안녕하십니까? ‘2024 국립국어원 인공지능의 한국어 능력 평가 경진 대회'에 참가해 주셔서 진심으로 감사드리며, 아래와 같이 모델 기술서 제출에 대하여 안내해 드립니다.

1. 모델 기술서 제출 안내

- **모델 기술서 제출 기한: 2024년 8월 30일까지**
  - ‘모델 기술서’란 모델의 설계 과정, 모델의 구조 및 설계의 이유, 개발 상세 내용, 모델 재현을 위한 사용설명서 등을 담고 있는 문서입니다(붙임1 참조).
- **모델 기술서 제출 방법: 전자 우편(ybjeong@teddysum.ai, hansongi@korea.kr)으로 제출**
- **모델 기술서 작성 시 고려 사항**
  - **모델 기술서 양식을 준수하여 작성합니다(붙임 1. 모델 기술서 양식).**
    - 모델 기술서의 항목별 내용은 자유롭게 작성(작성 예시는 참고용으로 활용)
  - **작성 분량은 10쪽 이내를 권장합니다(필수 아님).**
    - 작성 분량은 심사에 영향을 미치지 않습니다. .
  - **모델을 내려받을 수 있는 링크가 포함되어야 합니다(깃허브 등).**
    - 깃허브(github) 소스코드의 경우, 해당 코드는 추가 환경 설정 없이 작동되어야 합니다. (예를 들어 !pip install 등 라이브러리 설치 코드도 모두 포함하여야 함)
    - 수상작이 되면, 소스코드 및 모델을 깃허브를 통해 공개하여야 합니다.

2. 수상자 선정 기준 안내

- **본상(대상, 금상, 은상) 선정 기준**
  - **순위표(리더보드)의 점수와 정성 평가 점수, 전문가 평가 점수(일상 대화 요약에 한함), 발표 평가 점수를 합산하여 본상(대상, 금상, 은상) 수상자를 결정합니다.**
  - **모델에 대한 정성평가는 모델 기술서 및 제출한 코드를 바탕으로 진행됩니다.**
  - **‘일상 대화 요약’은 출력 결과물에 대한 글쓰기 채점 전문가의 평가 점수를 합산합니다.**
    - 출력 결과물 정성 평가에 대한 평가 기준은 붙임 3. 참조
  - **정량, 정성평가의 점수를 합산하여 과제별 3팀에 대해 발표 평가를 진행합니다.**
    - 과제당 3팀 총 12팀이 발표평가를 진행합니다.
  - **발표 평가 대상인 12개 팀 중 대상(문화체육관광부 장관상) 1팀, 금상 2팀, 은상 2팀을 수상자로 선정합니다.**

붙임 1. 모델 기술서 양식
붙임 2. 모델 정성 평가 기준
붙임 3. 일상 대화 요약 전문가 정성 평가 기준

붙임 1. 모델 기술서 양식

모델 기술서	
참가팀	bd
모델명	z
작성 일자	2024.09.01
모델 깃허브 링크	https://github.com/DonghaeSuh/korean_dialouge_summarization

1. 라이브러리 및 데이터

1.1. 주요 스펙 및 라이브러리

- **Python 3.10.12** 및 Jupyter 사용
- **transformers** 라이브러리 사용 (pip install transformers)
- pre-trained language model 로 **MLP-KTLim/llama-3-Korean-Bllossom-8B** 사용
  목적 : 한국어 10B 이하 모델 중 SOTA 점수를 받았고, 시도한 다른 모델의 baseline과 비교했을 때, test 데이터셋에 대해 가장 우수한 성능을 보임

- **rouge_score**, **bert_socre**, **google-research/bleurt** 라이브러리 사용
  목적 : validation 데이터셋에 대한 평가를 진행할 때 test 데이터셋에 대한 평가 지표와 일치시키기 위함.
- **bitsandbytes**, **trl, peft** 라이브러리 사용
  목적 : QLoRA 방식으로로 LLM을 fine-tuning하기 위함.
- **wandb** 라이브러리 사용
  목적 : 학습 로그 관리를 위함.

1.2. 사용한 데이터셋

- 대회에서 주어진 일상 대화 데이터셋만 사용함.

2. 데이터 전처리

데이터 전처리는 크게 두 가지 방식으로 진행했습니다.

2.1 의미 없이 끼어있는 단어 제거

의미 없이 끼어있는 단어의 유형은 다음과 같습니다.

