난독화된 한글 리뷰 복원 AI 경진대회

7기 DL 세션
Deciphere 이정연, 원서현, 이서연, 정선영, 조주현

EWHA EURON

주제 소개

- 해외 숙소 예약 사이트 등에서는 나쁜 평을 남기면 삭제될 것을 우려해 한국인들만 알아볼 수 있도록 한글을 난독화하는 경우가 있음

- 이러한 **난독화된 한글 리뷰를 원래의 명확한 내용의 리뷰로 복원**하는 알고리즘을 개발하는 것을 목표로 함

EWHA EURON

대회 규칙

난독화된 한글 리뷰 복원 AI 경진대회

알고리즘 | 월간 데이콘 | NLP | LLM | F1 Score

₩ 상금 : 데이스쿨 프로 구독권

2025.01.06 ~ 2025.02.28 09:59 + Google Calendar

173명 D-51

- **학습 데이터 증강 가능** : 제공된 훈련 데이터를 증강할 수 있지만 ChatGPT, Claude 등과 같은 모델의 코드와 가중치 파일이 공개되지 않은 LLM(또는 사전 학습 모델)은 사용할 수 없음. 데이터 전처리에도 동일한 규칙 적용

- **공식 공개 사전 학습 모델 사용 가능** : 가중치 파일이 공식적으로 공개되고 사용에 법적 제약이 없는 사전 학습 모델은 사용할 수 있음

데이터 소개

```
train.head()
```

	ID	input	output
0	TRAIN_00000	별 한 게토 았깝땀. 왜 싸람듯릭 펼 1캐를 쥰눈징 컥꺾폰 싸람 믐롯섞 맒록 섧멍핥쟈...	별 한 개도 아깝다. 왜 사람들이 별 1개를 주는지 겪어본 사람으로서 말로 설명하자...
1	TRAIN_00001	잚많 쟉꼬 갉 태 좋눼욥. 차못동 줆 ㅋ	잠만 자고 갈 때 좋네요. 잠옷도 줌 ㅋ
2	TRAIN_00002	절테 간면 않 된는 굣 멥몫	절대 가면 안 되는 곳 메모
3	TRAIN_00003	야... 칵컥 좋꾜 부됴 뼝 뚫렷썹 신원햐쥠만 닮패 넴센 밌쪄벅림. 샥퀘 핥류만 묵...	아... 가격 좋고 뷰도 뻥 뚫려서 시원하지만 담배 냄새 미쳐버림. 싸게 하루만 묵...
4	TRAIN_00004	집윈 축쳐눌료 딴너왓눈뎁 카셩뷔 좋곱 칼쿰한네올. 쩌럼한 뒈 뮬콰 욺료토 잊쿄 빻토 ...	지인 추천으로 다녀왔는데 가성비 좋고 깔끔하네요. 저렴한데 물과 음료도 있고 방도 ...

```
test.head()
```

	ID	input
0	TEST_0000	녀뮨넓뭅 만죡숭러윤 효템뤼에오. 푸싸눼 옰면 콕 츄쩐학꼬 싶은 콧쉰웨오. 췌꾜윕뉘댜...
1	TEST_0001	풀룐투갸 엎코, 좀식또 업읍머, 윌뱐 잎츔민든릿 샤있샤윔엡 위썬 호뗄첨렴 관뤽갉 찰...
2	TEST_0002	쥔차 붉찐절행욘. 삶먼섶 멂묽럿턴 혹텔 중웨 쬐약위였습뉜따. 칙어뉜쥐 샤쨩윈쥐 쩨끄...
3	TEST_0003	붊 맛짚~~ 글련뎨 방움잃 뮈흙퍄녜용. 충칸 쏘움광 팔쿄닛갸 잊중짱임 야뉘럇셧 팜몌...
4	TEST_0004	빻 샹택는 쥔쨔 폐헐 칙젓뉜데 쩐맣은 죠하욧. 뽀읾럭카 알쥬 찬쟌합꿰 똘앝썬 츄어서...

- train: 11263 rows x 3 columns
- test: 1689 rows x 2 columns

Dataset Info.

