



Fine. : 하루 요약 피드 생성 챗봇
(부스트캠프 **AI Tech 5기 Level3 Final Project**)

NLP-05 : 브레멘 음악대
변성훈, 서보성, 이상민, 이승우, 이예원

<목 차>

I. 서론

- 1. 프로젝트 개요 ----- 3
- 2. 프로젝트 팀 구성 및 역할 ----- 3

II. 본론

- 1. 프로젝트 수행 절차 ----- 4
- 2. 프로젝트 수행 내용
 - 2-1 Model Pipeline ----- 4
 - 2-2 Modeling
 - 2.2.1 ChatBot ----- 5-11
 - 2.2.2 Diary Generation ----- 11-16
 - 2.2.3 Generation Style Transfer ----- 16-17
 - 2-3 Serving
 - 2.3.1 System Architecture ----- 18
 - 2.3.2 Front-End ----- 18-19
 - 2.3.3 Back-End ----- 19-20

III. 결론

- 1. 결과 ----- 21
- 2. 자체 평가 의견 ----- 21-22

IV. 개인회고

I. 서론

1. 프로젝트 개요

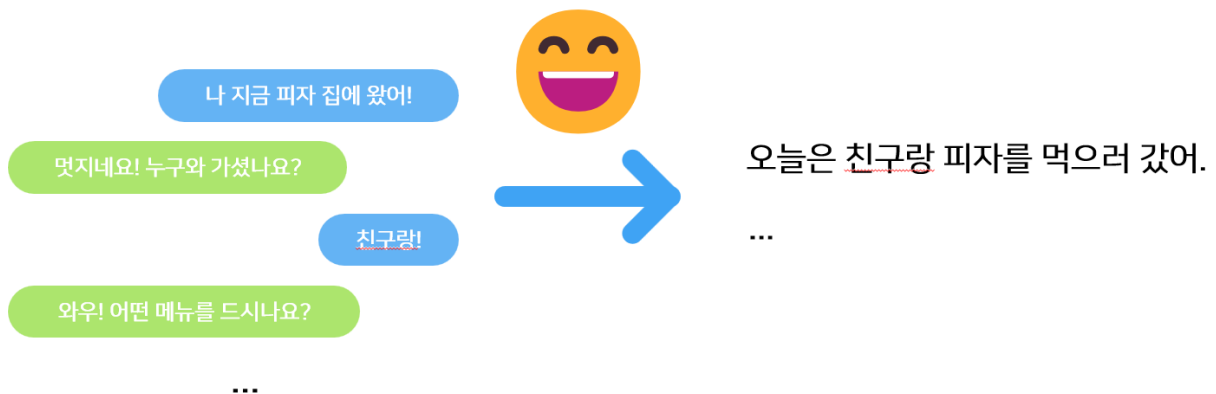
누구나 살다 보면 기록의 중요함을 자주 깨닫게 된다. 지난 일들을 떠올리고자 해도 기억이 안나는 경우가 정말 많고, 중요한 정보나 소중한 추억, 감정을 잊을 수도 있다. 이 경우 사람들은 옛날부터 일기를 써서 문제를 해결할 수 있었다. 현대엔 SNS도 존재하지만, 사소하거나 개인적인 일상까지 공유하긴 힘들기 때문에 일기를 대체할 수는 없다.

문제는, 일기를 매일 꾸준히 쓰기에는 현대인들은 너무 바쁜 데다, 일기를 쓰는 건 상당히 귀찮은 작업이기도 하다. 이 때문에 사람들은 보통 일기에 시간투자를 하진 않지만, 하루 종일 친한 친구들 및 가족들과 메신저 및 SNS로 수다를 떨면서 오늘 있던 일들을 말하고 있다.

위의 경우를 Motivation으로 하여, 친구와 수다를 떨듯이 오늘의 일들을 말하다 보면 자동으로 일기를 써주는 서비스 “Fine.”를 개발했다.

Fine.