- **물결이 붙는 추임새** : 음~, 그~
- **대화 중 습관적으로 사용되는 추임새** : 어, 뭐, 음, 예, 이제
- **노이즈에 해당하는 단어** : 좋, 크, 스, 하
- **x 를 포함한 단어** : xx, 부x, xx게새

위와 같은 단어들은 정규표현식을 이용하여 제거해주었습니다.

2.2 반복되는 어구(단어) 한 개로 통일

대화 데이터를 분석한 결과, 화자의 발화(utterance) 내에 반복되는 단어 및 어구가 빈번하게 등장하는 것을 확인했습니다.

예를 들어,

- **반복되는 단어**:
  - "또 체코를 하니까 생각이 났 났던 게"
  - "김치찌개 제육볶음 된장찌개 된장찌개도 정식에도"

- **반복되는 어구**:
  - "나의 그 모습을 체크한다거나 말투 같은 거 이런 거를 말투 같은 거 이런 거를 체크하는 편이에요."
  - "대자연에 또한 가을에서 주는 또한 가을에서 주는 그 색채 또 가을이 주는 그 느낌"

이와 같은 패턴의 반복 어구(단어)는 Train 데이터셋에서 827개, Dev 데이터셋에서 195개, Test 데이터셋에서 641개 발견되었습니다.

이러한 반복 어구(단어)가 많이 등장하는 이유는, 일상 대화 데이터가 두 명의 화자가 나눈 음성 대화를 음성 인식 텍스트 변환 프로그램을 이용해 자동으로 전사하는 과정에서 비롯된 것으로 보입니다.

그러나 모든 반복 어구(단어)가 노이즈로 간주될 수는 없다는 점을 발견했습니다.

예를 들어,

- **"가 가"**는
  - Train 데이터셋 내에서 62번 등장한 반복 단어이고
  - "A 가 가지고 있다"에서처럼 **조사 "가"**로 사용된 경우가 많이 존재했습니다.

- **"다 다"**는
  - Train 데이터셋 내에서 29번 등장한 반복 단어이고
  - "모두 다 다르다"에서처럼 전부(All)를 뜻하는 **부사 "다"**로 사용되었습니다.

따라서, 데이터셋 내 여러번 등장하는 반복 어구(단어)의 경우 특별한 의미를 지닐 수 있기 때문에

- 데이터셋 내에 10번 이상 등장하는 반복 어구(단어)에 대해 정성적인 평가를 진행하였고,
- **"가"**와 **"다"**의 경우를 제외한 나머지 반복 어구(단어)를 모두 1개로 통일하는 전처리를 적용했습니다.

고

흥미롭게도, 1차 전처리로 인해 생기는 새로운 반복 어구가 발견되었습니다.

예를 들어,

- **"하고 있 하 하고 있고요"**
  - 1차 전처리 결과: "하고 있 하고 있고요"

- **"가을 중에서 어떤 계 계절 중에서 어떤 계절을"**
  - 1차 전처리 결과: "가을 중에서 어떤 계절 중에서 어떤 계절을"

이와 같은 패턴의 반복 어구는 Train 데이터셋에서 17개, Dev 데이터셋에서 1개, Test 데이터셋에서 12개 발견되었습니다. 때문에, 1차 전처리에서 생긴 새로운 반복 어구 중 "가 가"와 "다 다"를 제외한 2차 전처리를 수행하여 반복 어구를 모두 제거했습니다.

3. 데이터 분석

이번 대회에서 제공된 데이터셋은 Train 506개, Dev 102개, Test 408개로, 상대적으로 적은 양이었습니다. 따라서 데이터를 보다 효율적으로 활용하여 학습하는 것이 핵심 과제가 되었습니다.

이를 해결하기 위해, 저는 **구조화된 요약을 생성**할 수 있도록 **프롬프트**를 구성하고, **Instruction fine-tuning**을 적용했습니다.

이 방식은 모델이 단순히 하나의 긴 문장을 생성하는 것이 아니라, **## 전반적인 요약**, **## Speaker 1 요약**, **## Speaker 2** 요약과 같은 정해진 **Header 토큰** 뒤에 올 토큰들을 예측하게 함으로써, 더 간단하고 일관된 작업을 수행하도록 유도하는 것입니다.

