

# 기말고사 대체 레포트



## 자연어처리 및 응용 노래 가사 생성 서비스 제작

과목명	자연어처리및응용
담당교수명	이상홍
학과	의용생체공학과
학번	202138114
이름	김유리

## 목차

1. 프로젝트 개요	2
2. 모델 훑어보기	2
2.1 모델 구조	
2.2 KoGPT2	
3. 데이터셋	3
3.1 데이터셋 선정	
3.2 EDA	
3.3 레이블 구성	
4. 모델 학습하기	6
4.1. 코랩 노트북 초기화하기	
4.2. 각종 설정하기	
4.3. 데이터 전처리	
4.4. 토큰나이저 준비하기	
4.5. 데이터 구축하기	
4.6. 모델 불러오기	
4.7. 모델 학습시키기	
5. 학습 마친 모델을 실전 투입하기	12
5.1. 환경 설정하기	
5.2. TF-IDF 설정하기	
5.3. 토큰나이저 및 모델 불러오기	
5.4. 모델 출력값 만들고 후처리하기	
5.5. 웹서비스 시작하기	
6. 최종 결과물	15
7. 한계 및 발전 가능성	17

## 1. 프로젝트 개요

본 프로젝트는 교재 ‘Do it! BERT 와 GPT 로 배우는 자연어 처리 (이기창, 2021)’ 의 내용을 참고하여 진행되었다. 본문 중 ‘8 장: 문장 생성하기’ 실습 예제를 사용해 노래 가사 생성 서비스를 개발했다.

모든 실습은 구글 코랩에서 제공하는 L4-GPU 를 사용해 이루어졌다. L4-GPU 의 스펙은 다음과 같다.

- System RAM 53.0 GB
- GPU RAM 22.5 GB
- Disk 201.2 GB

조건에 맞게 batch\_size 및 epoch 를 조정하여 학습을 진행했다.

노래를 작사할 때는 다양한 것을 고려해야한다. 초기의 아이디어를 얻는 것이 어려울 경우도 존재를 한다. 또한 장르에 따라 작사하는 가사가 달라진다. 이러한 초기 노래 가사 작성에 도움을 줄 수 있는 인공지능을 개발하는 것을 목표로 했다.

작성한 프로그램은 장르를 선택한 후, 가사를 입력하면 노래 가사를 완성시켜준다. 장르는 데이터세트에서 가장 많이 등장하는 5 개 그리고 그 외로 6 가지 중에서 고르는 것이 가능하다. 장르를 선택 후 단어를 입력하면 그에 따른 노래가사를 출력을 해주고, 데이터베이스 속 가장 유사한 가사를 가진 노래 상위 5 개와 실제 유사도를 계산을 해 숫자로 표현을 한다.

## 2. 모델 훑어보기

### 2.1. 모델 구조

‘문장 생성’ 이란 이전 단어 (컨텍스트)가 주어졌을 때 다음으로 오는 단어의 확률을 계산하여 그것을 기준으로 맞추는 태스크를 뜻한다. 문장 생성 모델은 주로 트랜스포머를 사용한다. 크게는 BERT(Bidirectional Encoder Representations from Transformers)과 GPT(Generative Pre-Trained Transformer)가 있다. Transformer 의 Encoder 부분만을 사용하는 BERT 는 빈칸을 맞추는 태스크를 수행할 때 사용되고, Decoder 부분만을 사용하는 GPT 는 다음 단어를 맞추는 태스크를 수행할 때 사용된다.

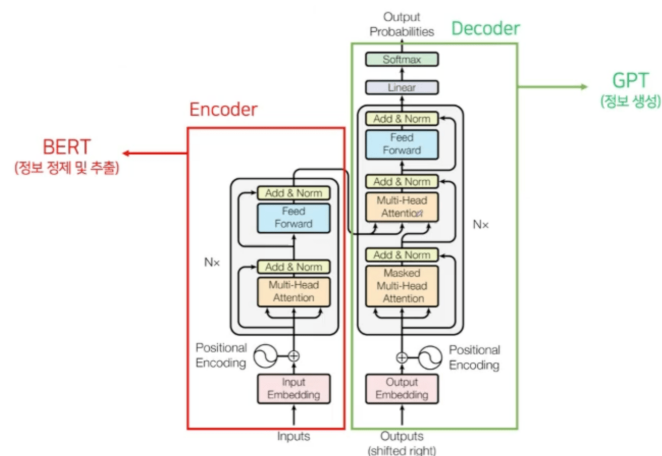


Figure 1: The Transformer - model architecture.

## 2.2. KoGPT2

프로젝트에서 사용한 모델은 교재 8 장에서 사용한 SKT-AI 의 KoGPT2 이다. 흔하게 알려진 GPT-2 의 한글판인 KoGPT2 2.0 는 GPT-2 모델을 파인튜닝한 2020 년 2 월에 개발된 KoGPT2 1.0 의 업그레이드 버전이다.