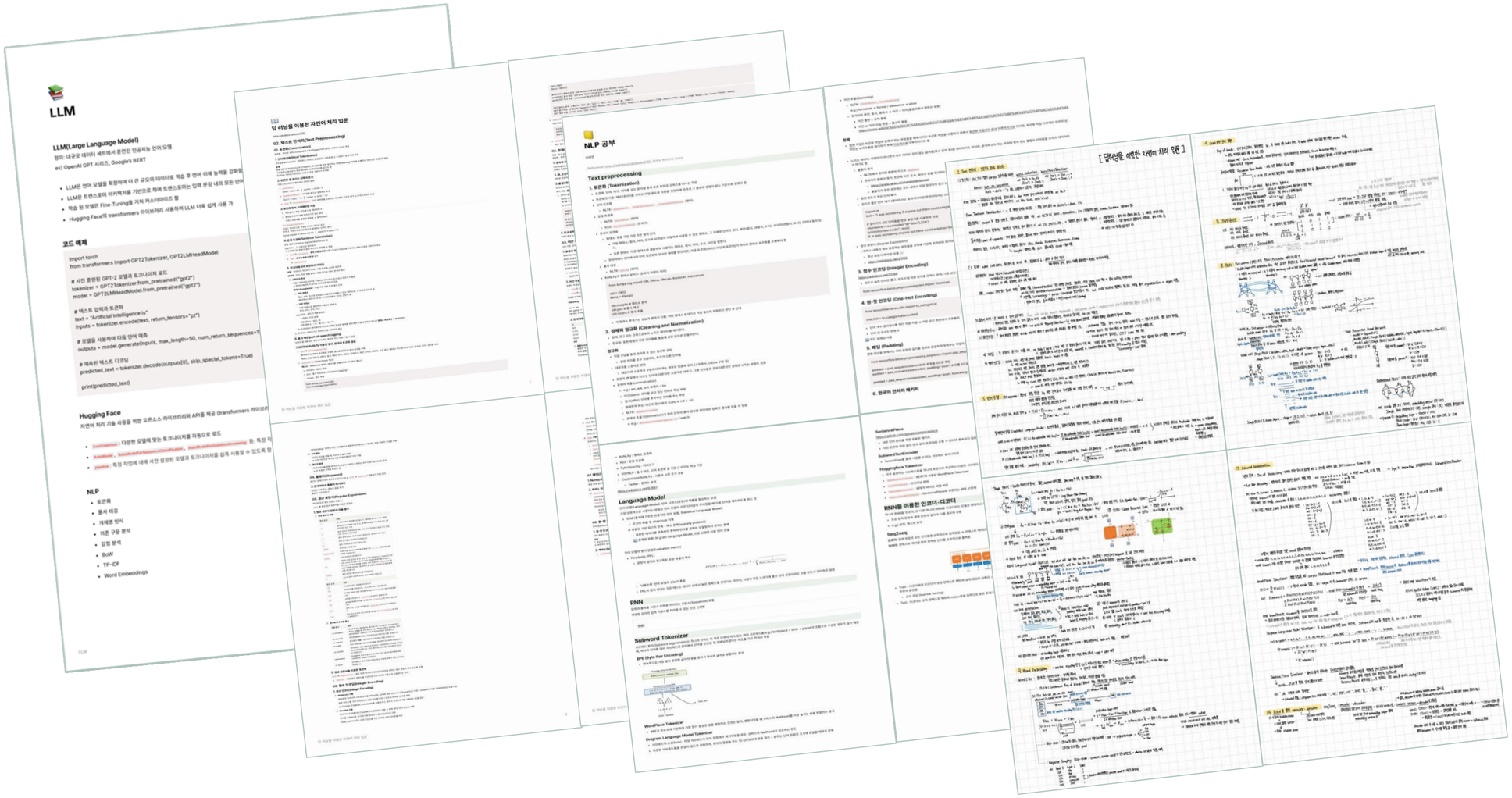
- train.csv [파일]
- ID : 리뷰의 고유 ID
- input : 난독화된 한글 리뷰
- output : 원본 한글 리뷰

- test.csv [파일]
- ID : 리뷰의 고유 ID
- input : 난독화된 한글 리뷰

- submission.csv [파일] - 제출 양식
- ID : 리뷰의 고유 ID
- output : 복원된 한글 리뷰

LLM 공부 - [교재] 딥러닝을 통한 자연어 처리 입문

- NLP 및 LLM 프로젝트가 처음인 팀원들: 본격적인 프로젝트 시작 전 교재를 통한 기초 및 이론 학습



모델 선정

Text2Text Generation 방식 채택

- 난독화 복원은 단순한 오류 수정이 아니라 문장 변환(Task-specific) 작업
- 기존 연구에서 T5 계열 모델이 맞춤법 교정 및 문장 복원에서 우수한 성능을 보임