그런데, 학습된 모델을 Greedy Decoding으로 inference할 때, 생성되는 문장이 길어질수록 문장의 품질이 저하되는 현상이 관찰되었습니다. 이는 ***exposure bias***(Ranzato et al., 2016;Bengio et al., 2015)에서 기인한 문제로 볼 수 있습니다.

exposure bias란 학습 시에는 정답인 이전 토큰에 기반해 다음 토큰을 예측하지만, inference 시에는 생성된 토큰에 기반해 예측을 수행하는 과정에서 발생합니다. 잘못된 토큰이 생성되면, 그 이후의 토큰 예측에도 악영향을 미쳐 결과적으로 문장 품질이 저하되고, 학습 시의 문장 확률 분포에서 점차 벗어나는 현상을 뜻합니다.

이를 보완하기 위해, 서로 다른 형태의 데이터셋(Train 데이터셋만 사용한 모델과 Train+Dev 데이터셋 전체를 사용한 모델)으로 학습된 두 모델의 Inference 결과물을 비교하여, **더 짧은 문장을 선택하는 앙상블** 방식을 적용했을 때, 성능이 크게 향상되었습니다.

그러나 여전히 뒷부분의 문장(Speaker 2 요약)의 성능은 미흡했습니다.

이 문제를 해결하기 위해 Task를 분리하여 학습을 진행했습니다. 특히, 문제가 되었던 뒷 문장(Speaker 2 요약)을 더 정확하게 예측하기 위해, **"전반적인 요약"과 "Speaker 2 요약"을 예측하는 별도의 LLM을 학습**시켰습니다.

이후, 이 모델의 Inference 결과와 기존 앙상블 결과의 "Speaker 2 요약"을 비교하여 더 짧은 요약을 선택하는 2차 앙상블을 수행했으며, 이로 인해 소폭의 성능 향상을 이끌어낼 수 있었습니다.

3.1. Reference 요약문 구조 분석

우선, 입력 대화에 대한 Reference(정답) 요약문을 정성적으로 분석했습니다.
그 결과, 요약문이 다음과 같은 구조로 이루어져 있음을 통계적으로 확인했습니다

- 한 문장의 **전반적인 요약** (예외적으로 4개의 샘플은 두 문장으로 구성됨)
- 한 문장 이상의 **화자 1 요약**
- 한 문장 이상의 **화자 2 요약**

3.2. 전반적인 요약 구조 분석 및 형식 통일

전반적인 요약은 다음 6가지 타입으로 구성되어 있음을 확인했습니다

- **"두 화자는 이 대화에서 ~"**
- **"두 화자는 ~"**
- **"화자들은 ~"**
- **"두 사람은 ~"**
- **"이 대화에서는 ~"**
- **"SD\d{7}(?:와|과).*SD\d{7}(?:은|는) ~"** (예: "대화에서 화자 SD2001966와 SD2001967은 ~")

이 중 가장 빈도가 높은 **"두 화자는 이 대화에서"** 타입을 기준으로 모든 전반적인 요약 문장을 통일했습니다.

"두 화자는 이 대화에서" 타입 통계는 다음과 같습니다.
- Train의 경우 506개의 데이터셋 중에서 157개
- Dev의 경우 102개의 데이터셋 중에서 60개

이를 통해 모델이 예측해야 할 시작 토큰을 일관되게 설정하여, 모델이 실질적으로 중요한 정보인 전반적인 요약 내용을 생성하는 데 집중할 수 있도록 했습니다.

3.3. 화자 1, 화자 2 요약 구조 분석 및 형식 통일

전반적인 요약과 마찬가지로, **화자 1**과 **화자 2** 요약의 형식도 통일했습니다.

화자 1과 **화자 2**를 구분하기 위해, 통계적으로 확인한 결과
대부분의 경우 **담화의 서두를 여는 화자**가 **화자 1에 해당**한다는 점을 발견했습니다.

담화의 서두를 여는 화자가 화자 1에 해당하는 경우의 통계는 다음과 같습니다.
- Train의 경우 506개의 데이터셋 중에서 460개
- Dev의 경우 102개의 데이터셋 중에서 95개

이를 기반으로, **담화의 첫 번째 발화자**를 **화자 1**로, **나머지 발화자**를 **화자 2**로 정의했습니다.

3.4. 형식 통일이 평가 지표 성능에 끼치는 영향 분석

추가적으로, 사용된 평가 지표인 ROUGE-1의 경우, 생성된 출력에서 토큰의 순서는 평가 점수에 영향을 미치지 않으며, 정답 요약문과 비교하여 얼마나 정확한 토큰이 등장하는지 여부(Precision, Recall)만 고려된다는 점이 중요합니다.