하루 동안 챗봇과 대화하면, 하루의 끝에 일기를 생성해주는 서비스



2. 프로젝트 팀 구성 및 역할

변성훈: Generation Style Transfer Model & Back-End

서보성: Product Serving & Back-End

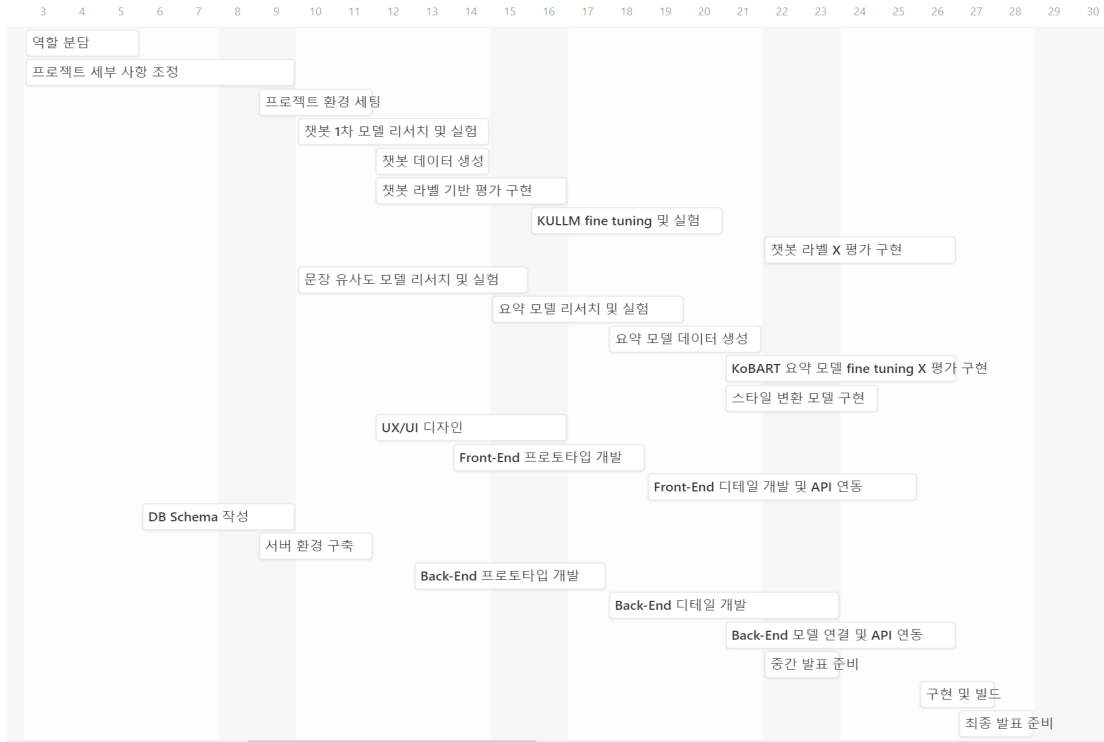
이상민: Summarization Model

이승우: Product Manager & Front-End

이예원: ChatBot Model

II. 본론

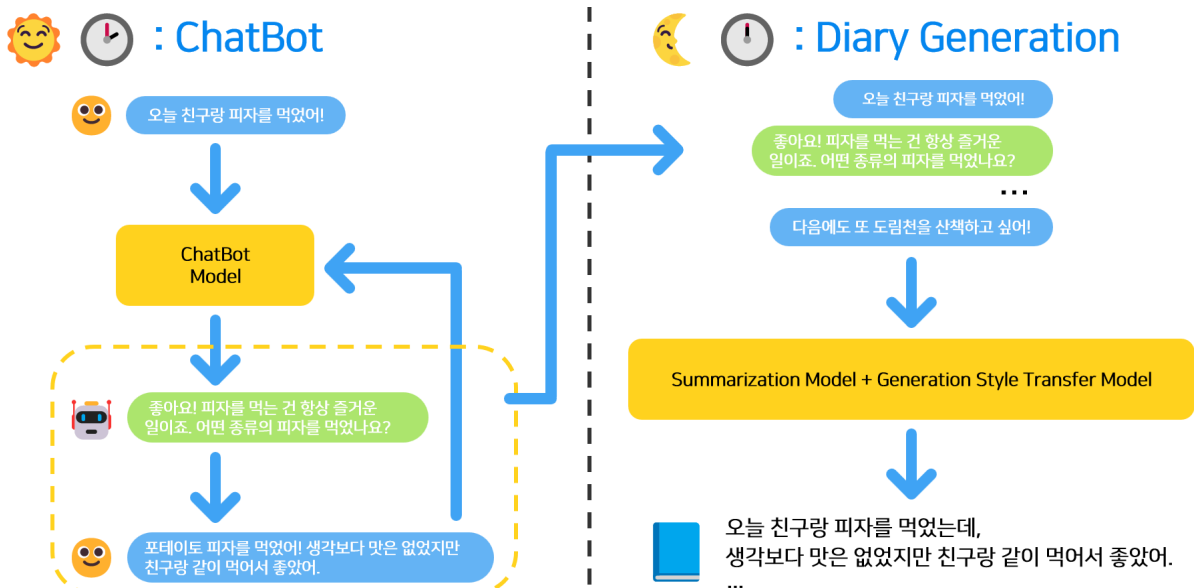
1. 프로젝트 수행 절차



2. 프로젝트 수행 내용

2-1 Model Pipeline

챗봇과 사용자의 자연스러운 대화를 통해 만들어진 대화 데이터를 이용해 하루를 요약 일기를 생성한다.



2-2 Modeling

2.2.1 ChatBot

초기에는 일기 생성이라는 목적 지향성 대화형 모델을 학습시키는 것에 어려움을 느껴 아래 프롬프트를 주고 **GPT API**를 이용해 대화를 진행할 예정이었다.

지금부터 일기 생성을 위한 대화를 진행할 것입니다.

요구사항은 다음과 같습니다.

1. 다양성을 극대화하기 위해 각 답변에 대해 동사를 반복하지 않도록 하세요.
2. 답변에 사용되는 언어도 다양해야 합니다. 예를 들어, 질문과 리액션을 결합해야 합니다.
3. 답변은 한국어로 작성해야 합니다.
4. 답변을 **1~3**문장으로 작성하세요. 답변에는 무조건 질문이 포함되어야 합니다.
5. 답변은 구어체로 답변해야 합니다.
6. 오늘 하루에 대한 정보를 질문을 통해 유도해야 합니다.
7. 출력은 명령어와 입력에 대한 적절한 응답이어야 합니다.

위의 **7**개의 명령어에 따라 일기 생성을 위한 적절한 대화를 저와 시작해주세요!

대화의 시작은 오늘 하루에 대한 간단한 질문으로 시작하면 됩니다!

해당 프롬프트를 주고 여러번 대화를 나눠 본 결과 **GPT**로도 목적에 맞는 대화가 충분히 가능하다는 점을 확인하였으나 **token** 당 가격을 측정하는 방식으로 인한 비싼 비용과 **open API** 서버의 불안정성 문제 때문에 **API** 사용을 포기하고 **GPT**의 대화를 **fine tuning** 모델로 재현하는 것을 목표로 하였다.

1) Data

AI HUB 감성 대화 말뭉치, **AI HUB** 주제별 일상 대화 데이터, **GPT**를 이용한 생성 데이터 이 세가지로 학습시켜본 결과 최종적으로는 **GPT**를 이용한 생성 데이터를 사용하게 되었다.

AI HUB 감성 대화 말뭉치 (질문만)	AI HUB 주제별 일상 대화 데이터
Bot: 안녕하세요! 오늘 하루는 어떠셨나요? 무슨 재미있는 일이 있었나요? User: 오늘 친구랑 피자를 먹었어. Bot: 친구분과 피자를 드셨군요. 무슨 일	Bot: 안녕하세요! 오늘 하루는 어떠셨나요? 무슨 재미있는 일이 있었나요? User: 오늘 친구랑 피자를 먹었어. Bot: 오늘 친구랑 피자 먹었어?

있으셨나요? User: 아니! 그냥 피자가 먹고 싶어서 먹은거야. Bot: 그냥 피자가 먹고 싶어서 먹은거군요? User: 응 맞아! Bot: 그냥 피자가 먹고 싶어서 먹은거군요?	User: 응. 포테이토 피자를 먹었는데, 별로 맛은 없었지만 친구랑 같이 먹어서 좋았어. Bot: 포테이토 피자 먹었어? User: 정말 맛있더라...! Bot: 응응 맛있었어!
---	---

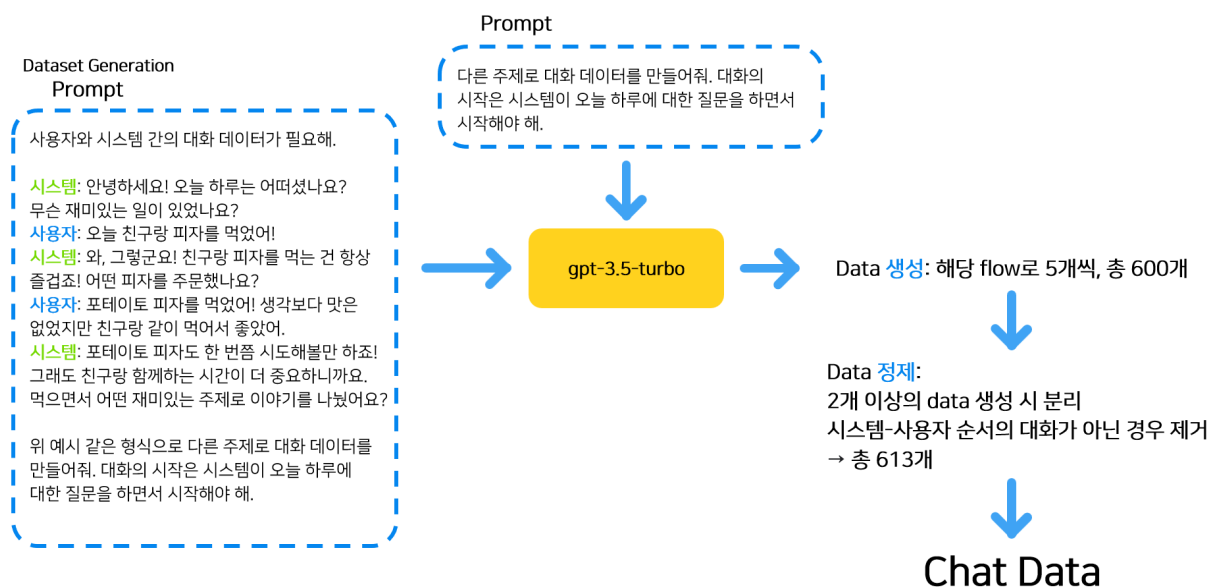
- 감성 대화 말뭉치

감성 대화 말뭉치 데이터 전체를 학습할 경우, **User**의 응답을 반복하여 리액션하는 단조로운 응답만을 생성하여 시스템 답변의 유형이 질문일 경우만 학습시킨 경우가 더 자연스러웠다. 그럼에도 상당 데이터 기반이라는 특징과 질문의 유형이 다양하지 않다는 특징으로 인해 최종적으로 사용하지 않았다.

