
# train/val split 이전에 라벨별 비율로 증강 추가
AUGMENT = True
AUG_SEED = 42

# 라벨별 증강 비율(원본 대비 추가 생성 비중)
# - 유해(0~3)는 표현 다양성 확보를 위해 더 크게 적용
# - 일반(4)은 의미 훼손이나 혼입 위험 때문에 아주 보수적으로 적용
AUG_RATIO = {0: 0.6, 1: 0.6, 2: 0.6, 3: 0.4, 4: 0.1}

import re

_money_re = re.compile(r"(\\d{1,3}(?:,\\d{3})+|\\d+)(\\s*)(원|만원|천원)?")
_space_re = re.compile(r"\\s+")
_punc_re = re.compile(r"([!?.,])\\1{1,}")

def _normalize(text: str) -> str:
    text = str(text)
    text = text.replace("\u200b", " ").replace("\xa0", " ")
    text = _space_re.sub(" ", text).strip()
    return text

def _aug_spacing(text: str, rng: random.Random) -> str:
    # 공백/구두점 약간 변형(의미 보존)
    t = _normalize(text)
    if rng.random() < 0.35:
        t = re.sub(r"\\s*([!?.,])\\s*", r"\\1 ", t)
    t = _space_re.sub(" ", t).strip()
    # 문장 끝 강조 1회
    if rng.random() < 0.20:
        t = _punc_re.sub(r"\\1\\1", t)
    return t

def _money_variant(t: str, rng: random.Random) -> str:
```

```
    # 갈취/돈 관련: 금액 표기 다양화 (예: 10만원 ↔ 100,000원)
    def repl(m):
        num = m.group(1).replace(",", "")
        unit = m.group(3) or ""
        try:
            v = int(num)
        except:
            return m.group(0)
        if unit == "만원":
            v_won = v * 10000
        elif unit == "천원":
            v_won = v * 1000
        else:
            v_won = v
        cand = [f"{v_won:,}원", f"{v_won}원"]
        if v_won % 10000 == 0:
            cand.append(f"{v_won//10000}만원")
        if v_won % 1000 == 0:
            cand.append(f"{v_won//1000}천원")
        return rng.choice(cand)
    return _money_re.sub(repl, t)

def _label_specific(text: str, label: int, rng: random.Random) -> str:
    t = _normalize(text)

    # 라벨별 의미를 바꾸지 않는 표현 variant를 추가 및 치환
    if label == 0:  # 협박
        tails = [" 가만 안 둔다.", " 각오해.", " 후회하게 해줄게.", " 진짜로 가만두지 않을
거야."]
        if rng.random() < 0.35:
            t = t + rng.choice(tails)
        return t

    if label == 1:  # 갈취
        t = _money_variant(t, rng)
        tails = [" 지금 보내.", " 송금해.", " 계좌로 입금해.", " 돈 보내라."]
        if rng.random() < 0.35:
            t = t + rng.choice(tails)
        return t

    if label == 2:  # 직장 괴롭힘
        role_map = {
            "팀장": ["파트장", "리더", "조장"],
            "부장": ["상사", "관리자"],
            "인사팀": ["HR", "총무팀"],
            "사장": ["대표", "회장"],
        }
        for k, vs in role_map.items():
            if k in t and rng.random() < 0.30:
                t = t.replace(k, rng.choice(vs), 1)
        return t

    if label == 3:  # 기타 괴롭힘
        tails = [" 진짜 못됐다.", " 너무 심하다.", " 그만 좀 해.", " 이건 괴롭힘이야."]
```

```
        if rng.random() < 0.25:
            t = t + rng.choice(tails)
        return t

    return t  # 일반(4) 포함: 라벨별 변형은 최소화

def augment_text(text: str, label: int, rng: random.Random) -> str:
    t = _aug_spacing(text, rng)
    # 일반은 보수적으로: label-specific는 제외(혼입 방지)
    if label != 4 and rng.random() < 0.85:
        t = _label_specific(t, label, rng)
    return _normalize(t)