이는 출력 요약문에서 어느 화자의 요약이 먼저 등장하는지의 순서가 ROUGE-1 점수에 영향을 미치지 않으므로, 요약문 구조를 유연하게 구성할 수 있음을 의미합니다.

동일한 원리가 전반적인 요약에도 적용됩니다.
"이 대화에서 두 화자는..." 이외의 다른 타입을 분석해보면,
사용된 토큰들은 주로 **"화자 이름(SD\d{7})"**과 **"사람"**입니다.

하지만,
"화자 이름(SD\d{7})" 토큰은 이후 화자 요약에서 등장할 것이므로 전반적인 요약에서 굳이 언급할 필요가 없으며
"사람" 토큰은 등장 빈도가 적기 때문에(Train: 13개, Dev: 4개) 무시했습니다.

3.5. 통일된 형식에 맞춘 구조화된 출력

이러한 구조화된 출력을 유도하기 위해

- **Markdown** 형식의 **강조 태그** "**"와 **Header 토큰** "##"를 추가한 프롬프트를 구성했습니다.
- (Xu et al., 2023)에 따르면, 앞서 주어진 프롬프트를 **"다시 읽도록(Re-Reading)"** 지시하는 것이 LLM의 추론 능력을 향상시킬 수 있다고 합니다. 이를 반영하여 키워드와 대화 내용 이후에 **"위 대화 내용을 다시 한번 잘 읽어주세요."**라는 프롬프트를 추가했습니다.

사용한 프롬프트는 아래와 같습니다.

```
[System Prompt]
당신은 유능한 AI 어시스턴트(assistant) 입니다. **대화 내용**과 **대화 키워드**를 보고, **대화 키워드**와 연관된 한국어 대화 요약문을 생성해주세요.

[User Prompt]
**대화 키워드** : {키워드1}, {키워드2}, . . . 에 대한 대화 내용입니다.
**대화 내용** :
{화자 1} : {발화 1}
 . . .
 . . .

위 대화 내용을 다시 한번 잘 읽어주세요.
이제 ## 전반적인 요약, ## {화자 1} 요약, ## {화자 2} 요약 구조의 한국어 대화 요약문을 생성해주세요.

[Generation Prompt]
Assistant :
```

이 프롬프트에 따른 Reference output의 형식은 아래와 같습니다.

```
## 전반적인 요약
{전반적인 요약}

## {speaker_1} 요약
{speaker_1_summary}

## {speaker_2} 요약
{speaker_2_summary}
```

이와 같은 구조화된 출력 방식을 통해

- 모델이 **적은 데이터로도 효율적으로 학습**하고
- **보다 빠르게 수렴**하며
- **높은 품질의 요약문을 생성**할 수 있도록 했습니다.

4. 모델 개요

4.1. 적용한 모델에 대한 내용

MLP-KTLim/llama-3-Korean-Bllossom-8B
: MLP-KTLim/llama-3-Korean-Bllossom-8B 모델은 LLaMA 3을 기반으로 한 한국어-영어 bilingual 언어 모델로, 서울과학기술대학교 MLPLab, 테디썸, 연세대학교에서 개발하였습니다.

이 모델은 한국어와 문화에 맞춤화된 개선점을 포함하고 있습니다

- **Knowledge Linking**: 추가적인 사전 학습을 통해 한국어와 영어 지식의 연결을 강화하였습니다.
- **Vocabulary Expansion**: 한국어 Vocabulary를 확장하여 한국어 표현력을 강화하였습니다.
- **Instruction Tuning**: 한국어와 한국 문화에 특화된 사용자 맞춤형 명령어 학습 데이터를 사용하여 fine-tuning 하였습니다.
- **Human Feedback**: Direct Preference Optimization(DPO)를 적용하여, 출력이 인간의 선호도에 더 가깝게 조정되었습니다.
- **Vision-Language Alignment**: 이 언어 모델에 Vision Transformer를 통합하여 시각적 정보와 텍스트 정보를 모두 포함하는 작업에서 multimodal 능력을 강화하였습니다.