- 주제별 일상 대화

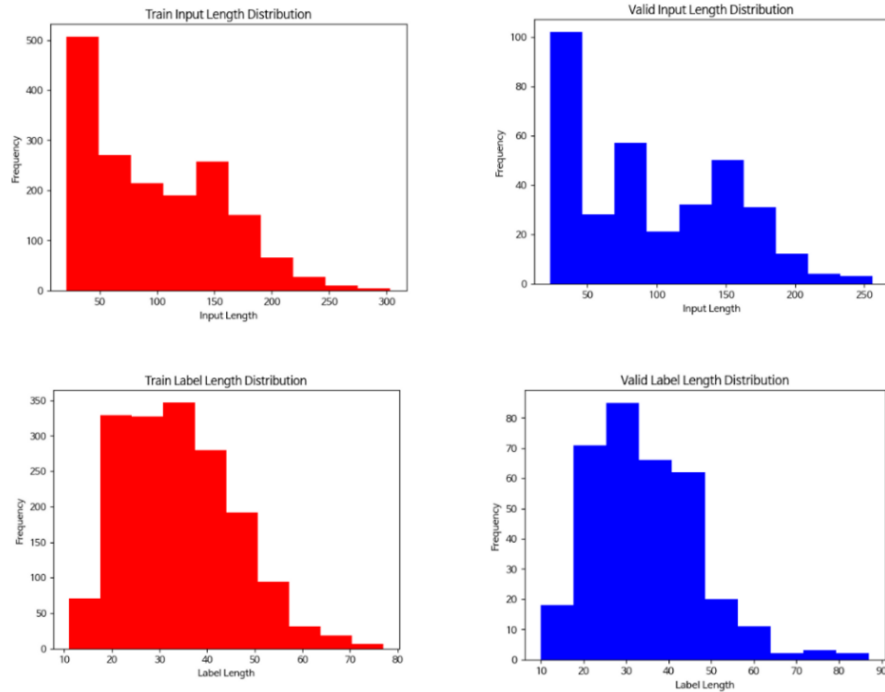
감성 대화 말뭉치의 데이터가 대화 turn마다의 상관관계가 별로 없어 학습한 모델이 이전 대화와의 관계를 학습할 수 없고, 답변이 단조롭다는 점을 보완하기 위해 주제별 일상 대화 데이터를 학습시켜 보았다. 그러나 답변의 유형에 일관성이 있는 것도 아니며 사람조차도 답변을 예상하기 힘들 만큼 다양성을 가진 데이터 특성을 모델이 학습하는 것은 어려운 것으로 보였다.

- GPT 생성 데이터 ★



초기 프롬프트에 주어진 예시는 **GPT API**를 이용하려 했을 때 사용한 프롬프트로 대화를 진행했을 때의 예시이다. 위 **flow**를 5번씩 반복하여 총 600개의 대화 데이터를 생성하였고, 답변이 중간에 끊긴 데이터, 한번에 2묶음 이상의 대화가 생성된 데이터, 시스템-사용자 순서로 대화가 진행되지 않은 데이터 등을 처리한 결과 총 613개의 대화 데이터를 얻을 수 있었다.

얻은 대화 데이터를 모델이 학습하기 위한 형태로 바꿔주기 위해서 **label**로는 시스템 응답을 **input**으로는 해당 시스템 응답 이전의 최대 **3 turn**의 대화를 **concat**한 정보로 사용하였고 그 결과 **train 1696개, test 340개**의 데이터로 구성되었고 **gogamza/kobart-base-v2 tokenizer** 기준 길이 분포는 아래와 같다.



2) Evaluation

Input 마지막 문장
user: 네, 많은 사람들이 이 책의 신비로운 분위기와 밀도 높은 스토리에 감탄했어요. 또한, 작가의 솜씨와 문체에 대해 이야기하며 새로운 시각을 얻을 수 있었어요.
Label
system: 작가의 솜씨와 문체는 정말 중요한 요소죠. 독서 모임에서는 다양한 시각을 얻을 수 있어서 좋았을 것 같아요. 다음 독서 모임에 어떤 주제의 책을 함께 읽고 이야기하고 싶으신가요?
Prediction
system: 독서 모임은 책을 읽고 이야기를 나누는 특별한 경험이에요. 작가의 솜씨와 문체에 대해 이야기하며 새로운 시각을 얻을 수 있죠. 독서 모임에서 다른 재미있는 이야기로 대화를 이어나갔나요?

- Reference Based Metric

GPT 데이터에서 대부분의 시스템 응답이 유저에 대한 ‘간단한 리액션 + 유저 응답 재서술 + 맥락에 맞는 질문’으로 구성되어 있기에 학습이 잘 되었다면, 필수적으로

Label과 Prediction과의 유사한 점이 있을 수밖에 없었고 따라서 라벨을 참조하는 평가 메트릭을 사용하였다.

대화 모델의 성능을 평가하는 것이 번역보다는 요약 task에 더 가깝다고 생각해

BLUE Score 가 아닌 ROUGE Score를 사용하였고 한국어가 어순이 중요하지 않다는

특성을 고려해 ROUGE-2나 ROGUE-L이 아닌 ROGUE-1 과 ROGUE-SU를

사용하였다.

ROUGE-1 tokenizer → Mecab

Recall = target과 prediction의 공통 **unigram**의 수 / target 의 unigram 수

Precision = prediction에서 target의 공통 **unigram**의 수 / prediction의 unigram 수

ROUGE-SU max skip → 2

Skip Bigram + Unigram

(ex) target: the brown for jumps → [the, brown, for, jumps, (the, brown), (the, for),

(brown, for), (brown, jumps), (for, jumps)]

prediction: the fox → [the, fox, (the, fox)]

→ **Recall = 1/9, Precision = 1/4, F1 = 0.1538**

토큰이 겹치는 지로 라벨과의 유사도를 재는 것으로는 불충분하다 판단하여 의미적 유사도도 볼 수 있는 jhgan/ko-sbert-sts를 이용한 Cosine Similarity도 평가 메트릭으로 사용하였다.

- Reference Free Metric

대화 모델 특성상 라벨과 다르더라도 충분히 올바른 답변이 될 수 있기에 라벨을

참조하지 않는 평가 또한 필요하여 Perplexity와 G-Eval을 사용하였다.

Perplexity - 문장의 길이로 정규화된 문장 확률의 역수: 모델의 **헛갈림**의 수치

$$PPL(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, w_3, \dots, w_N)}} = \sqrt[N]{\frac{1}{\prod_{i=1}^N P(w_i | w_1, w_2, w_3, \dots, w_{i-1})}}$$

G-Eval - gpt-3.5-turbo 모델을 이용하여 생성 모델 평가

시스템과 유저의 대화 내역(**Conversation History**)을 받게 될 것입니다. 그리고

대화 내역에 대한 시스템의 응답(**Response**)이 제시됩니다.

당신의 작업은 평가 단계에 따라 응답을 평가하는 것입니다.

이 평가 기준을 꼼꼼히 읽고 이해하는 것이 중요합니다. 평가하는 동안 이 문서를 계속 열어두고 필요할 때 참조해 주세요.

평가 기준:

- 이해 가능성 (0 - 1): *Conversation History*에 기반하여 *Response*를 이해 할 수 있나요?
- 맥락 유지 (1 - 3): *Conversation History*를 고려했을 때 *Response*가 맥락을 유지하나요?
- 흥미롭기 (1 - 3): *Response*가 지루한가요, 아니면 흥미로운가요?
- 전반적인 품질 (1 - 5): 위의 답변을 바탕으로 이 발언의 전반적인 품질에 대한 인상은 어떤가요?

평가 단계:

1. *Conversation History*와 *Response*를 주의깊게 읽습니다.
2. 위의 평가 기준에 따라 *Response*를 평가합니다.

Conversation History:

{{Conversation History}}

Response:

{{response}}

Result

- 이해 가능성 (0 - 1):
- 맥락 유지 (1 - 3):
- 흥미롭기 (1 - 3):
- 전반적인 품질 (1 - 5):

위 프롬프트를 사용해 얻은 **Score**는 대체로 정수이기에 더 정교한 평가를 위해 원래 **Score**에 대한 예측 확률을 곱해주어야 하였지만 현 **gpt API**에서 **logits**이나 **probs**를 반환해주는 기능이 없었다. 따라서 해당 논문 레포에서 20개의 답변을 평균내리는 방법을 참고해 3개의 답변을 평균하여 사용하였다.