## 3. 데이터세트

### 3.1. 데이터세트 선정

#### 출처

<https://github.com/EX3exp/Kpop-lyric-datasets>

#### 데이터세트 설명

2000 년부터 2023 년 10 월까지 음악 스트리밍 사이트 멜론 (<https://melon.com>) 의 월간 차트 100 위의 곡들의 정보를 수집한 데이터세트이다.

#### 라이선스

BSD 3-Clause License

### 3.2. EDA

본 EDA (Exploratory Data Analysis)는 데이터세트의 구조, 내용 및 품질을 파악하기 위해 진행되었다.

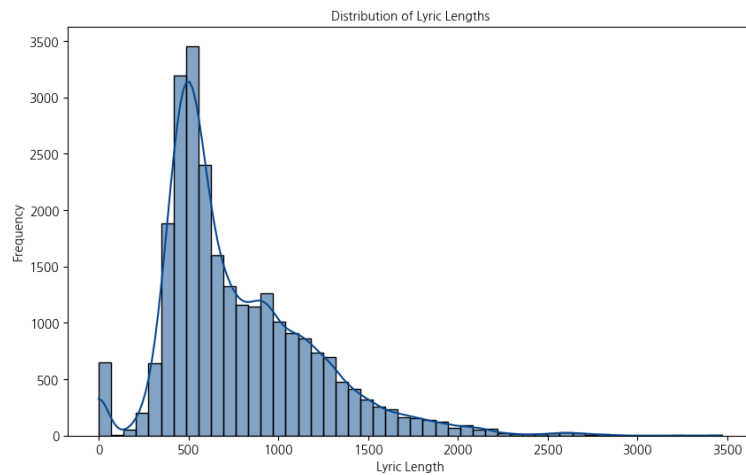
깃허브의 파일 구성은 아래와 같다.

Folder Structure
-- .gitignore
-- LICENSE
-- utils
-- __init__.py
-- data_parser.py
-- melon
-- monthly-chart
-- melon-2000
-- melon-2000-01
--melon-monthly_2000-01_01.json
-- ...
--melon-monthly_2000-01_75.json
-- ...
-- melon-2000-12
-- ...
-- melon-2023
-- README.md
-- composers.txt
-- lyricists.txt
-- arrangers.txt
-- artists.txt

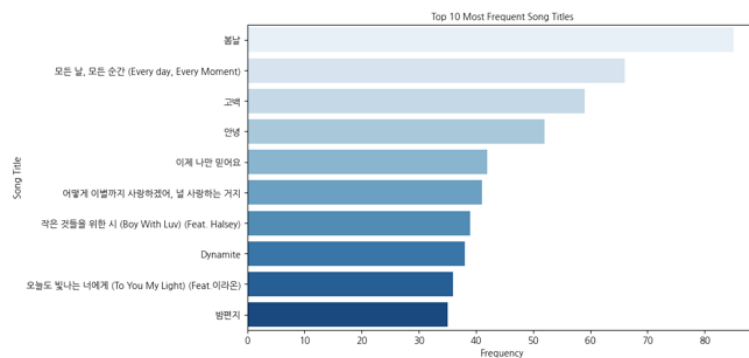
- 데이터세트 크기: 2691232 bytes
- 총 json 파일 수: 25,876
- Unique 한 노래 수: 7519
- 중복되는 노래 수: 18357
- 장르 수: 66
- 총 한글 단어 수: 9878113
- 총 영문 단어 수: 4594561

다음은 데이터세트의 몇가지 Visualization 결과이다.