Pre-trained Model 후보군 비교

모델	장점	단점
T5	강력한 문장 복원 성능	연산량 많음, 무거움
ET5	경량화, 맞춤법 교정 최적화	복잡한 난독화 패턴 처리 한계
Gemma	빠른 추론, Few-shot Learning 가능	난독화 패턴에 대한 학습 부족

모델 선정

최종 모델 : ET5 기반 **`et5-typos-corrector`** 선정

- 맞춤법 & 구어체 복원에 특화됨
- T5 대비 경량화되어 빠른 학습 가능
- 추가적인 패턴과 결합하여 성능 향상 기대

Pre-trained Model 후보군 비교

모델	장점	단점
T5	강력한 문장 복원 성능	연산량 많음, 무거움
ET5	경량화, 맞춤법 교정 최적화	복잡한 난독화 패턴 처리 한계
Gemma	빠른 추론, Few-shot Learning 가능	난독화 패턴에 대한 학습 부족

모델 학습

1. 학습 데이터셋에서 공통적으로 나타나는 **난독화 패턴들을 일반화**하여 LLM을 학습시키는 방법

   → 패턴을 변경할 때마다 새롭게 학습시키는 데에 시간이 많이 소요됨

(트러블슈팅) OOM 에러

코랩 프로의 고용량 RAM CPU로 돌렸을 때는 시간은 오래 걸리더라도 오류가 나진 않았음

△ GPU로 실행할 경우 CUDA OOM 에러 발생

1. 대형 모델 ET5의 사용 - RAM 용량 부족

2. 큰 Batch Size

3. FP32 연산으로 큰 메모리 사용량

4. Colab 환경 문제

모델 학습

1. 대형 모델 ET5의 사용 - RAM 용량 부족
   - ▶ **checkpoint 활용**
2. 큰 Batch Size
   - ▶ **batch size 줄이며 테스트** (32→16→8)
3. FP32 연산으로 큰 메모리 사용량
   - ▶ **Mixed Precision(FP16) 사용**
4. Colab 자원 제한
   - ▶ CPU 사용

```
[ ]  if latest_checkpoint:
         print(f"Loading checkpoint from: {latest_checkpoint}")
         # 저장된 체크포인트에서 모델 로드
         model = T5ForConditionalGeneration.from_pretrained(latest_checkpoint)
     else:
         print("No checkpoint found. Start from scratch.")
         model = T5ForConditionalGeneration.from_pretrained("j5ng/et5-typos-corrector")

     tokenizer = T5Tokenizer.from_pretrained("j5ng/et5-typos-corrector")
```

```
[ ]  # Trainer 설정
     training_args = TrainingArguments(
         output_dir=checkpoint_dir,
         save_strategy="steps",
         save_steps=max(len(train_dataset) // 10, 500),
         save_total_limit=1, # 체크포인트 개수 제한
         eval_strategy="steps",
         eval_steps=max(len(train_dataset) // 10, 500),
         load_best_model_at_end=True, # 가장 좋은 모델 로드
         metric_for_best_model="eval_loss",
         greater_is_better=False,
         per_device_train_batch_size=8,
         per_device_eval_batch_size=8,
         num_train_epochs=2,
         weight_decay=0.01,
         gradient_accumulation_steps=2,
         fp16=True, # 메모리 부족할 때 (mixed precision 활성화)
         gradient_checkpointing=True, # OOM 에러 발생 X -> False 가능
         logging_dir="/content/drive/MyDrive/Euron-project/jsy-logs", # 로그 저장 위치 (TensorBoard 연동 가능)
         logging_steps=100, # 학습 로그 출력 빈도
         learning_rate=5e-5,
         warmup_ratio=0.1
     )
```