3) Result

1차 모델 선정

초기에는 LLM을 사용하는 것이 부담스러울 것이라 생각해 **KoGPT2**, **Ko-DialoGPT**, **KoBART**, **T5** 4개의 모델을 알아보았다. **AI HUB** 감성 대화 데이터셋으로 학습하였을 때, 대부분의 응답이 유사하였지만 답변의 길이가 적당히 길고 문맥을 어느정도 이해하는 것으로 보이는 **KoBART**로 이후 실험을 진행하였다.

- Data Input Style

(1) EOS token 구분

안녕하세요! 오늘 하루는 어떠셨나요? 무슨 재미있는 일이 있었나요? </s>

오늘은 공원에서 산책을 했어요. </s>

(2) System과 User로 구분

system: 안녕하세요! 오늘 하루는 어떠셨나요? 무슨 재미있는 일이 있었나요?

user: 오늘은 공원에서 산책을 했어요.

	ROUGE-1	ROUGE-SU	Similarity	Perplexity
(1)	39.357	26.402	77.41	1.85662
(2)	43.837	29.29	78.034	1.62204

input data를 concat하는 style을 위 두 가지로 실험하였고 최종적으로 **system**과 **user**를 구분하는 방식을 채택하였다.

- KoBART 학습 결과

유저의 응답에 대한 적절한 답변이 어려울 경우 이전 시스템 응답을 반복하는 경우가 잦으며 이는 결국 문맥 파악을 못하는 것에 근본적인 원인이 있었다. 이 문제를 해결하기 위해 한국어 LLM의 필요성이 대두되었다.

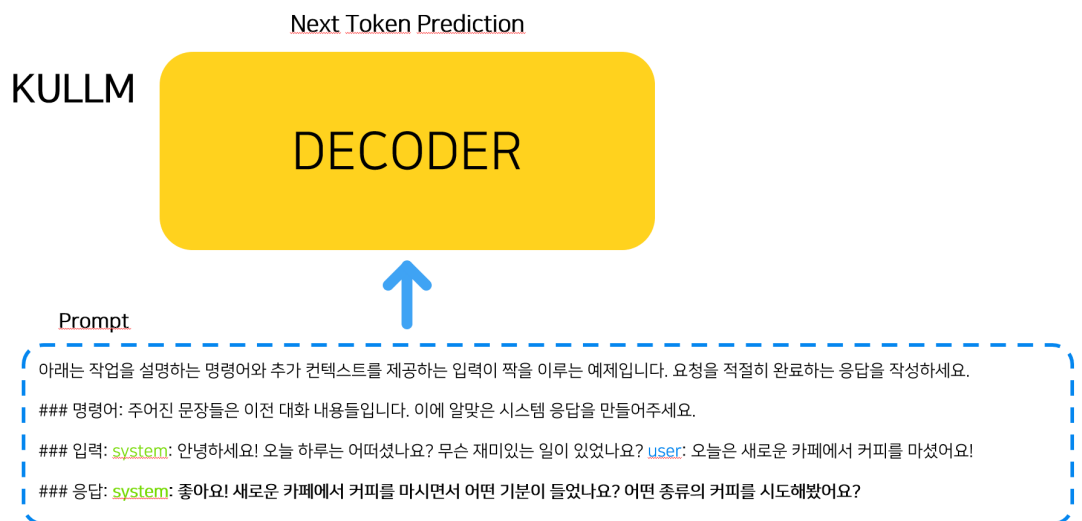
- 2차 모델 선정

KoAlpaca, KoVicuna : 대화보다는 정보 제공용 답변에 특화되어 있어 평서문으로 사용자가 답변하는 것에 적절한 응답을 제공해주지 않았다.

KULLM: 가끔 답변이 무한히 길어지는 단점이 있기는 했지만 그나마 일상 대화에 적합한 답변을 제공하였고 이 단점은 **fine-tuning**으로 해결할 수 있었다.

- KULLM fine tuning 방식

KULLM 모델의 Instruction Tuning과 Parameter-Efficient Fine Tuning



KULLM은 Instruction Tuning이 된 모델로 **fine tuning** 단계에서도 위와 같이 **prompt**를 구성하여 **Instruction**을 주었다. 또한 **embed out**과 마지막 **layer**를 제외한 모든

parameter를 freezing 하여 fine tuning을 진행하였다. 이 과정에서 label의 loss만을 계산하여 시스템 응답 기능에 더 초점을 맞추었고 parameter freezing으로 사전학습된 일반화 지식 활용 성능을 기대하였다.

- 최종 모델 선정

	ROUGE-1	ROUGE-SU	Similarity	Perplexity
KoBART	43.837	29.29	78.034	1.62204
KULLM	45.227	30.695	80.917	1.36967

G-Eval

	이해 가능성	맥락 유지	흥미롭기	전반적인 품질
KoBART	0.997	2.937	2.264	3.406
KULLM	0.999	2.957	2.275	3.459
GPT	0.999	2.953	2.315	3.525

실제 여러번 대화를 진행해보았을 때, KULLM이 KoBART에 비해 답변이 자연스럽고 합리적이었으며 모든 평가지표에서 더 좋은 점수를 보였다. 또한 GPT로 생성한 데이터로 G-Eval Score와 비교해도 크게 뒤처지지 않고 맥락 유지 측면에서는 더 높은 스코어를 보였기에 최종 모델로 KULLM을 선정하였다.

- 한계점 및 후처리

학습 데이터가 대부분이 의미있는 정보로 구성되어 있기 때문에 길이가 짧은 경우가 거의 없었다. 따라서 실제로 짧은 답변이 들어왔을 경우 적절한 답변 생성이 이루어지지 않아 답변의 길이가 6이하일 경우 미리 정해진 응답으로 다른 주제로 넘어가도록 하였다.

2.2.1 Diary Generation

1) Text Similarity

사용자의 대화 데이터 -> 주제 별 분류(Text Similarity) -> 각 주제별 대화 데이터를 요약
-> 요약 문장들을 합쳐 일기 생성

<개요>

프로젝트 초기 챗봇은 리액션 중심 대화 모델이었다. 그렇기에 챗봇이 대화의 흐름 및 주요 내용을 바꾸어 주지 않았고, 사용자의 대화 데이터만을 가져와도 충분히 하루를 요약할 수 있다고 판단. 사용자의 대화 데이터만을 가지고 일기 생성을 시도하였다.

사용자 대화 데이터 요약 시, **input**의 길이와 무관하게 내용을 한 줄로 요약해주는 **gogamza/kobart-summarization** 이용하였다. 다양한 주제를 가지거나, 길이가 긴 **input**이 요약 모델에 들어가면 생략되는 부분이 증가되며 주요 내용이 생략되기도 하였다. ‘일기가 한 문장으로 생성’되는 것도 이상하고, ‘삭제되는 부분이 많아질 수 있다’는 문제가 발생하였고, 따라서 대화 데이터를 주제별로 나눠 요약 모델에 여러 개의 **input**을 넣어주면서 주제별 **output**을 생성하였다. 또한 주제 별로 여러 문장의 **output**이 생성되기 때문에 ‘한 문장 일기’와 ‘주요 내용 생략’이라는 문제를 해결하였다. 이후에 여러 줄의 **output**을 **concat**하여 하루 요약 일기를 생성하였다.