def apply_classwise_augmentation(df: pd.DataFrame,
                                 text_col: str = "conversation",
                                 label_col: str = "label",
                                 ratio_by_label: dict = None,
                                 seed: int = 42) -> pd.DataFrame:
    if ratio_by_label is None:
        ratio_by_label = {0:0.6, 1:0.6, 2:0.6, 3:0.4, 4:0.1}
    rng = random.Random(seed)

    aug_rows = []
    for lab, ratio in ratio_by_label.items():
        sub = df[df[label_col] == lab]
        if len(sub) == 0 or ratio <= 0:
            continue
        n_add = int(len(sub) * ratio)
        if n_add <= 0:
            continue
        pick = sub.sample(n=n_add, replace=True, random_state=seed)
        for _, r in pick.iterrows():
            new_text = augment_text(r[text_col], int(lab), rng)
            aug_rows.append({"class": r.get("class", None), text_col: new_text, lab
el_col: int(lab)})

    if not aug_rows:
        return df

    df_aug = pd.concat([df, pd.DataFrame(aug_rows)], ignore_index=True)
    df_aug = df_aug.sample(frac=1, random_state=seed).reset_index(drop=True)
    return df_aug
```

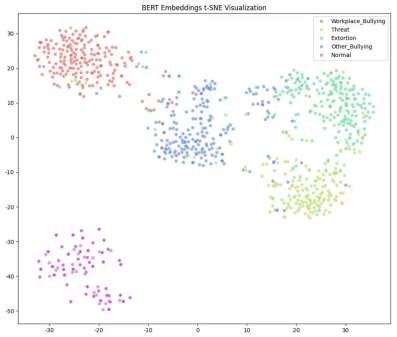
### 태경 try)

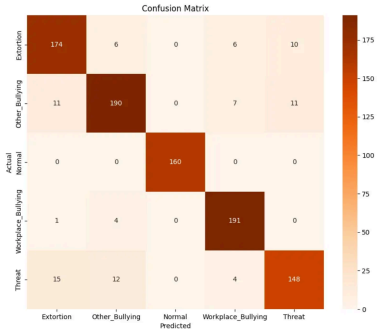
기존 Pre-trained BERT 기반 fine-tuning 학습 결과 → `public score : 0.65`

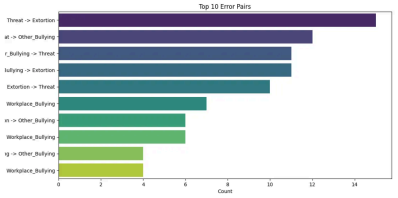
#### 문제점 분석

#1. 전체 데이터셋에 대해서 tsne 결과 확인

#2. confusion matix, 오분류된 샘플들 분석




- validation set에서 기타 괴롭힘 class의 범위가 산발적으로 분포
- 협박 ↔ 갈취 간의 클래스 경계도 불명확함 확인
  - 가장 큰 혼동 : 협박↔ 갈취, 협박↔ 기타 괴롭힘

# 3. 클래스마다 f1 score 값도 차이가 많이 남

| 클래스 | Best model validation f1 score |
|---|---|
| **Workplace_Bullying** | 1.0000 |
| **Other_Bullying** | 0.9620 |
| **Extortion** | 0.9045 |
| **Normal** | 0.8667 |
| **Threat** | 0.8623 |

## 개선시도 with loss

### 1. Supervised Contrastive Loss (SupCon, 대조 손실) 사용

- TSNE 시각화에서 클래스 간 중첩 발생 → 이를 해결하기 위해 사용
- 배치(Batch) 내에서 동일 클래스 샘플 간 표현은 가깝게, 서로 다른 클래스 샘플 간 표현은 멀어지도록 학습하기 위해 supervised contrastive loss를 사용

Extortion(갈취)과 Threat(협박)처럼 의미적으로 유사한 클래스들도 임베딩 공간에서 서로 다른 영역으로 분리되도록 학습하여, 클래스 간 결정 경계를 더욱 명확하게 만들고자 했습니다.

▼ loss 설계

- **Projection**: BERT의 `[CLS]` 벡터를 projection head를 통해 128차원의 임베딩 공간으로 매핑하도록 함
- **Similarity**: 배치 내 모든 샘플 쌍 간의 유사도(보통 코사인 유사도 또는 내적 기반 유사도)를 계산
- **Optimization**: 같은 클래스인 긍정 쌍(Positive pairs)의 유사도는 높이고, 다른 클래스인 부정 쌍(Negative pairs)의 유사도는 낮추도록 학

### 2. Weighted CrossEntropyLoss 사용

클래스별로 다른 중요도(가중치)를 부여해서 소수 클래스의 오분류 에 대해 더 큰 penalty를 주도록 함

기존 BERT 학습 결과 F1 score 확인

| 클래스 | Best model validation f1 score |
|---|---|

| | |
|---|---|
| **Workplace_Bullying** | 1.0000 |
| **Other_Bullying** | 0.9620 |
| **Extortion** | 0.9045 |
| **Normal** | 0.8667 |
| **Threat** | 0.8623 |

Threat ≈ Normal < Extortion < Other_Bullying < Workplace_Bullying