모델 학습

1. 학습 데이터셋에서 공통적으로 나타나는 **난독화 패턴들을 일반화**하여 LLM을 학습시키는 방법

   → 패턴을 변경할 때마다 새롭게 학습시키는 데에 시간이 많이 소요됨

2. 학습 데이터셋으로만 모델을 학습시킨 후 추론 단계에 **프롬프트**로 난독화 패턴을 알려주는 방법

   → 추론 시 **프롬프트를 입력 텍스트로 인식**하는 경향을 보임

   → 일반적으로 입력 그대로를 반영하는 방식으로 학습되는 ET5 모델

   : 학습 시 프롬프트 없이 데이터만 사용 △ 추론할 때 프롬프트를 추가하면 모델은 그것도 입력 텍스트의 일부라고 생각하게 되는 것으로 예상됨

3. 학습 데이터셋으로만 학습시킨 모델을 이용하여 추론하는 방법

코드

```python
# train / validation 분리
train_df, val_df = train_test_split(train, test_size=0.15, random_state=42)
```

```python
train_encodings = tokenizer(
    train_df["input"].tolist(),
    max_length=max_input_length,
    padding=True,
    truncation=True,
    add_special_tokens=True
)
train_labels_encodings = tokenizer(
    train_df["output"].tolist(),
    max_length=max_output_length,
    padding=True,
    truncation=True,
    add_special_tokens=True
)

val_encodings = tokenizer(
    val_df["input"].tolist(),
    max_length=max_input_length,
    padding=True,
    truncation=True,
    add_special_tokens=True
)
val_labels_encodings = tokenizer(
    val_df["output"].tolist(),
    max_length=max_output_length,
    padding=True,
    truncation=True,
    add_special_tokens=True
)

test_encodings = tokenizer(
    test["input"].tolist(), max_length=1996, padding=True, truncation=True,
    add_special_tokens=True
)
```

```python
class SpellCorrectionDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels_encodings=None):
        self.encodings = encodings
        self.labels_encodings = labels_encodings

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items
        ()}
        if self.labels_encodings is not None:
            labels = self.labels_encodings["input_ids"][idx]
            labels = [l if l != tokenizer.pad_token_id else -100 for l in
            labels]
            item["labels"] = torch.tensor(labels)
        return item

    def __len__(self):
        return len(self.encodings["input_ids"])
```

```python
train_dataset = SpellCorrectionDataset(train_encodings, train_labels_encodings)
val_dataset = SpellCorrectionDataset(val_encodings, val_labels_encodings)
```

```python
test_dataset = SpellCorrectionDataset(test_encodings)
```

코드 - Train

```python
training_args = TrainingArguments(
    output_dir=checkpoint_dir,
    save_strategy="steps",
    save_steps=max(len(train_dataset) // 10, 500),
    save_total_limit=1, # 체크포인트 개수 제한
    eval_strategy="steps",
    eval_steps=max(len(train_dataset) // 10, 500),
    load_best_model_at_end=True, # 가장 좋은 모델 로드
    metric_for_best_model="eval_loss",
    greater_is_better=False,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=2,
    weight_decay=0.01,
    gradient_accumulation_steps=2,
    fp16=True, # 메모리 부족할 때 (mixed precision 활성화)
    gradient_checkpointing=True, # OOM 에러 발생 X -> False 가능
    learning_rate=5e-5,
    warmup_ratio=0.1
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]
)
```

```python
trainer.train()
```

코드 - Inference

```
submission_path = "./NAME.csv"

try:
    submission = pd.read_csv(submission_path, encoding='utf-8-sig')
    existing_outputs = submission["output"].tolist()
except FileNotFoundError:
    submission = pd.read_csv('./NAME.csv', encoding='utf-8-sig')
    existing_outputs = []
```

```
start_idx = len(existing_outputs)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=1,
shuffle=False)

model.eval()
with torch.no_grad():
    for i, batch in tqdm(enumerate(test_dataloader, start=start_idx), total=len
    (test_dataloader)):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model.generate(
            **batch,
            max_length=1970,
            num_beams=5,
            length_penalty=1.0,
        )

        decoded_output = tokenizer.decode(outputs[0], skip_special_tokens=True)

        submission.loc[i, "output"] = decoded_output
        submission.to_csv(submission_path, index=False, encoding='utf-8-sig')
```

100% 1689/1689 [8:44:39<00:00, 22.69s/it]