예를 들어, 햄버거를 먹은 일을 요약하는 상황에서 햄버거의 구성과 맛에 대한 주제 하나, 햄버거 집에 대한 정보 하나, 같이 먹은 사람들에 대한 정보 하나 등 세부적인 주제로 구분해준 뒤, 각 주제 별 대화 데이터를 모델의 **input**으로 넣고 주제 별 요약 문장을 얻어 합쳐주며 하루 요약 일기를 생성하였다.

<데이터셋>

kakaobrain/kor-nlu-dataset의 KorSTS dataset

<평가 방법>

pearson score : 두 변수간의 상관 관계를 계량화한 수치. 코시-슈바르츠 부등식에 의해 +1 ~ -1 사이의 값을 가지며, +1은 완벽한 양의 선형 관계, 0은 선형 관계 없음, -1은 완벽한 음의 선형 관계를 의미한다.

<모델>

(1) KoBART Encoder

요약 시 이용되는 **data preprocessing** 과정을 줄이기 위해서 요약과 같은 모델

KoBART의 Encoder 부분으로 유사도 평가 모델 학습

Eval Loss(dev) : 1.422

(2) klue/roberta-large ★

Eval Loss(dev) : 0.5809, pearson score(test dataset) : 0.8694, storage size : 2.54GB

(3) monologg/koelectra-base-v3-discriminator

Eval Loss(dev) : 0.5787, pearson score(test dataset) : 0.8536, storage size : 862MB

(4) snunlp/KR-ELECTRA-discriminator

Eval Loss(dev) : 0.5890, pearson score(test dataset) : 0.8311, storage size : 832MB

<결과>

1. 유사도 평가 모델, 요약 모델을 각각 이용하며 이용하는 모델의 수 증가하여 **cost**가 증가한다.
2. 요약 모델 이용 시 생기는 오류와 주제 별로 요약된 데이터를 부드럽게 이어붙이는 **fine-tuning** 데이터셋을 구하지 못하였다.
3. 프로젝트 도중 챗봇의 성능 향상으로 챗봇이 대화의 흐름을 바꿀 수 있는 발화를 하게 된다. 따라서 하루에 대한 정보 요약 시, 챗봇의 대화 데이터도 필요하다.

따라서 새로운 방법으로 **Diary Generation**을 시도했다.

2) KoBART-Summarization Fine-Tuning

대화 데이터를 input으로 받고 여러 줄의 요약 데이터를 반환하는 **fine-tuning** 진행

<개요>

앞 선, **text similarity** 기반 일기 생성 모델의 단점인 **cost** 증가와 챗봇의 대화 데이터를 이용하지 못한다는 단점을 해결하기 위해 문장 유사도 평가 모델을 없애고, 'gogamza/kobart-summarization' 모델을 Input data를 대화 데이터 형식으로 output을 여러 줄의 요약 데이터로 **fine-tuning**.

<데이터셋>

gpt turbo 3.5를 이용해 만든 대화 데이터셋과 **chat gpt**에 대화 데이터를 넣어 생성한 대화 요약 데이터셋 : **350**개

-> 이상 데이터 (대화 데이터셋 중 대화가 중간에 끊어진 데이터) 제거 : **9**개

-> 총 **341**개 데이터 생성

[생성 프롬프트]

###대화를 보고 사용자에게 대한 하루를 요약해줘

<대화 데이터>

대화를 보고 사용자에게 대한 하루를 요약해줘

시스템: 안녕하세요! 오늘 하루는 어떠셨나요? 무슨 재미있는 일이 있었나요?

사용자: 오늘은 새로운 공원을 방문했어요.

시스템: 새로운 공원 방문은 정말로 즐거운 활동이네요! 어떤 종류의 공원이었나요?

사용자: 오늘은 수목이 아름답게 조성된 자연 공원에 갔어요. 싱그러운 공간에서 산책하며 휴식 시간을 보낼 수 있었어요.

시스템: 수목이 아름답게 조성된 자연 공원은 자연의 아름다움을 느낄 수 있는 좋은 장소입니다. 산책하면서 어떤 경치를 감상할 수 있었나요?

사용자: 공원에는 시원한 연못과 다채로운 꽃들이 있는 정원이 있었어요. 산책하면서 그 경치를 감상하니 마음이 편안해지고 기분이 좋았어요.

시스템: 공원의 시원한 연못과 다양한 꽃들을 감상하면서 마음을 편안하게 만들 수 있었을 거예요. 공원에서 다른 사람들과 함께했나요?

사용자: 공원을 친구들과 함께 방문했어요. 함께 산책하고 이야기하면서 즐거운 시간을 보냈어

결과) summarized data

사용자는 오늘 새로운 수목이 아름답게 조성된 자연 공원을 방문하여 시원한 연못과 다양한 꽃들이 있는 정원을 감상하며 산책하고 휴식을 즐겼습니다. 이 특별한 경험을 친구들과 함께 나누면서 즐거운 하루를 보냈습니다.

<학습 방법>

대화 데이터이기에 발화자가 중요하다고 판단하여 input data를 모두 동일한 형식으로 만들어 주었다.

대화 데이터를 모델에 넣을 때, 사용자와 챗봇을 special token([USER], [SYSTEM])으로 만들어 주었다. 그리고 '사용자'의 하루를 요약해야하기에 '[USER]'의 하루를 요약해줘.'라는 내용을 마지막에 공통적으로 넣어주었다.

Input data formatting)

<s>[SYSTEM]:챗봇말1[USER]:사용자말1[USER]:챗봇말2 ... \n###[USER]의 하루에 대해서 요약해줘:</s>

예시)

<s>[SYSTEM]:안녕하세요! 오늘 하루는 어떠셨나요? 무슨 흥미로운 일이 있었나요?
[USER]:오늘은 독서 클럽 모임에 참여했어요![SYSTEM]:독서 클럽 모임은 정말 좋은 방법이죠! 어떤 책을 읽고 어떤 주제로 이야기를 나눴나요? [USER]:저희는 '1984'라는 책을 읽고 토론했어요. 인터넷 감시, 빅 브라더, 독재 등의 주제로 이야기를 나눴는데 정말 흥미로웠어요. \n###[USER]의 하루에 대해 요약해줘:</s>

<평가방법>

Rouge score

정답(summary reference)와 요약(predict summary)의 n-gram recall과 precision을 구해서 f1 score 계산.

Tokenized by 'Mecab'

- Rouge - 1 (1-gram)

ex)

reference sentence : 나는 TV와 오토바이를 샀다.

predict sentence : 나는 자동차를 샀다.

reference -> {'나는', 'Tv와', '오토바이를', '샀다'}

predict -> {'나는', '자동차를', '샀다'}

recall = 2/4 & precision = 2/3 -> f1 score = 4/7

- Rouge - SU (Skip Bigram + Unigram, skip_max = 2)

ex)

reference sentence : 나는 TV와 오토바이를 샀다.

predict sentence : 나는 자동차를 샀다.

reference -> {'나는', 'Tv와', '오토바이를', '샀다', ('나는', 'TV와'), ('나는', '오토바이를'), ('TV와', '오토바이를'), ('TV와', '샀다'), ('오토바이를', '샀다')}

predict -> {'나는', '자동차를', '샀다', ('나는', '자동차를'), ('나는', '샀다'), ('자동차를', '샀다')}

recall = 2 / 9 & precision = 2 / 6 -> f1 score = 2 / 15

<모델 선정 이유>

1. LLM에 비해 작은 사이즈
2. GPT API를 쓰기엔 금전적 부담
3. 생성 모델보다 요약 모델이 하루 요약 일기에 더 부합하다고 판단

<Hyperparameters>

lr	batch_size	epoch	EVAL : ROUGE-1	EVAL : ROUGE-SU
1e-4	32	25	0.7076	0.4969
1e-4	64	25	0.7208	0.5162
3e-4	64	25	0.7215	0.5178
1e-4	64	30	0.7079	0.5074
9e-5	64	25	0.7178	0.5112
7e-6	64	25	0.6840	0.4920

<결과>

Kobart Model의 단독 사이즈는 약 500MB 정도로 모델 사이즈 경량화가 되었고, 챗봇의 대화 데이터도 받아와서 대화 정보를 모두 이용해 요약 데이터 생성 & 여러 줄로 데이터 생성이 가능해졌다.