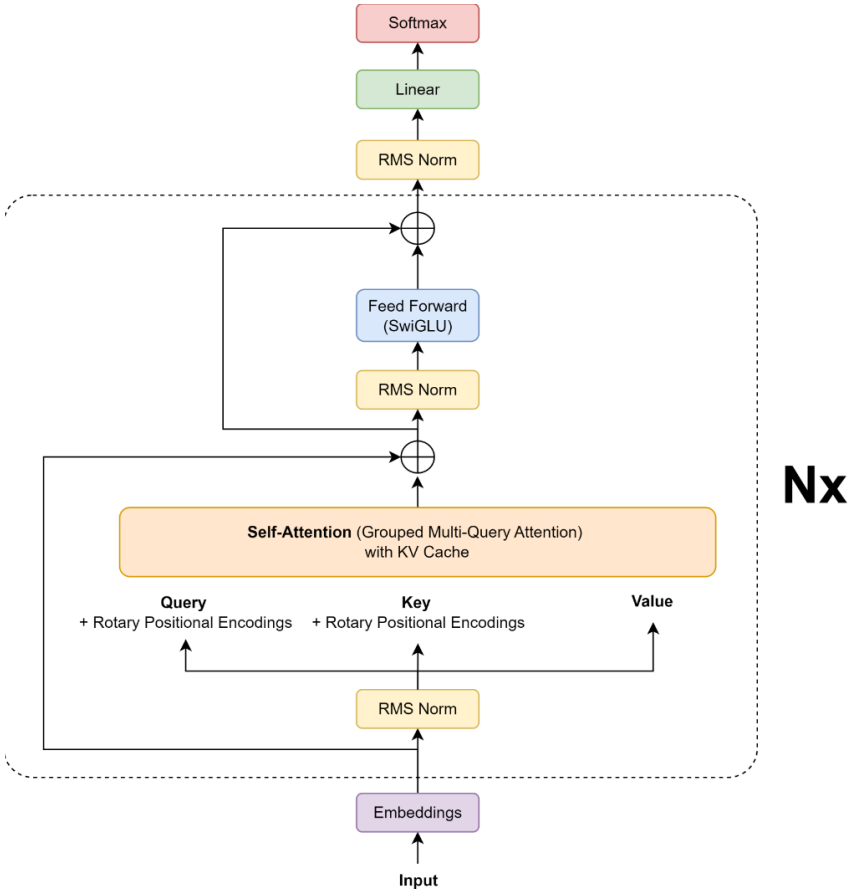
4.2. 해당 모델 선택 이유

본 대회 베이스라인으로 주어진 MLP-KTLim/llama-3-Korean-Bllossom-8B 모델은 Llama 3 기반의 LLM으로 한국어 자연어 생성 Task에서 매우 뛰어난 성능을 보입니다.

본 대회 Task인 일상 대화 요약 Task의 경우 입력을 받아 추상적인 요약문을 생성해내는 작업이므로, 문장 생성에 최적화된 모델 구조(Transformer Decoder)와 사전 학습(Next token Prediction)으로 만들어진 모델인 위 모델을 선택하였습니다.

또한, Baseline 세팅에 대해 다른 여러 모델들과 성능 비교 결과 가장 좋은 성능을 보인 위 모델을 선택하였습니다.

4.3. 자세한 모델 학습과정 설명 및 소스코드



MLP-KTLim/llama-3-Korean-Bllossom-8B 모델의 아키텍처는 (그림 1)과 같습니다.

| 입력부터 출력까지의 벡터의 흐름

1. **Token id sequence**를 입력받으면
2. **Embedding Layer**를 통해 각 Token id에 해당하는 임베딩 벡터로 변환이 되고
[3]. **RMS Norm**을 통한 정규화 이후
[4]. **Dense Layer**를 거쳐 **Query**, **Key**, **Value**를 얻어냅니다.
[5]. 이때, **Query**와 **Key**에 대해서만 **Rotary Positional Encoding**을 적용해 Token sequence에 대한 정보를 각 Token에 주입합니다.
[6]. 이제, 효율적인 Attention 연산을 위한 **Grouped Multi-Query Attention**을 수행한 결과와 RMS Norm 이전의 벡터와 **Residual connection**이 수행됩니다.
[7]. **Attention Layer** 이후 **RMS Norm**이 다시 한 번 수행되고
[8]. Activation function이 **SwiGLU**인 **Feed Forward Network**를 통과합니다.
[9]. Feed Forward Network의 결과와 RMS Norm 이전의 벡터와 Residual connection이 수행되고