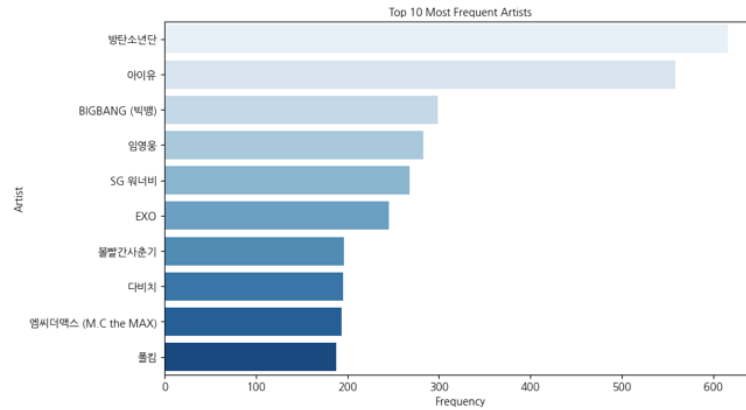
노래 가사 길이 분포



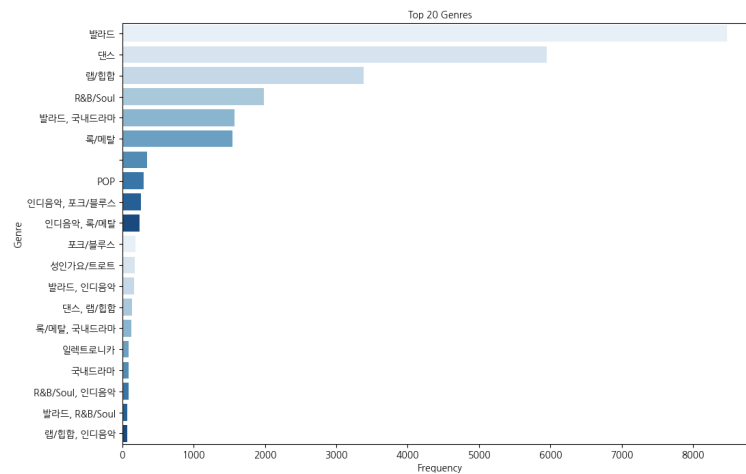
가장 자주 등장한 노래 제목 (중복된 노래는 이후에 전처리 과정에서 제거)



가장 자주 등장한 가수 10 명



장르 66 개 중 가장 자주 등장한 20 개



### 3.3. 레이블 구성

각 노래에 대한 json 파일이 하나씩 존재한다. 파일 속 레이블의 구성과 관련된 정보는 아래와 같다.

레이블	설명
info	차트에 존재했을 때의 날짜 'year' , 'month' , 당시 순위 'rank' , 그리고 사이트명 'website' 에 대한 정보를 포함
song_id	노래 아이디 번호
song_name	제목
album	노래가 포함된 앨범명
release_date	출시 날짜
artist	가수
genre	장르
lyric_writer	작사가
composer	작곡가
arranger	편곡가

lyrics	총 가사 줄의 개수 'row_num' , 그리고 그에 해당하는 가사가 'lines' 에서 list 형식으로 기재
--------	--

Json 파일 예시
<pre>{   "info": [     {       "year": 2023,       "month": 7,       "rank": 16,       "type": "월별차트",       "website": "Melon"     },     ""   ],   "song_id": "35454425",   "song_name": "Attention",   "album": "NewJeans 1st EP 'New Jeans'",   "release_date": "2022.08.01",   "artist": "NewJeans",   "genre": "댄스",   "lyric_writer": "Gigi",   "composer": "Duckbay (Cosmos Studios Stockholm)",   "arranger": "다니엘 (DANIELLE)",   "lyrics": {     "row_num": 79,     "lines": [       "You and me",       "내 맘이 보이지",       "한참을 쳐다봐",       "가까이 다가가",       "You see",       "You see, ey ey ey ey",       "",       "One, two, three",       "... etc."     ]   } }</pre>

## 4. 모델 학습하기

### 4.1. 코랩 노트북 초기화하기

본 프로젝트는 코랩 환경에서 진행이 되었다. 데이터는 깃허브 주소를 clone 하여 다운받는다.

코드 4-1

```
!git clone https://github.com/EX3exp/Kpop-lyric-datasets.git
```

## 4.2. 각종 설정하기

관련 라이브러리 및 패키지를 설치한다.

코드 4-2-1

```
# 의존성 패키지 설치
!pip install transformers[torch]
!pip install gradio
```

파일 저장을 드라이브에 하기 위해 드라이브를 연결시킨다.

코드 4-2-2

```
# 저장을 위해 구글 드라이브와 연결
from google.colab import drive
drive.mount('/gdrive', force_remount=True)
```

코드 4-2-3

```
# 라이브러리 불러오기
import os
import json
import pandas as pd
from transformers import PreTrainedTokenizerFast, GPT2LMHeadModel,
GPT2Config, Trainer, TrainingArguments
import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import gradio as gr
```

## 4.3. 데이터 전처리

장르의 분포는 다음과 같다.

장르	개수
발라드	2810
댄스	1867
Other	1413
랩/힙합	1028
R&B/Soul	635
발라드, 국내드라마	499

데이터셋 경로를 알맞게 설정을 한다.

코드 4-3-1

```
# 데이터 경로 지정
```



```
dataset_folder = '/content/Kpop-lyric-datasets/melon/monthly-chart'
```

사용한 데이터는 json 파일 속에 저장되어 있기 때문에 이를 읽고 Pandas 라이브러리의 DataFrame 으로 변환하는 작업을 수행해야 한다.

#### 코드 4-3-2

```
# JSON 파일들을 탐색할 수 있도록 함수 지정
def load_data(folder):
    dataframes = []
    for root, dirs, files in os.walk(folder):
        for file in files:
            if file.endswith('.json'):
                filepath = os.path.join(root, file)
                with open(filepath, 'r', encoding='utf-8') as f:
                    data = json.load(f)
                    # JSON → DataFrame
                    df = pd.json_normalize(data)
                    dataframes.append(df)
    return pd.concat(dataframes, ignore_index=True)
```

다음 코드를 실행하여 데이터를 폴더에서 읽어온다.

#### 코드 4-3-3

```
# 데이터셋 불러오기
df = load_data(dataset_folder)
```

DataFrame 으로 변환 후, 필요한 정보만을 남기고, 중복인 노래 (곡 제목, 가사가 같은 노래)들을 제거한다.