하지만 단점으로는 요약 시, 같은 내용의 불필요한 중복이 발생하기도 했다.

2.2.3 Generation Style Transfer

현재 챗봇 모델과 요약 모델이 존재하므로 Style Transfer 모델은 가능하면 이미 fine-tuning되어 있는 모델을 사용하고자 하였다.

우선 style transfer 관련 한국어 데이터셋이 존재하는지 찾아보았고, smailgate에서 발표한 **Korean SmileStyle Dataset** 이 존재한다는 것을 알게 되어 이 데이터셋으로 훈련을 진행한 모델을 조사해 보았다.

huggingFace에 **NHNDQ/bart-speech-style-converter** 모델이 위에 언급한 데이터셋으로 모델을 훈련시켰고, 다양한 style로 변환이 가능하여 실험을 진행하였다.

- BLEU Score: **25.35**
- Style: ["문어체", "구어체", "안드로이드", "아재", "채팅", "초등학생", "이모티콘", "enfp", "신사", "할아버지", "할머니", "중학생", "왕", "나루토", "선비", "소심한", "번역기"]
- 실험 결과

input_text: {style} 형식으로 변환:오늘은 닭볶음탕을 먹었다. 맛있었다.)

style	output
문어체	오늘은 닭볶음탕을 먹었어. 맛있었어.
구어체	오늘은 닭볶음탕을 먹었어. 맛있었어.
안드로이드	오늘. 닭.볶음. 먹음. 맛있음.
아재	아재 : 오늘 닭 닭볶음탕 먹었어~ 맛있었어~

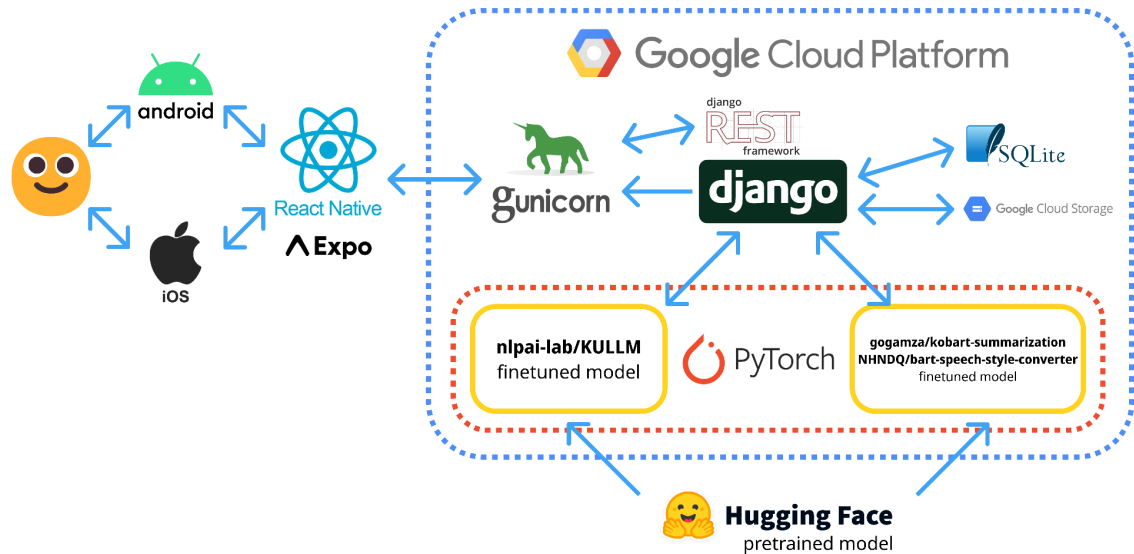
채팅	오늘은 닭볶음탕 먹음 맛있음
초등학생	오늘 닭볶음탕 먹음 맛있음
이모티콘	오늘은 닭볶음탕 먹었어! (★~★)
enfp	오늘 닭 닭볶음탕 먹구 맛있영!!
신사	오늘은 닭볶음탕을 먹었습니다. 맛있었습니다.
할아버지	오늘은...볶음탕을 먹었구먼...맛있었구먼...
할머니	오늘 닭 닭볶음탕 쳐먹었네
중학생	오늘 닭볶음탕 먹었는데 맛있음
왕	금일은 닭볶음탕을 먹었소. 맛있었소.
나루토	오늘은 닭볶음탕을 먹었다니깐! 맛있었다니깐!
선비	오늘은 닭볶음탕을 먹었소! 맛있었소!
소심한	오늘은 닭볶음탕 먹었는데.. 맛있어..
번역기	오늘 나는볶음탕을 먹었고, 그것은 맛있었다.

추가로 요약 모델의 **output**을 변환하는 실험을 진행한 결과 구어체가 원본 훼손이 적고, 변환 결과도 일관성있다고 판단하여 구어체를 채택하였다.

후에 **style**을 사용자가 직접 선택하여 일기를 생성할 수 있도록하는 기능을 구현할 계획이다.

2-3 Serving

2.3.1 System Architecture



2.3.2 Front-end

1) React Native Expo

Front-end framework로는 React Native Expo를 사용했다. React Native는 Android, iOS App 개발을 위한 Cross-Platform framework로, 추가 framework인 expo를 사용하면 초기 설정이 단순하고 큰 작업 없이 cross-platform으로 편하게 App을 빌드할 수 있다. 서비스의 특성 상 web이 아닌 app 기반이어야 하므로 cross-platform이면서 빠르고 간편하게 개발할 수 있고, 다루어본 적이 있었기 때문에 선택했다.

2) UX/UI (Design Concept)

실제 서비스를 가정하여, 타겟이 젊은 층이고, 감성적인 주제이므로 user needs에 맞게 UX/UI 및 디자인의 concept를 정했다.



특히, 하루를 기록하는 앱 특성에 맞게 시간대 별로 color theme가 다르게 표시되도록 개발했다.

2.3.3 Back-end

1) Google Cloud Platform

chatbot의 경우 serving을 할 때, 즉각적인 inference 과정이 필수적이므로, GPU 사용이 가능하고 cloud computing 서비스를 일부 무료로 제공받을 수 있는 google cloud platform을 사용했다.

2) Gunicorn

구현된 웹 어플리케이션 서버와 웹 서버 간의 통신을 중개하기 위해서 사용한 wsgi 서버로 여러 웹 어플리케이션 프레임워크와 호환성이 뛰어나고 안정적인 운영이 가능하다는 점에서 gunicorn을 사용했다.

3) Django REST Framework

Python 웹 프레임워크인 Django를 확장하여 RESTful API를 구축할 수 있게

해주는 라이브러리로 쉬운 접근성과 높은 범용성을 가진다는 점에서 Django REST Framework를 사용했다.

4) **SQLite**

응용 프로그램에 임베디드되어 동작하는 DBMS 소프트웨어로 별도의 서버 프로세스가 필요하지 않고 경량이면서 빠른 처리속도와 높은 호환성의 특징을 이유로 SQLite를 사용했다.

5) **Google Cloud Storage**

객체(이미지)를 저장하기 위한 클라우드 기반의 서비스로 데이터의 내구성이 뛰어나며 암호화를 지원하여 보안이 좋기 때문에 google cloud storage를 사용했다.

Ⅲ. 결론

1. 결과



그림은 실제 App 구동 화면을 캡처한 것이다. 조금 더 자세한 user flow는 [Demo Video](#)에서 확인할 수 있다.