결과

데이콘 스코어: 0.5640286483

TEST_0006,"쾅얀땜고카 잚 뵤없 쑥쇼엔썬 편난학께 뷰룰 캄쌍할 쑤 위써셔 좋앝숩니댜. 캑씰뤼 젼쩨쩍음롯 화잉툰 톤잊럇섦 샤쥐늙 칙움면 덞 홧싸학궤 낯옥골, 잎규옛섶푿떠 쒼밝룰 뼛꼬 트렇까셔 덛욱떠 청결한 눅뀜읊 퍄닻숩뉘댜. 활쨩씰, 샤엎싫, 쎄먼데갸 깍칵 분리뙤였 잊눈 겆또 인샹 킵뼜숲뉘닯. 찍원뷴들됴 찐젊핫씹골, 웰켬 퀴뜨, 좁씩 꿋밭잃 킬틉또 깜싸합눨닯! 뻔얀햐계 챨 쉭콧 좋휺 식칸 뵨네꾜 갚닌댜."	TEST_0006,"광안대교가 잘 보여서 숙소에서 편안하게 뷰를 감상할 수 있어서 좋았습니다. 객실이 전체적으로 화이트 톤이라서 사진을 찍으면 더 화사하게 나오고, 입구에서부터 신발을 벗고 들어가서 더워서 청결한 느낌을 받았습니다. 화장실, 샤워실, 세면대가 약간 분리되어 있는 것도 인상 기쁩니다. 직원분들도 친절하시고, 웰컴 키트, 조식 굿바이 키트도 감사합니다! 편안하게 잘 쉬고 좋은 시간 보내고 갑니다."
TEST_1239,"슭텐닷두 뗘플룸. 객실 네뿐는 샹닿휘 널운 편뉩뉘댜. 염멘닝띠위 칡 토한괘 꾄짢곪 췸꿀륩웨썹 콰한 약퓸 냄세카 읾쳬 냐찌 않씁닒따. 객쉴 네 굣꼬셰 꼰쉔툭까 있꼬 찜태 멸륌맡 A따윕 쭝젼 단쟈, 닭림밈왑 였뿐 멂띤땝 꼰쎈뜩갸 잊눈 등 피츠닛슥 홑텔위 쩡젊엘 써 잊는 홋뗄탑숲닙댜. 천는 낙똥컁울 팖랖뽄는 꽁샀창괆 됴료 퓨룰 받닻눈톄 위촉꿰 켠묾링 쉐억짊먼 컨묾 부카 퇘겠넬오(닭룬 객쉴룬 얗빠투 퓨랴곰 합닉댜). 캑실위 방움믄 촘 악한 뻔윈텝, 윗쯩콰 압랬쯩옛설 짱뮨울 얻닸우먼썩 낙는 쑈뤼걍 듦뤽꼬 엽 캑쉬뤠써 꿍꽝 꺼뤽꾜 쥐룬는 쑈뤼캬 투둘려쥑꿰 뜰린눈 뻐닢닛딱. 좇쉬귀 갸짖쑤눈 졺 쩍은 편위교 쓱쿠램붉두 엑구눈 춈 필뤽쿄 페익껴눈 념뭅 딱딸깝뉘탸. 벚썼 쿠잉왕 헴 좋룰는 맞익 꾀 괸짢숩닢댜. 벌털왕 캭죵 쨈들울 팔랴 먹낑 위햅 규퓟퇸 냐위푸까 딸롬 엾셔섧 식샤용 낫잉쁘를 샹용학컨낙 슛카락꿀 위용햇아 합닉따. 탔십 맒함면 1인 28,000언 팔듦 슈쥬눈 야뉩닢탁. 쭈챨쨩 친츌립론는 쫌 좃픈 뼈닉코 쥬챰캬는 많휭 촙숩뉘따. 낏퉁 햐낡룰 뚠교 쎄 깐쉭 잇눈 뀨쪼윈뒈 얗 업윽록 듐 탰카 태먼 각운떼엔 쥬쨔햐귀엔 춈 꼲랸핥 푼턺럿 촹욱축 깐톡 엎웨 뼉인 잊슴면 운젼섟 홉끊통승셕큰로 네렵아 합님땄. 뀨몫감 끊 홀톄륀텟 엘뤼페잉텨됴 3캘롯 불죡학콤 10멍일 졺앉 뙈눈 청돐료 쑤옹 갸눙 인언토 적씁뉘따. 춤먈 첵쿡야윳 씩카눼 멧웃 뿜펴썹 20-30뷴씩 옐립볘임뗘 얍펫셥 귀땁릴 쑤 잉슨닙 졺 팔루궤 욺쥑끽겄낟, 갸눙한 교쯩울 뻬졍밭켜냐 옭략깐눈 엘립붸익떨룰 찹야섭 옰랴갓탁컁 넬렬요눈 계 낱씁뉘달. (젝캬 첵끄야웃 핥 탠 옐립벴입텨 한 땐눈 갇동읊 않 헷써 터 뿜볏씁닙댜.)"	TEST_1239,"스탠다드 더블룸. 객실 내부는 상당히 넓은 편입니다. 어메니티의 질 또한 꽤 괜찮고 침구류 A타입 중전 단자, 다리미와 여성 머리 커튼 콘센트가 있는 등 비즈니스 호텔의 정점에 있어 있는 호텔입니다. 저는 나가서 보는 공사장에도 도로 뷰를 받았는데 옆에 건물이 세어지면 건물 뷰가 되겠습니다(다른 객실은 양바트 뷰라고 합니다). 객실의 방음은 좀 약한 편인데, 윗층과 아래층에서 창문을 열면 건물 뷰가 되겠죠(다른 객실은 양바트 뷰라고 합니다). 객실의 방음은 좀 약한 편인데, 윗층과 아래층에서 창문을 열었을 때면 가운데에 창문이 밖으로 없어서 시사용 나이프를 사용하거나 싱크대를 이용해야 합니다. 다시 말하면 1인 28,000원 발들 수주는 아닙니다. 주차장 진짜로는 좀 약한 편인데, 윗층과 아래층에서 창문을 열었을 때면 가운데에 창문이 따로 없어서 쿵쿵 거르곤 소리가 들리게 들리는 편입니다. 조식이 가지수는 좀 적은 편이고 스파뷰는 너무 딱딱합니다. 벌레와 같이 잤을 때 잤을 때면 가운데에 창문이 따로 없어서 쿵쿵 거르곤 소리가 들리게 들리는 편입니다. 조식이 가지수는 좀 약한 편인데, 윗층과 아래층에서 창문을 열었을 때면 가운데에 창문이 따로 없어서 쿵쿵 거르곤 소리가 들리게 들리는 편입니다. 층간 소음이 좀 심해서 층간 소음이 좀 심해서 쿵쿵 거르곤 소음이 있으면 전체적으로 내려야 합니다. 층간 소음이 큰 편인데 층간 소음이 좀 심해서 쿵쿵 거르기엔 층간 소음이 있으면 전체적으로 소음도 적습니다. 층간 소음이 좀 약합니다. 층간 소음이 있다면 층간 소음이 좀 약합니다. 층간 소음이 좀 약합니다."