#### 코드 4-3-4

```
# Lyrics 한 줄로 병합하기
if 'lyrics.lines' in df.columns:
    df['lyrics'] = df['lyrics.lines'].apply(lambda lines: ' '.join(lines) if
    isinstance(lines, list) else '')
else:
    raise ValueError("Column 'lyrics.lines' not found in the dataset.")
# 필요한 데이터만 보존
df = df[['song_name', 'artist', 'album', 'release_date', 'genre', 'lyrics']]

# 중복되는 노래 제거
df = df.drop_duplicates(subset=['song_name', 'artist'])

# 상위 5개 장르 및 그 외 지정
top_5_genres = df['genre'].value_counts().index[:5].tolist()
```

```
df['genre'] = df['genre'].apply(lambda x: x if x in top_5_genres else 'Other')
```

8:2 비율로 학습데이터 그리고 검증데이터로 나눈다.

코드 4-3-5

```
# Train/Test Split
train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)

# 각 데이터를 CSV 파일로 저장
train_df.to_csv('train_kpop_lyrics.csv', index=False)
val_df.to_csv('val_kpop_lyrics.csv', index=False)
```

#### 4.4. 토큰라이저 준비하기

KoGPT2 모델이 사용하는 토큰라이저를 선언한다. 교재를 참고하여 eos 는 SK 텔레콤에서 지정한 대로 </s>를 사용하고, 문자열의 길이를 맞추기 위해 패딩 토큰인 <pad>를 이용한다.

코드 4-4

```
tokenizer = PreTrainedTokenizerFast.from_pretrained("skt/kogpt2-base-v2",
    bos_token='</s>', eos_token='</s>', unk_token='<unk>',
    pad_token='<pad>', mask_token='<mask>')
```

#### 4.5. 데이터셋 구축하기

사용하는 데이터는 가사 그리고 ‘장르’ 를 레이블로 사용한다. 따라서 이 클래스는 DataFrame 에서 가사와 장르를 추출하고, 이를 토큰라이저를 사용하여 토큰화한 뒤, PyTorch 모델이 입력으로 사용할 수 있는 형식으로 반환한다.

코드 4-5-2

```
# 데이터셋 클래스 선언
class LyricsDataset(Dataset):
    def __init__(self, dataframe, tokenizer, max_len=512):
        self.tokenizer = tokenizer
        self.data = dataframe
        self.texts = dataframe.lyrics
        self.labels = dataframe.genre
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, index):
        text = self.texts.iloc[index]
        genre = self.labels.iloc[index]
```

```
# 토큰화
inputs = self.tokenizer.encode_plus(
    genre + " " + text,
    None,
    add_special_tokens=True,
    max_length=self.max_len,
    padding='max_length',
    truncation=True,
    return_tensors="pt"
)

ids = inputs["input_ids"].flatten()
mask = inputs["attention_mask"].flatten()

return {
    'input_ids': ids,
    'attention_mask': mask,
    'labels': ids
}
```

그리고 다음코드를 이용해 학습 및 검증 데이터셋을 구축한다.

코드 4-5-2

```
# 데이터셋 구축
train_dataset = LyricsDataset(train_df, tokenizer, max_len=512)
val_dataset = LyricsDataset(val_df, tokenizer, max_len=512)

# 데이터 로더 지정
train_dataloader = DataLoader(train_dataset, batch_size=4, shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=4, shuffle=False)
```

#### 4.6. 모델 불러오기

Pretrained 된 모델을 불러온다. GPT2LMHeadModel 은 KoGPT2 가 프리트레인할 때 썼던 모델 클래스이다. 프리트레인 태스크와 파인튜닝 태스크 모두 '다음 단어 맞히기' 이다.

코드 4-6

```
# 모델 초기화
config = GPT2Config.from_pretrained("skt/kogpt2-base-v2")
model = GPT2LMHeadModel.from_pretrained("skt/kogpt2-base-v2", config=config)
```

#### 4.7. 모델 학습시키기

초기 설정은 책과 동일하게 batch\_size=4, epoch=3 으로 돌렸지만 본 실습에서는 batch\_size=4, epoch=4 로 지정을 했다.

## 코드 4-7-1

```
# 학습 설정
training_args = TrainingArguments(
    output_dir="/gdrive/MyDrive/KpopLyricsModel",
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=4,
    save_steps=500,
    save_total_limit=2,
    logging_steps=100,
    evaluation_strategy="steps",
    eval_steps=500,
    load_best_model_at_end=True
)
```

다음 코드를 사용해 학습을 진행하면, 구글 코랩 화면에서 진행상황이 출력되면서 모델이 학습된다.

## 코드 4-7-2

```
# 트레이너 정의
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer
)

# 학습 개시
trainer.train()
```

[4001/6584 30:21 < 19:36, 2.20 it/s, Epoch 2.43/4]		
Step	Training Loss	Validation Loss
500	2.016400	1.960941
1000	2.042300	1.915783
1500	1.944700	1.877837
2000	1.851700	1.868241
2500	1.765800	1.842894
3000	1.754700	1.829799
3500	1.584600	1.827905
[385/412 00:42 < 00:03, 8.97 it/s]		

## 5. 학습 마친 모델을 실전 투입하기

### 5.1. 각종 설정하기

4 의 학습을 마친 모델을 드라이브에 저장한다.