2. 자체 평가 의견 (개선 및 발전 방향)

• Modeling

- 챗봇 데이터의 생성 방법이 단순해 데이터의 다양성이 높지 않았다. 따라서 데이터 생성 flow를 재구성해서 응답 유형을 다양화할 수 있을 것이다.
- KULLM 모델의 사이즈로 인해 GPU 사용이 필수적이다. 하지만 유지 비용이 크므로 이에 대한 해결책이 필요하다.

- summarization 시, 동일한 내용을 중복으로 생성하는 경우가 많았다. 이 때, 같은 내용을 중복적으로 말하는 것은 **ROUGE score**에 적극적으로 반영되지 않아 단순히 높은 **ROUGE score**의 모델을 선택하는 것에 무리가 있었다. 이에 따라 다른 평가 지표 이용하고, 모델 성능의 개선 역시 필요할 것이다.

- **Service**

- 대화 내용 감성 분석을 바탕으로 그 날에 어울리는 노래, 미디어 등 콘텐츠 추천 기능을 추가할 수 있다.
- **QA Task** 적용을 통한 사용자의 문장형 질문에 알맞는 일기 내용을 검색하는 기능을 추가할 수 있다.
- **Generation Style Transfer** 모델을 활용해 챗봇 및 일기 말투 커스터마이징이 가능할 것이다.
- 음성 인식, 결제 내역, 지도 등 외부 **API** 연결을 통한 다양한 기능 추가가 가능하다.
- 예시로 든 기능 이외에도 더욱 다양한 기능 추가 및 구독 시스템 등 수익 모델 기획을 통해 상업성을 도모할 수 있다.

IV. 개인 회고

1. 변성훈

- 말은 역할
 - 보성님과 함께 Django를 이용해 **Back-End** 기본 틀을 잡아두었습니다. AI모델을 개발하는 기간동안 **Back-End** 틀을 잡아둔다면 후에 모델을 **API**에 적용시켜 프로젝트를 완성하는 시간을 단축시킬 수 있을 것이라 생각했습니다.
 - 요약 모델에서 나온 문장의 말투를 변경하는 모델을 담당했습니다. 우선 데이터셋을 찾고, **huggingface**에 **fine-tuning** 되어 있는 모델이 있는지를 조사하였습니다. 운이 좋게도 **smilegate**에서 공개한 말투 변경 데이터셋이 존재했고, 이 데이터셋으로 훈련한 모델 또한 **huggingface**에 있었기에 몇 가지 실험을 진행한 후 그대로 써도 될 것 같다고 판단하였습니다.
- 어려웠던 점
 - 백엔드 틀을 잡는 작업에서 주로 **DB**를 담당하였는데 해당 작업을 진행하면서 **API**에 연동시키기 위해 적절한 **field**를 선택하고, 아직 확정되지 않는 AI모델의 **field**를 정해주는 과정에서 어려움을 겪었습니다.
 - **huggingface**에 존재하는 모델 자체의 성능이 우수한 것은 맞지만 요약 결과물에 따라 내용이 왜곡되거나 정보가 손실되는 경우가 있었습니다. 이에 따라 **style transfer** 모델이 요약 모델에 큰 의존성을 가진다는 것을 확인할 수 있었습니다.
- 느낀 점
 - 웹 서비스에 AI모델을 연동시키는 것 자체가 하나의 큰 도전과제라는 것을 느꼈습니다. 요즘 핫한 **GPT**는 모델 자체가 너무 무거우며, **API**는 비용 지출을 해야하기 때문에 가벼운 프로젝트에 사용하기에 부담스러웠습니다.
 - 대회 프로젝트와 **product**를 만드는 프로젝트는 완전히 다르다고 생각하게 되었습니다. 개개인의 개발 실력도 중요하지만 특히나 프로젝트 기간 동안 커뮤니케이션이 원활하게 이루어지는 것이 더욱 팀에 필요한 요소라고 느꼈습니다.

2. 서보성

이번 프로젝트에서 나의 목표는 무엇이었는가?

이번 프로젝트에서는 성훈님과 같이 Django를 이용해서 Back-End 기본틀을 잡는 것을 목표로 했다. 특히 처음 해보는 작업이었기 때문에 DB Schema를 작성하는 것을 우선으로 작업하는 것으로 계획을 세웠으며, 학습이 완료된 딥러닝 모델들 및 원하는 기능들을 모두 API로 구현하는 것을 다음 목표로 잡았다.

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

먼저 적절한 기능을 구현하기 위한 계획을 세우기 위해서 이전에 대학교에서 배웠던 지식을 토대로 DB Schema를 작성했다. 또한 Udemy의 강의를 통해서 Django REST Framework에 대한 강의를 수강했고, 해당 강의를 토대로 기본 틀을 만들었다. 기능이 다 구현되었을 때, 모델 기능을 제외하고는 다른 기능들을 모두 API로 구현한 뒤 API 명세서를 작성했다. 그 뒤에 학습이 완료된 딥러닝 모델도 서버에 업로드하여 API로 구현했으며, 마지막으로 Front-End와 API의 기능들을 연결하는 작업을 진행했다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

처음 해보는 Back-End 분야의 기능 구현에서 발생하는 에러를 수정하는데 굉장히 많은 시간을 사용했다는 점이 가장 큰 문제였다. 기능 구현의 시간보다 에러 수정에 굉장히 많은 시간이 들었고, 이 부분이 Back-End 기능 구현 이외에 다른 작업을 진행하지 못했던 부분이다. 이 부분이 프로젝트를 진행하면서 굉장히 아쉬웠던 점이다.

한계/교훈을 바탕으로 다음 프로젝트에서 시도해볼 것은 무엇인가?

이번 기회를 통해서 Back-End에 대한 지식이 많이 상승했다. 특히 내가 구현하고자 하는 기능이 서버에서는 어떻게 동작하고 있는지에 대해서 자세하게 볼 수 있었던 시간이었다. 또한 GPU가 있는 서버를 유지하는데 얼마나 비용이 드는지 모델을 가볍게 만드는 것이 왜 좋은 것인지에 대해서 알 수 있었던 시간이었다. 그래서 이번 경험을 토대로 다음번에는 단순히 기능만을 고려하여 모델을 개선하는 것이 아닌 기능 외의 부분에 대해서도 고민을 해서 구현을 해보는 작업을 진행해보고 싶다.

3. 이상민

- 대회 들어가기 전, 프로젝트에서의 목표

- Product Serving 에 필요한 Back/Front 경험해보기 (X)
- 실제 눈으로 확인할 수 있는 output을 내기 (O)
- 책임감 있게 한 부분을 맡아 작업을 완수하기 (O)

- 프로젝트에서 시도한 것

저번 프로젝트부터 학습 시, **prompt**를 건드려보고 싶었다.

그래서 이번 KoBART fine-tuning 시, [USER], [SYSTEM] 스페셜 토큰 만들고, 대화 데이터에서 사용자의 말 앞에는 [USER]를 두고, 챗봇의 말 앞에는 [SYSTEM]을 두었고, '###[USER]의 하루를 요약해줘:'를 넣어주어 input data의 형식을 일정하게 만들어주었다.

말하는 주체가 누구인지 파악을 하고, '사용자'의 하루를 요약해주는 fine-tuning에 긍정적인 영향을 끼친 것 같다.