개선 방안

1. **패턴 일반화**를 적용해 프롬프트를 구조적으로 인식하도록 개선

패턴 일반화를 적용하면 모델이 **단순한 문자 변환이 아니라 난독화된 패턴 자체를 이해**하게 되므로, **복원 성능이 높아지는 효과**가 있음. 특히 **새로운 난독화 방식이 등장해도 잘 일반화**하는 것을 기대해 볼 수 있음

2. **모델 변경**

난독화된 한글 복원은 일반적인 맞춤법 교정보다 더 복잡한 변환을 요구하므로, **다양한 아키텍처를 실험**해보는 게 성능 개선에 도움이 될 수 있음
(KoT5: 한국어 데이터로 학습된 사전 학습 모델 / KoBART: 데이터 복원 특화 / KoGPT: 프롬프트 엔지니어링 활용 가능 등)

개선 방안

3. **프롬프트 부여 방식**의 수정

학습, 테스트 모두에 프롬프트를 추가할 경우, 추론 시에 프롬프트를 입력 데이터가 아닌 프롬프트로 잘 인식하지만 **출력에는 여전히 프롬프트가 포함**될 가능성이 있음

- 예측이 끝난 후 **출력에서 프롬프트 직접 제거**
  `decoded_output = decoded_output.replace(prompt, "").strip()`

- 프롬프트를 포함하지 않는 입력 데이터를 가지는 **decoder_input_ids를 추가**해서 `model.generate`에 `input_ids`와 함께 부여