```
class_weights = [1.6, 1.1, 0.6, 0.9, 1.4]
# 순서: Threat, Extortion, Workplace, Other, Normal
```

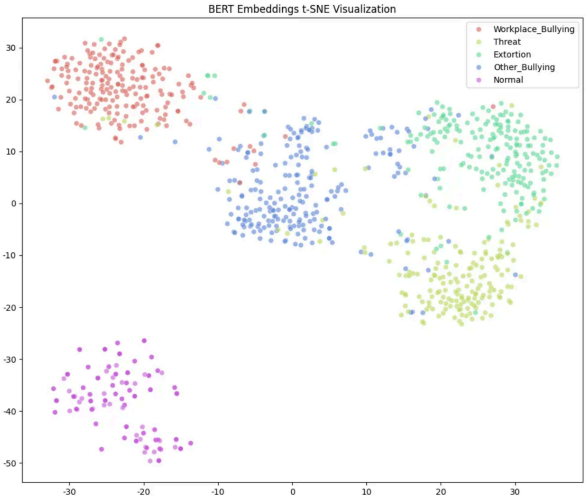
### 3. 최종 loss

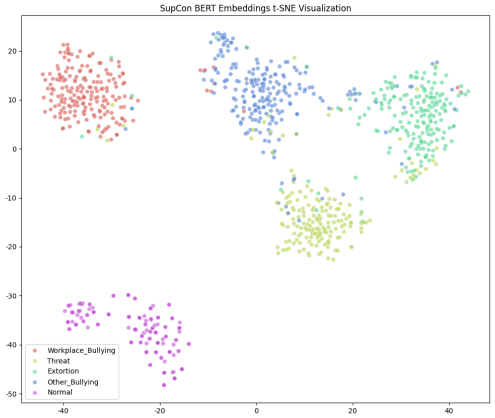
모델 pre-trained kobigbird로 수정후, 똑같이 사용

$L = \alpha L_{CE} + (1 - \alpha) L_{SC}$

| alpha | val f1score |
|---|---|
| **1.0 (SC loss 적용 x)** | **0.9058** |
| 0.3 | **0.9152** |
| 0.7 | **0.9315** |

- 실험하면서 CE loss 비중보다 Contrastive loss 비중을 높이니 분류 학습이 잘 안되는 문제도 존재
- trade off 발생 → 좀 영향도를 조절하기 위해 실험 진행




결과 class간의 경계가 명확해진것을 확인할 수 있었음

하지만, 성능에 대한 개선은 이루어지지 않았음 (test set에 대해서)

**kobigbird (CE loss) 0.67 → kobigbird(CE loss + SupCon loss) 0.67**

**최종 0.9315 모델로 public score 0.67 에서 더 개선 x**

🙂 고찰

Supervised Contrastive Loss를 적용한 결과, 임베딩 공간 상에서 클래스 간 분리도가 향상되고 t-SNE 시각화에서도 중첩이 감소하는 것을 확인하였다. **그러나 이러한 표현 공간의 개선이 반드시 분류 성능 향상으로 직결되지는 않았으며, Threat와**

**Extortion과 같이 의미적으로 중첩되는 클래스의 경우 라벨 자체의 모호성으로 인해 결정 경계 개선에 한계가 존재함을 확인하였다.**

또한 Contrastive loss의 비중이 커질수록 분류 손실의 학습 신호가 약화되어 성능 저하가 발생하는 trade-off가 관찰되었다.

- Supcon loss를 통해서도 hard sample들에 대해서는 잘 분류하지 못하기 때문에 계속 같은 성능이 발생한것 아닐까 하는 고찰
- 모호한 경계들에 대한 추가적인 해결 필요...

## 희망 try)

trial 1) 합성데이터만 사용

| Epoch | Training Loss | Validation Loss | Accuracy | F1 Macro |
|---|---|---|---|---|
| 1 | No log | 0.386237 | 0.860000 | 0.867926 |
| 2 | No log | 0.328789 | 0.908421 | 0.912417 |
| 3 | 0.404877 | 0.332617 | 0.908421 | 0.912114 |

trial 2) Back-Translation & Random Swap/Deletion

(증강 비율) `협박 대화` : 0.3, `갈취 대화`: 0.3, `직장 내 괴롭힘 대화`: 0.3, `기타 괴롭힘 대화` 0.2