코드 5-1-1
<pre># 모델 저장 model.save_pretrained("/gdrive/MyDrive/KpopLyricsModel") tokenizer.save_pretrained("/gdrive/MyDrive/KpopLyricsModel")</pre>

CPU 에 있던 모델을 GPU 로 옮긴다 .

코드 5-1-2
<pre># GPU 로 설정하기 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")  # 모델을 GPU 로 옮기기 model.to(device)</pre>

### 5.2. TF-IDF 설정하기

가사 간의 유사성을 계산하기 위해서 TF-IDF(Term Frequency-Inverse Document Frequency)를 사용한다. TF-IDF 는 단어의 빈도와 역 문서 빈도를 사용하여 단어들마다 중요한 정도에 따라서 가중치를 부여하는 방법이다. 생성을 위해서는 문서를 사용된 단어의 벡터로 나타내야 한다.

<TF-IDF 계산 방법>

$$TF(t, d) = \frac{\text{문서 } d \text{ 에서 단어 } t \text{ 가 등장한 횟수}}{\text{문서 } d \text{ 에 등장한 모든 단어의 수}}$$

$$IDF(t, D) = \log \left( \frac{\text{총 문서의 개수}}{\text{단어 } t \text{ 를 포함하는 문서의 수}} \right)$$

$$TF-IDF(t, d, D) = TF(t, d) * IDF(t, D)$$

코드 5-2-1

```
# TF-IDF 벡터화 셋업
vectorizer = TfidfVectorizer().fit(df['lyrics'])
```

### 5.3. 모델 출력값 만들고 후처리하기

더 다양하고 반복없는 퀄리티 좋은 문장을 생성하기 위해 다양한 샘플링 인자들을 적용한다.

- do\_sample=True
- top\_k=50
- top\_p=0.95
- repetition\_penalty=1.0 (반복구절을 없애줌)
- temperature=1.0 (문장을 다양하게 생성하게 함)

코드 5-2-2

```
# 가장 유사도가 높은 5 개의 곡을 탐색
def generate_lyrics(genre, prompt_text, max_length=100):
    input_text = genre + " " + prompt_text
    input_ids = tokenizer.encode(input_text, return_tensors='pt')

    sample_outputs = model.generate(
        input_ids,
        do_sample=True,
        max_length=max_length,
        top_k=50,
        top_p=0.95,

        num_return_sequences=1
    )

    generated_text = tokenizer.decode(sample_outputs[0],
    skip_special_tokens=True)
    generated_text = generated_text[len(genre) + 1:]

    generated_vec = vectorizer.transform([generated_text])
    existing_vecs = vectorizer.transform(df['lyrics'])
    similarities = cosine_similarity(generated_vec, existing_vecs).flatten()

    top_5_indices = similarities.argsort()[-5:][::-1]
    top_5_songs = df.iloc[top_5_indices][['song_name', 'artist', 'lyrics']]
    top_5_songs['similarity'] = similarities[top_5_indices]

    return generated_text, top_5_songs

genres = top_5_genres + ["Other"]
```

## 5.4. 웹서비스 시작하기

본 프로그램은 교재와는 다르게 더 간단한 Gradio 라이브러리를 사용한다. 인공지능 서비스의 UI를 프로토타입으로 간편히 제작하기 좋은 프레임워크이며 현재 사용하는 허깅페이스(Hugging Face) 플랫폼이기도 해서 간편하게 연결이 가능하다.

코드 5-3-1

```
# Gradio 인터페이스 설정
def gradio_interface(genre, prompt):
    generated_text, top_5_songs = generate_lyrics(genre, prompt)
    top_5_songs_str = top_5_songs.to_string(index=False)
    return generated_text, top_5_songs_str

gr_interface = gr.Interface(
    fn=gradio_interface,
    inputs=[gr.Dropdown(choices=genres, label="Genre"),
    gr.Textbox(label="Input Text")],
    outputs=["text", "text"],
    title="Lyric Generator"
)
```

다음 코드를 사용해 Gradio를 실행시킨다.

Output에 출력되는 셀 속에서 바로 작업을 진행해도 되며, 셀 안에 생성되는 URL을 사용해 웹으로 접속을 해도 된다.