- 느낀 점

- 첫 product serving project를 해봐서 점수가 아닌 챗봇, 생성된 일기들을 output으로 확인할 수 있었던 점이 신기했다. 이 과정에서 Back/Front ends를 경험해보고 싶었는데 그럴지 못한 점이 아쉬웠다.
- 공통 프로젝트의 주제를 정하며 세부적인 부분을 맞추는 것과 내가 맡은 부분에서 어떤 문제를 정의하고 어떻게 문제를 해결할지에 시간 소요가 많이 되었다.
- 평가 메트릭에 대한 중요성을 느꼈다. 어떤 project를 하던, 잘함과 못함의 지표를 나타낼 수 있는 평가 메트릭이 매우 중요하다고 느꼈다. (사람의 주관보다 객관적인 지표를 찾아 설정하는 것이 중요하다고 느낌)
- 요약이라는 것은 특정한 정답이 없다고 생각했다. 그래서 요약 시, 정답 데이터를 만들어 학습시키고 싶지 않았다. Unsupervised Learning으로 요약을 시켜주고 싶었지만, 실패했고 시간 소요를 많이 했다. 그래서 구현에서 다양한 시도들과 디테일을 챙기지 못했던 것이 아쉽다.
- Unsupervised Learning과 Labeling에 대해 고민하면서, 모델들도 중요하지만 데이터와 학습 시 어떤 것을 Loss로 볼 지와 학습 방향의 중요성을 다시 한 번 느꼈다.
- 마지막 KoBART fine-tuning 시, hyperparameter의 중요성을 느꼈다. 처음 7e-6의 learning rate으로 학습을 하고 inference를 해봤을 땐, 사람이 느끼기에 이상한 output을 냈다. 그 후 Pretrained된 모델과 지금의 task는 꽤 다르다고 느껴 학습률을 높여(1e-4) 좋은 결과를 얻을 수 있었다.

4. 이승우

프로젝트에서의 목표

제대로 된 **Product serving**을 경험해 본 인원이 나 뿐이었기에, **Product Manager**를 맡았다. 완성된 서비스 한 개를 만들어내기에는 주어진 시간이 부족했기 때문에, **PM**으로서 그간의 경험을 살린 자신 나름대로의 프로젝트 진행 방식을 적용해 최대한 체계적으로 프로젝트를 진행시켜 자원의 큰 낭비 없이 무사히 완성하는 것을 목표로 삼았다.

또 프로젝트 주제의 경우, 모델 자체가 아닌 서비스를 개발해야하다 보니, 마지막 결과물이 조금은 불완전하더라도 **motivation** 및 **target user**층이 확실하고 기능 추가 등 발전 가능성이 큰 주제를 선정하고자 했다. **AI** 서비스 특성 상 기능 추가가 아니더라도 시간이 지남에 따라 모델의 개선 또한 필요하기에 더욱 알맞은 개발 방향이었던 것 같다.

프로젝트에서의 역할

먼저 **PM**으로서 각자 맡은 파트의 목표 기간 설정, **Project**의 **Architecture** 수립 및 팀원들이 처음 다루어보는 **github issue**나 **pull request** 등의 각종 컨벤션을 수립하고 사용 방법을 알리는 등 팀원들이 외부적인 요소에 신경쓰지 않고 맡은 바에 최대한 집중할 수 있도록 편리한 환경을 만들고자 노력했다.

또, 프로젝트 전체 진행 과정을 자세하게 알고자 했고 따라서 팀원들과 자주 소통하고자 노력했다.

결과적으로 프로젝트 전체를 정확하게 알고 있다 보니 발표 자료 작성 및 발표도 맡았다.

또한 다수의 **Front-end** 개발 경험을 살려 **Front-end** 역시 맡았다. 그에 따라 **App** 디자인 역시 맡았는데, 실제 서비스를 가정하면 젊은 층을 타겟으로 한 감성적인 주제이므로, **Front-end** 및 **UX/UI** 디자인은 몹시 중요한 요소라고 생각했다. 따라서 유저가 만족할 수 있도록 유저 입장에서 생각하며 디테일한 부분까지 신경쓰고자 했다.

느낀점

갈수록 맡은 작업량이 많아진 데다 인원 및 시간이 부족했던 탓에, 언젠가 모델 개발에 참여할 수 있도록 모델 구조 및 작동 방식을 이해해두고 있었음에도 불구하고 모델 개발에 직접 참여할 시간이 없어 내 역량을 보이지 못한 점이 가장 아쉽다. 하지만 초기의 목표는 달성했고, 프로젝트를 리드하고 많은 작업을 도맡아 하여 완성에 큰 기여를 한 것 같아 아주 뿌듯하다. 앞으로 **AI** 서비스 개발 프로젝트를 어떻게 진행, 또 리드해야 하는지 깨닫게 된 값진 경험이었던 것 같다.

5. 이예원

이번 프로젝트에서 나의 목표는 무엇이었는가?

우선 처음에 워낙 구체화된게 없는 상태에서 시작했기 때문에 이 프로젝트가 무사히 마무리되는 것이 우선적인 목표였다. 오히려 처음에 목표로 잡은 것이 없었는데 하면서 점점 목표가 생겨난 것 같다. 멘토님과 멘토링을 진행하면서, 내가 하고 있는 것에 대한 결과를 정리해서 보여주는 것이 얼마나 중요한 것인지 실감했고 내용 정리를 하고 **To Do List**를 그때그때 생각하면서 작업하는 것을 목표로 잡았다.

나는 내 학습목표를 달성하기 위해 무엇을 어떻게 했는가?

팀 노선에 **Data, Train, Metric, Test** 목차를 나누어 매번 누군가에게 보여줘야한다는 마음가짐으로 정리하며 작업했고 모든 실험을 정리하지는 못했지만 실험을 어느 정도 정리해놓은 덕분에 나중에 정리하기 편했다. 멘토님이 주신 피드백과 이번주 안에 끝내야할 현실적인 작업들을 고려해 **To Do List**를 미리미리 생각해놓으며 작업했기에 처음에 추상적이고 막막했던 작업이 제대로 진행될 수 있었던 것 같다.

나는 어떤 방식으로 모델을 개선했는가? 그리고 어떠한 깨달음을 얻었는가?

사실 특별한 방법으로 모델을 개선하지 못한 것 같다. 가장 크게 개선한 부분은 데이터를 생성하여 우리의 목적에 맞는 성능에 더 가까워졌다는 점과 한국어 LLM인 KULLM을 이용해 문맥 파악 능력을 향상시켰다는 점이다. 이 과정에서 LLM이 확실히 좋긴 하다는 것을 느꼈고, 무엇보다 데이터의 중요성을 느꼈다. GPT로 생성한 데이터로 학습했을 때가 생성한 데이터 + 감성 대화 데이터로 학습했을 때보다 더 자연스러운 응답을 생성하였는데 이를 통해 데이터의 양보다 퀄리티가 더 중요하다는 것을 느꼈다.

마주한 한계는 무엇이며, 아쉬웠던 점은 무엇인가?

데이터 생성할 때, 이게 될까? 라는 마음으로 너무 대충 생성하였었다. 결과 성능이 좋아 해당 데이터를 사용하긴 하였으나 나중에 다시 생성 flow를 구성하려고 했을 때는 너무 낮은 상태였다. 다양한 케이스에 대해 학습할 수 있도록 flow를 구성하여 생성했으면 예외 케이스에도 답변할 수 있는 좋은 Fine가 되었을텐데 이 부분이 챗봇 성능의 너무 큰 단점을 만들어 버려서 상당히 아쉬웠다.

한계/교훈을 바탕으로 다음 프로젝트에서 스스로 새롭게 시도해볼 것은 무엇일까?

다음 프로젝트, 부캠이 끝나고 또 새로운 프로젝트를 해보게 된다면 구체적인 계획과 목표를 설정할 것이 당연히 최우선으로 지켜야 할 목표이다. 또한 이번 프로젝트에서 발표자료를 이렇게 많은 공을 들여 구성해본 것은 처음이었는데 그 과정에서 내가 한 작업이 얼마나 차별성이 없는지에 대해 뼈저리게 느껴 다음 프로젝트부터는 어떻게 하면 이 프로젝트가 더 돋보이게 할 수 있는가에 대해 더 고민하고 차별성이 있는 결과물을 만들고 싶다.