```
Processing class: 협박 대화 (Count: 896)
Back-Translation: 100%|██████████| 112/112 [13:19<00:00,  7.14s/it]

Processing class: 갈취 대화 (Count: 981)
Back-Translation: 100%|██████████| 123/123 [16:34<00:00,  8.08s/it]

Processing class: 직장 내 괴롭힘 대화 (Count: 979)
Back-Translation: 100%|██████████| 123/123 [14:30<00:00,  7.08s/it]

Processing class: 기타 괴롭힘 대화 (Count: 1094)
Back-Translation: 100%|██████████| 137/137 [18:53<00:00,  8.27s/it]
```

| Epoch | Training Loss | Validation Loss | Accuracy | F1 |
|---|---|---|---|---|
| 1 | No log | 0.333259 | 0.905263 | 0.904931 |
| 2 | No log | 0.327148 | 0.908421 | 0.907910 |
| 3 | 0.907016 | 0.367445 | 0.923158 | 0.923052 |
| 4 | 0.907016 | 0.399546 | 0.922105 | 0.921864 |
| 5 | 0.217906 | 0.419256 | 0.923158 | 0.923154 |

trial 3) 역번역(Back-Translation) 적용

```
# ============================================
# 3.2. (추가) 역번역(Back-Translation) Augmentation
# ============================================
# - ko → en → ko 로 의미를 최대한 유지하면서 표현을 다양화
# - 번역 품질/의미 훼손 가능성이 있어 기본값은 일반대화(4) 제외
# - 실행 시 Hugging Face 번역 모델을 다운로드

USE_BACK_TRANSLATION = True

# 라벨별 역번역 증강 비율(원본 대비 추가 생성 비중)
# 기본: 유해(0~3)만 적용, 일반(4)은 0
BT_RATIO = {0: 0.3, 1: 0.3, 2: 0.3, 3: 0.2, 4: 0.0}

# 번역 모델(가벼운 MarianMT 계열)
KO_EN_MODEL = "Helsinki-NLP/opus-mt-ko-en"
EN_KO_MODEL = "Helsinki-NLP/opus-mt-en-ko"

def _load_translation_models(device):
```

```
    from transformers import MarianMTModel, MarianTokenizer
    ko_en_tok = MarianTokenizer.from_pretrained(KO_EN_MODEL)
    ko_en = MarianMTModel.from_pretrained(KO_EN_MODEL).to(device).eval()
    en_ko_tok = MarianTokenizer.from_pretrained(EN_KO_MODEL)
    en_ko = MarianMTModel.from_pretrained(EN_KO_MODEL).to(device).eval()
    return (ko_en_tok, ko_en, en_ko_tok, en_ko)

def _translate_batch(texts, tok, model, device, batch_size=16, max_length=256, num_
beams=4):
    import torch
    outs = []
    for i in range(0, len(texts), batch_size):
        batch = [str(t) for t in texts[i:i+batch_size]]
        enc = tok(batch, return_tensors="pt", padding=True, truncation=True, max_le
ngth=max_length).to(device)
        with torch.no_grad():
            gen = model.generate(**enc, max_length=max_length, num_beams=num_beams)
        outs.extend(tok.batch_decode(gen, skip_special_tokens=True))
    return outs