코드 5-3-2

```
# 서비스 개시
gr_interface.launch()
```

Setting queue=True in a Colab notebook requires sharing enabled. Setting 'share=True' (you can turn this off by setting 'share=False' in 'launch()') explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()

Running on public URL: <https://1fcbf31aee03f770c9.gradio.live>

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run 'gradio deploy' from Terminal to deploy to Spaces (<https://huggingface.co/spaces>)

### 노래 가사 생성 서비스

Genre

Input Text

Clear Submit

output 0

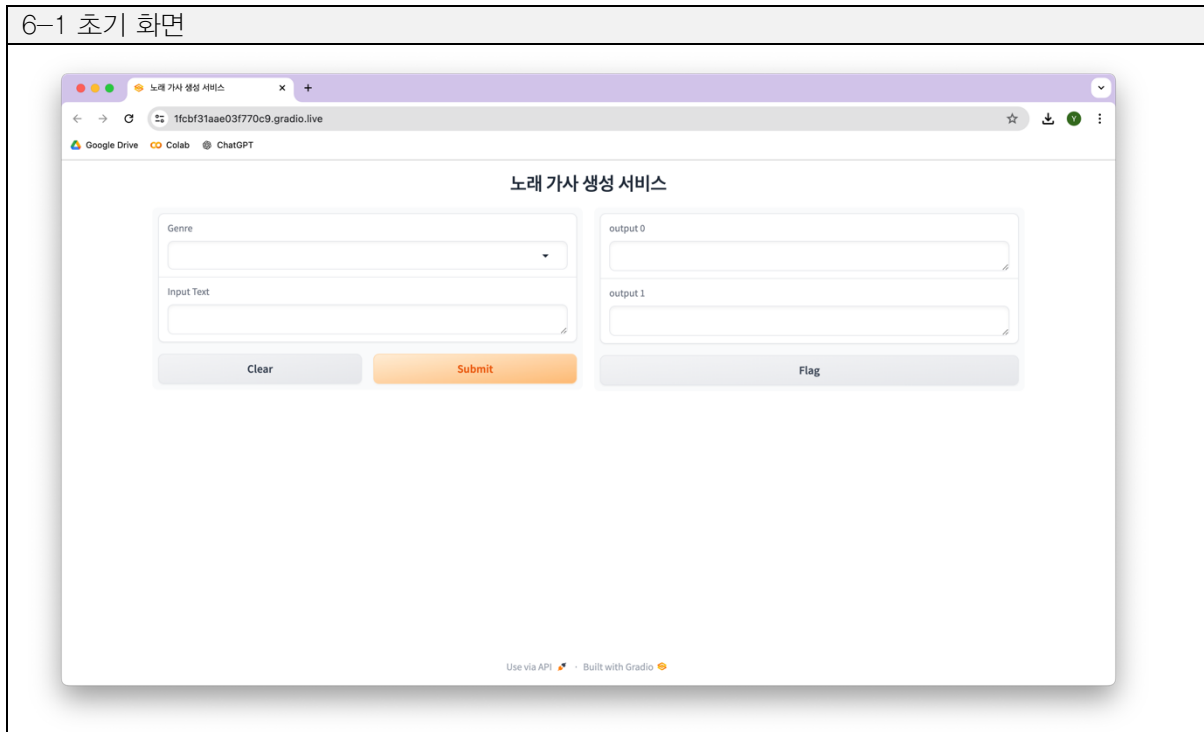
output 1

Flag