def back_translate_texts(texts, device, batch_size=16, max_length=256, num_beams=
4):
    """
    입력: 한국어 텍스트 리스트
    출력: ko→en→ko 역번역 결과 리스트(길이 동일)
    """
    try:
        ko_en_tok, ko_en, en_ko_tok, en_ko = _load_translation_models(device)
        en_texts = _translate_batch(texts, ko_en_tok, ko_en, device, batch_size=bat
ch_size, max_length=max_length, num_beams=num_beams)
        ko_texts = _translate_batch(en_texts, en_ko_tok, en_ko, device, batch_size=
batch_size, max_length=max_length, num_beams=num_beams)
        return [ _normalize(t) for t in ko_texts ]
    except Exception as e:
        # 번역 모델 로드/생성 실패 시 원문 반환 (학습 파이프라인 중단 방지)
        print(f"⚠ [Back-Translation] 실패하여 원문을 사용합니다: {e}")
        return [ _normalize(t) for t in texts ]

def apply_back_translation(df: pd.DataFrame,
                           text_col: str = "conversation",
                           label_col: str = "label",
                           ratio_by_label: dict = None,
                           seed: int = 42,
                           device = None,
                           batch_size: int = 16,
                           max_length: int = 256,
                           num_beams: int = 4) -> pd.DataFrame:
    """
    라벨별로 일부 샘플을 뽑아 역번역한 문장을 추가.
    """
    if ratio_by_label is None:
        ratio_by_label = {0:0.3, 1:0.3, 2:0.3, 3:0.2, 4:0.0}
    if device is None:
        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
    aug_rows = []
    rng = random.Random(seed)

    for lab, ratio in ratio_by_label.items():
        sub = df[df[label_col] == lab]
        if len(sub) == 0 or ratio <= 0:
            continue
        n_add = int(len(sub) * ratio)
        if n_add <= 0:
            continue

        pick = sub.sample(n=n_add, replace=True, random_state=seed)
        texts = [ _normalize(t) for t in pick[text_col].tolist() ]

        # 역번역 수행
        bt_texts = back_translate_texts(texts, device=device, batch_size=batch_siz
e, max_length=max_length, num_beams=num_beams)

        # 너무 비슷한(효과 미미) 샘플은 일부 제거(보수적)
        for orig, new_text, (_, r) in zip(texts, bt_texts, pick.iterrows()):
            if not new_text:
                continue
            if new_text == orig:
                # 완전 동일하면 스킵
                continue
            # 길이 변화가 너무 큰 경우 스킵(의미 훼손 가능성)
            if len(new_text) < 0.5 * len(orig) or len(new_text) > 2.0 * len(orig):
                continue
            aug_rows.append({"class": r.get("class", None), text_col: new_text, lab
el_col: int(lab)})

    if not aug_rows:
        return df

    df_bt = pd.concat([df, pd.DataFrame(aug_rows)], ignore_index=True)
    df_bt = df_bt.sample(frac=1, random_state=seed).reset_index(drop=True)
    return df_bt

if AUGMENT:
    print("\n[Augmentation] 적용 전 라벨 분포:\n", df_final['label'].value_counts().so
rt_index())
    df_final = apply_classwise_augmentation(df_final, ratio_by_label=AUG_RATIO, see
d=AUG_SEED)
    print("\n[Augmentation] 적용 후 라벨 분포:\n", df_final['label'].value_counts().so
rt_index())

    if USE_BACK_TRANSLATION:
        print("\n[Back-Translation] 적용 전 라벨 분포:\n", df_final['label'].value_cou
nts().sort_index())
        df_final = apply_back_translation(df_final, ratio_by_label=BT_RATIO, seed=A
UG_SEED, device=device, batch_size=16, max_length=256, num_beams=4)
        print("\n[Back-Translation] 적용 후 라벨 분포:\n", df_final['label'].value_cou
nts().sort_index())
```

```
train_df, val_df = train_test_split(df_final, test_size=0.2, random_state=42, strat
ify=df_final['label'])
train_dataset = Dataset.from_pandas(train_df[['conversation', 'label']])
val_dataset = Dataset.from_pandas(val_df[['conversation', 'label']])
```

- Process
  - 원문 데이터 : 한국어
    ↓
    중간 언어 데이터 : 영어 (DeepL/Helsinki-NLP)
    ↓
    Forward 번역(한국어→영어)
    ↓
    Backward 번역(영어→한국어)
    ↓
    필터링 : 의미보존체크 및 중복 제거
    ↓
    원본과 증강 데이터를 함께 학습
- 결과

  예상 결과
  - 의미는 유지하되 문장 구조와 어휘의 다양성 증가 예상(다양한 표현 확보)
  - 특정 표현 패턴에만 치중되는 현상을 최대한 줄여서 Overfitting이 완화 예상

  실제 결과
  - F1-Score 개선에는 큰 영향을 주지 못하였음
- 예시
  - 갈취 : "돈 내놔" → "Give me the money" → "돈을 주세요"
  - 일반대화 : “지금 어디야?” → "Give me the money" → “지금 어디에 있어?”

👌결과에 대한 고찰(Reflection & Insight)

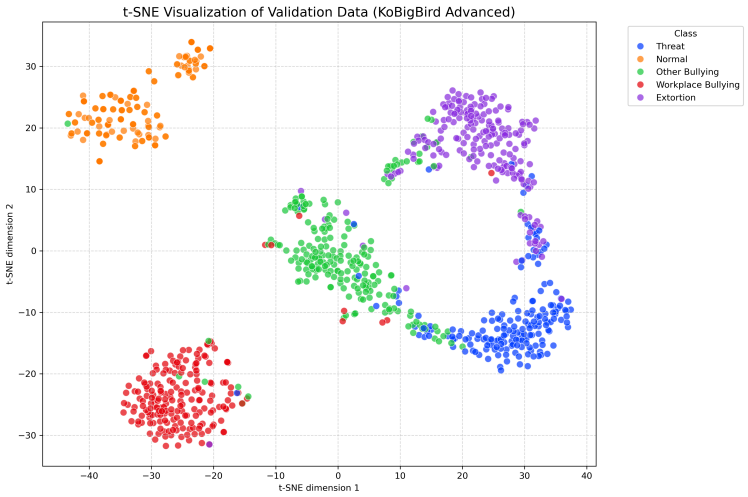
1. 데이터의 양보다 다양성이 핵심이다.

   다양한 LLM을 섞어 썼다면 어땠을까? 각 모델마다 선호하는 어휘와 문장 구조가 다르기 때문에, 중복률은 낮추고 문체의 다양성은 획기적으로 높일 수 있었을 것이다.
2. 프롬프트는 구체적일수록 좋다.

   갈취와 일반 대화를 구분해내는 결정적인 요인은 “돈을 빌리지만 정중한 거절” 같은 구체적인 시나리오였다. 프롬프트의 고도화는 단순한 테크닉이 아닌 모델의 성능을 결정짓는 하이퍼 파라미터와 같은 역할을 한다는 것을 확인하였다.

### 성현 try)

기존 데이터가 협박과 갈취 클래스에서 엮어있는 상황이 발생해서 이 두개의 클래스에 대해서만 역번역 진행했음




KoBigBird 학습 시작! (시간이 조금 더 걸릴 수 있습니다)

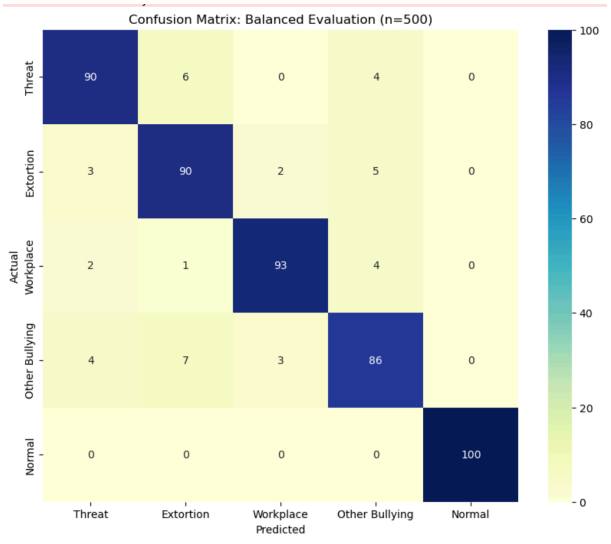
[1190/1190 20:11, Epoch 5/5]

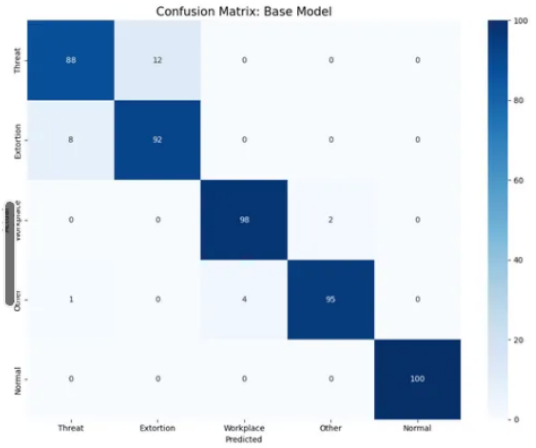
| ɔch | Training Loss | Validation Loss | Accuracy | F1 |
|---|---|---|---|---|
| 1 | No log | 0.381707 | 0.881053 | 0.880199 |
| 2 | No log | 0.308115 | 0.914737 | 0.914488 |
| 3 | 0.895885 | 0.384378 | 0.920000 | 0.920432 |
| 4 | 0.895885 | 0.405492 | 0.924211 | 0.924201 |
| 5 | 0.215782 | 0.403111 | 0.928421 | 0.928498 |

[1190/1190 10:25, Epoch 5/5]

| Epoch | Training Loss | Validation Loss | F1 |
|---|---|---|---|
| 1 | No log | 0.212901 | 0.936748 |
| 2 | No log | 0.246261 | 0.943272 |
| 3 | 0.557704 | 0.289666 | 0.938927 |
| 4 | 0.557704 | 0.279095 | 0.949482 |
| 5 | 0.088100 | 0.295828 | 0.950502 |

역번역 진행후 Validation Loss가 줄어들면서 일반화가 잘 되었음을 증명을 하였고 F1 점수가 오르면서 데이터가 서로 분별을 잘 하고 있음을 확인했음.




혼동 행렬에서도 역번역 데이터가 더 잘 분별됨을 확인도 했음