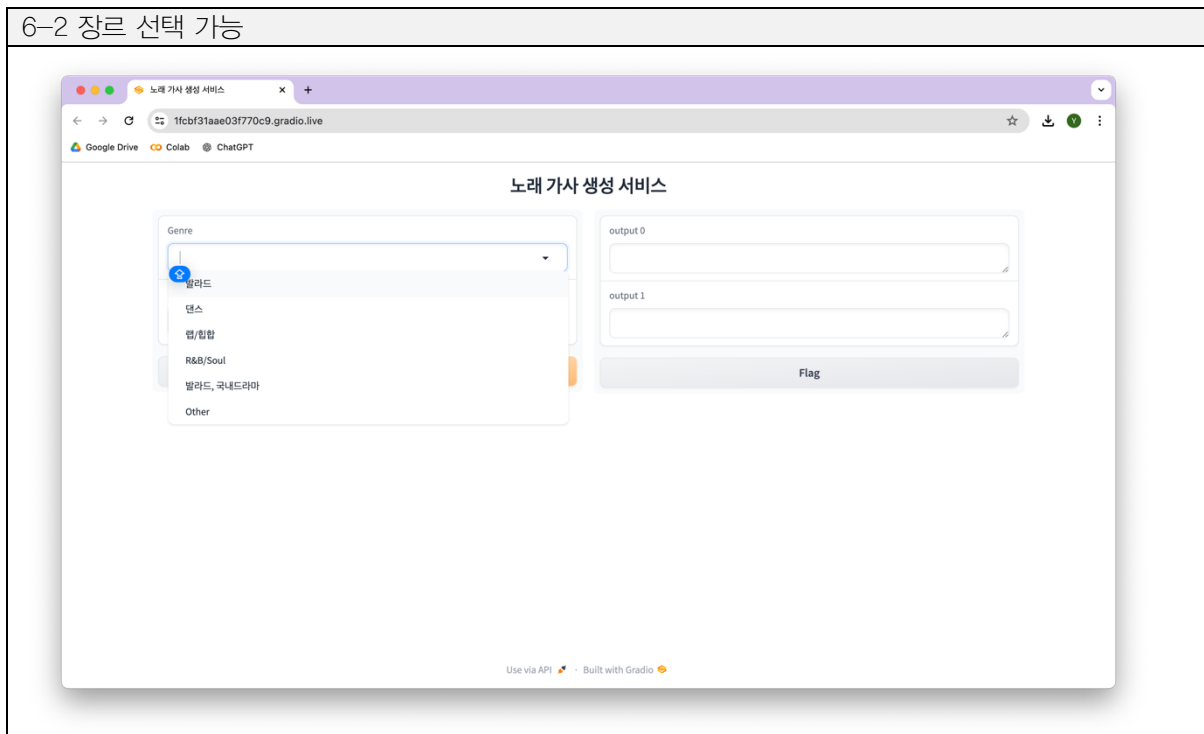
Use via API • Built with Gradio

## 6. 최종 결과물

### 6-1 초기 화면

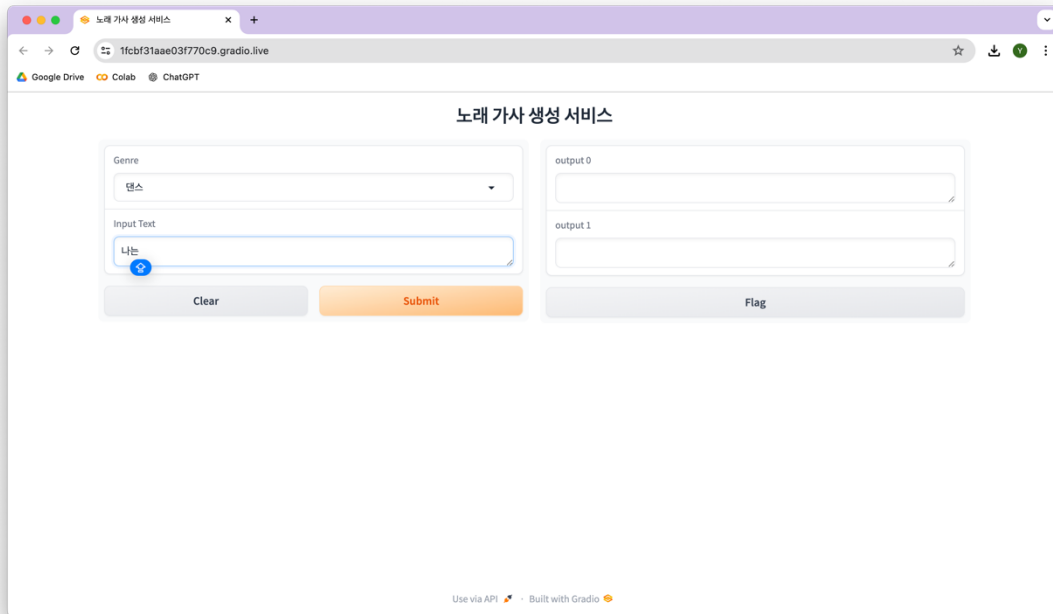


### 6-2 장르 선택 가능

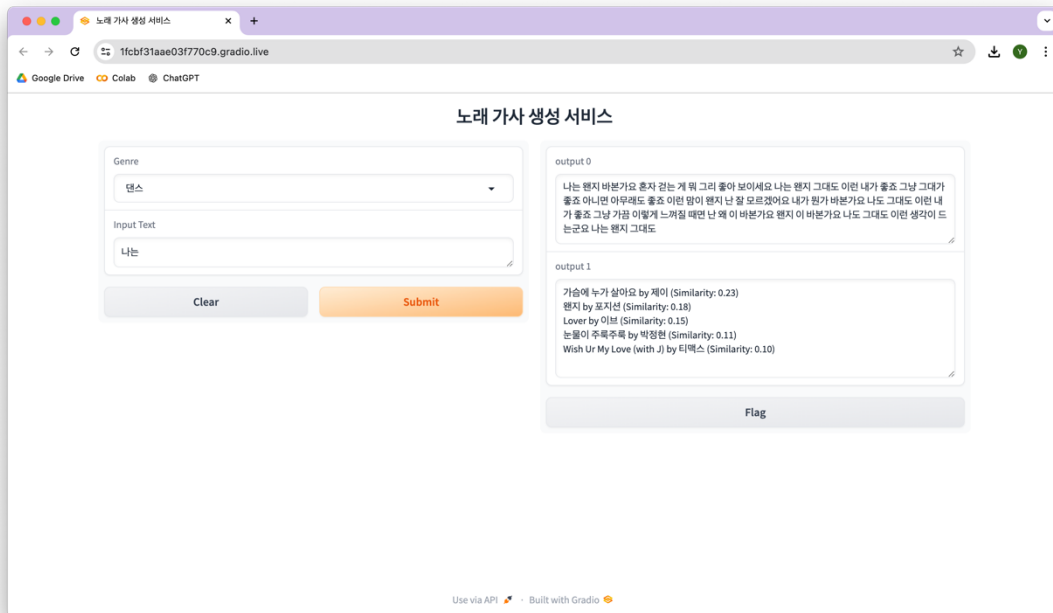




### 6-3 가사 입력



### 6-4 출력 (1: 생성된 가사, 2: 가장 유사한 5 개의 가수명과 곡명 및 유사도 점수)



## 7. 한계 및 발전 가능성

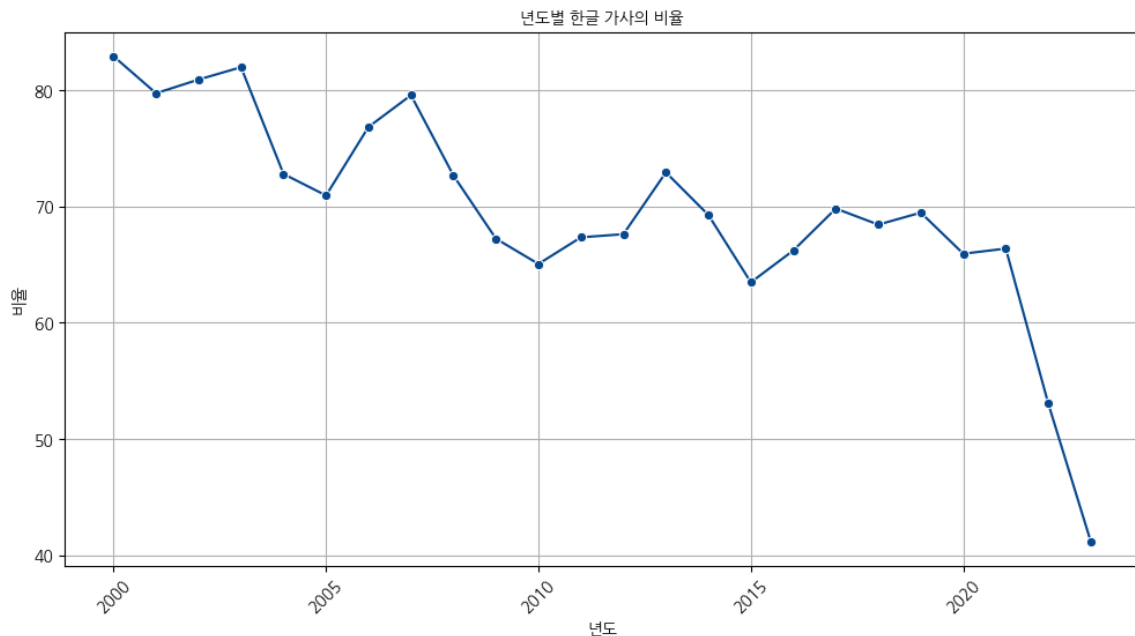
### 7.1. 데이터 부족

프로젝트에 사용된 데이터세트는 총 7519 개의 곡에 대한 가사로 이루어졌다. 순위권 내의 노래들을 담았기에 중복되는 노래들이 많았던 것이 문제로 작용했다. 자연어처리 학습의 관점에서 바라봤을 때는 상당히 부족한 크기의 데이터세트라는 것은 분명하다.

하지만 노래 가사 데이터는 상당히 많고 구축이 잘 되어있다. 