```
예측 완료! 'submission_final.csv' 저장됨
==========================================
[클래스별 예측 분포]
· Class 0 [협박 대화] : 25개
· Class 1 [갈취 대화] : 7개
· Class 2 [직장 내 괴롭힘 대화] : 145개
· Class 3 [기타 괴롭힘 대화] : 322개
· Class 4 [일반 대화] : 1개
```

하지만 테스트 결과 Submission_final.csv 파일에서는 오히려 class 2,3에 결과가 몰리는 것을 확인할 수 있었음.

결론: 데이터를 다시 역번역을 통해 데이터를 잘 만들어서 평가지표의 성능을 높여도 결국 테스트 값이 안나오는 상황이라, 그렇게 효과가 없었다는것을 느낌

### 수호 try)

추가 데이터 생성 및 분석

일반대화 외 4개 class에 대해서 각 200개씩 추가 데이터를 Chat GPT를 사용하여 생성 후 학습시켰으나, 모델 성능은 0.04가 낮았고, TestSet에서는 Baseline 모델 결과보다 0.01 낮게 나옴

결론

1. 데이터를 생성할 때 너무 일정한 패턴으로 나와서 성능이 올라가지 않았을 수 있다고 생각함. 따라서 데이터를 좀 더 확인해 보는 것이 필요할 것으로 보임
2. 향후 개선 방안으로는 합성데이터 말고 각 클래스에 맞는 실제 interaction 데이터를 가져와서 추가 학습시켜보면 성능이 올라갈 것으로 기대

```
==================================================
[최종 모델 평가 리포트 (Test Set: 100 samples/class)]
==================================================
정확도 (Accuracy): 0.9580
F1 점수 (Weighted): 0.9577
--------------------------------------------------