본 프로젝트에서는 이미 다른 사용자가 웹 크롤링하여 구축한 데이터세트를 사용했지만, 추후 연구를 할 때는 ‘멜론’이라는 사이트를 비롯한 다양한 스트리밍 또는 웹사이트에서 데이터를 얻을 수 있다면, 더욱 더 다양하고 좋은 퀄리티의 가사를 내는 모델이 될 수 있을 것으로 기대가 된다. 또한 사용한 레이블은 ‘장르’였지만, 더욱 세분화해서 플레이리스나 라디오를 바탕으로 ‘사랑’, ‘위로’, ‘여행’ 와 같은 테마를 이용해서 학습을 한다면 가사의 내용이 더욱 더 원하는 것과 가깝게 생성될 수 있을 것이다.

### 7.2. 한/영 가사 비율

현재 노래 가사를 보면 한글과 영어를 혼용한 경우가 상당 수이다. 특히 K-Pop 의 장르에서는 한글 가사보다 영어 가사가 많은 경우도 흔하게 존재한다. 데이터의 한글/영어 비율을 분석해보면, 총 가사 중에 한글은 68.25%, 영어는 31.75%에 해당되었다. 시간이 지날수록 한글 가사의 비율이 적어지는 것도 볼 수 있었다. 특히 2023년에는 한글 가사 비율이 41.19%로 절반 이하인 모습을 보였다.



현재의 프로젝트 결과를 관찰했을 때, 한글과 영어를 혼합해서 모델을 학습을 시키기에는 성능이 부족한 것으로 보인다. 본 프로젝트를 발전시킨다면, 한글과 영어 가사의 비율에 따라 학습을 하고, 더욱 더 자연스러운 생성을 할 수 있을 것이다.