[클래스별 상세 지표]
              precision    recall  f1-score   support

          협박     0.9151    0.9700    0.9417       100
          갈취     0.9574    0.9000    0.9278       100
      직장 괴롭힘     0.9615    1.0000    0.9804       100
      기타 괴롭힘     0.9684    0.9200    0.9436       100
       일반 대화     0.9901    1.0000    0.9950       100

    accuracy                         0.9580       500
   macro avg     0.9585    0.9580    0.9577       500
weighted avg     0.9585    0.9580    0.9577       500

==================================================
```

baseline 모델 평가

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 협박 | 0.85 | 0.93 | 0.89 | 213 |
| 갈취 | 0.88 | 0.92 | 0.90 | 231 |
| 직장 괴롭힘 | 0.94 | 0.95 | 0.94 | 231 |
| 기타 괴롭힘 | 0.92 | 0.80 | 0.85 | 254 |
| 일반 대화 | 1.00 | 1.00 | 1.00 | 160 |
| accuracy | | | 0.91 | 1089 |
| macro avg | 0.92 | 0.92 | 0.92 | 1089 |
| weighted avg | 0.91 | 0.91 | 0.91 | 1089 |

데이터 추가 후 모델 평가

## 프로젝트 최종 결론

KoBigBird만 사용했을 때보다 성능을 개선시키기 위해 다양한 시도를 해봤으나, 성능을 개선시키진 못함.

**추후 고려사항**

1. 생성 데이터에 대한 품질개선

- 생성 한 데이터에 대한 성능 문제로 test set의 일반데이터에 대한 분류가 잘 안될것 같음.
- 데이터 품질 개선 시도했지만 시간상 실패한것에 대한 아쉬움.

2. low Rank Adaptation 방법 처럼 pretrained 모델들을 활용한 PEFT(Parameter-Efficient Fine-Tuning) 방식 기법들을 추가해서 적용해보면 좋을 것 같음.

## 회고

안수호 : **아직까지 모델성능을 높이기 위해서 체계적으로 어떤 접근을 해야하는지 이해하지 못하고 있음을 인지함**. 제미나이에서 알려준 내용을 다 시도해봤음에도 성능이 0.67에서 머무르고 있는 것을 보면 완전히 처음부터 모델 설정, 하이퍼파라미터 최적화 등 체계적으로 분류할 필요가 있다는 것을 느낌. 어떤 것을 가장 먼저 기준에 두고 변경을 해나갈지에 대한 감을 잡는 것이 주요하다고 생각함

신대웅 : BERT 기반의 한국어 사전학습 모델들의 특징들을 알게되었고 KoBigBird를 선택하여 본 프로젝트에 적용해보았음. 우리조에서 실험해보았던 모델들 중 F1-Score가 가장 높게 나왔으나 가장 좋은 모델이기 때문은 아니고 현재 데이터셋(위협 대화와 일상적인 대화)에 적합한 모델이기 때문에 F1-Score 상승에 기여한 것이라고 생각함. **역번역과 텍스트 증강 방법을 연구하여 적용해본 것이 가장 큰 수확이었음.**

박성현 : 데이터 증강을 통해 모델의 성능을 높이는 것과 데이터 마다의 성격 그리고, 다른 데이터와의 관계를 통해 증강을 해야하며 유사도를 끊임없이 확인하며 다른 평가 지표도 활용해야하는 어려움을 느낌, 마**냥 평가지표 점수들이 좋다고 테스트 결과가 좋은게 아닌것을 이번 DLthon에서 확인**하였습니다.

서희망 : 이번 프로젝트는 **없는 데이터를 만들어내는 힘이 AI 엔지니어에게 얼마나 중요한지 깨닫게 해준 계기였습니다.** 단순히 주어진 데이터를 학습시키는 수동적인 태도에서 벗어나, 문제를 정의하고 필요한 데이터를 직접 설계하는 주도적인 엔지니어링을 경험했습니다.

김태경 : 기존 모델의 문제점을 분석해보고 성능 개선을 위한 노력을 해보았는데, validation score는 올랐지만 실제 test 성능이 오르지 않은것을 보고 일반화된 모델들을 만드는것에 대한 중요성을 느끼게 되었던것 같습니다. **토크나이저나 lora등의 학습 방식도 변경해보고 싶었지만 시간상 너무 아쉬운 프로젝트 였던것 같습니다.**