10. (3번-9번)까지의 과정이 하나의 **Transformer Decoder Block**이 되어 (N=32)개의 Block을 통과합니다.
11. 마지막 Block을 통과한 결과에 **RMS Norm**이 적용되며
12. Linear Layer을 통해 hidden dimension 으로부터 **vocabulary size dimension**으로의 벡터 차원이 확장됩니다.
13. vocabulary size dimension 크기의 벡터에 **Softmax**가 취해짐으로써 vocabulary 내 모든 Token들에 특정한 확률 값이 부여되고
14. 최종적으로 정답에 해당하는 Token이 가장 큰 값이 되도록 가중치가 업데이트되기 위한 Loss인 **Cross Entropy Loss**가 적용되어 Loss가 계산됩니다.

1. LlamaModel 코드

```
# LlamaModel
# 1. Token id sequence를 입력받으면
# 2. Embedding Layer를 통해 각 Token id에 해당하는 임베딩 벡터로 변환이 되고
inputs_embeds = self.embed_tokens(input_ids)

# LlamaDecoderLayer
# [3]. RMS Norm을 통한 정규화 이후
# [4]. Dense Layer를 거쳐 Query, Key, Value를 얻어냅니다.
# [5]. 이때, Query와 Key에 대해서만 Rotary Positional Encoding을 적용해 Token sequence에 대한 정보를 각 Token에 주입합니다.
# [6]. 이제, 효율적인 Attention 연산을 위한 Grouped Multi-Query Attention을 수행한 결과와 RMS Norm 이전의 벡터와 Residual connection이 수행됩니다.
# [7]. Attention Layer 이후 RMS Norm이 다시 한 번 수행되고
# [8]. Activation function이 SwiGLU인 Feed Forward Network를 통과합니다.
# [9]. Feed Forward Network의 결과와 RMS Norm 이전의 벡터와 Residual connection이 수행되고
residual = hidden_states

hidden_states = self.input_layernorm(hidden_states) # [3]

# Self Attention
hidden_states, self_attn_weights, present_key_value = self.self_attn( # [4], [5]
    hidden_states=hidden_states,
    attention_mask=attention_mask,
    position_ids=position_ids,
    past_key_value=past_key_value,
    output_attentions=output_attentions,
    use_cache=use_cache,
    cache_position=cache_position,
    position_embeddings=position_embeddings,
    **kwargs,)

hidden_states = residual + hidden_states  # [6]

# Fully Connected
residual = hidden_states
hidden_states = self.post_attention_layernorm(hidden_states)  # [7]
```

```
hidden_states = self.mlp(hidden_states)  # [8]
hidden_states = residual + hidden_states  # [9]

outputs = (hidden_states,)

if output_attentions:
    outputs += (self_attn_weights,)

if use_cache:
    outputs += (present_key_value,)

return outputs
```

```
# LlamaModel
# 10. (3번-9번)까지의 과정이 하나의 Transformer Decoder Block이 되어 (N=32)개의 Block을 통과합니다.
# 11. 마지막 Block을 통과한 결과에 RMS Norm이 적용되며
for decoder_layer in self.layers: # 10.
            if output_hidden_states:
                all_hidden_states += (hidden_states,)

            if self.gradient_checkpointing and self.training:
                layer_outputs = self._gradient_checkpointing_func(
                    decoder_layer.__call__,
                    hidden_states,
                    causal_mask,
                    position_ids,
                    past_key_values,
                    output_attentions,
                    use_cache,
                    cache_position,
                    position_embeddings,
                )
            else:
                layer_outputs = decoder_layer(
                    hidden_states,
                    attention_mask=causal_mask,
                    position_ids=position_ids,
                    past_key_value=past_key_values,
                    output_attentions=output_attentions,
                    use_cache=use_cache,
                    cache_position=cache_position,
                    position_embeddings=position_embeddings,
                )

hidden_states = self.norm(hidden_states) # 11
```

2. LlamaForCausalLM 코드

```
# LlamaForCausalLM
# 12. Linear Layer을 통해 hidden dimension 으로부터 vocabulary size dimension으로의 벡터 차원이 확장됩니다.
# 13. vocabulary size dimension 크기의 벡터에 Softmax가 취해짐으로써 vocabulary 내 모든 Token들에 특정한 확률 값이 부여되고
# 14. 최종적으로 정답에 해당하는 Token이 가장 큰 값이 되도록 가중치가 업데이트되기 위한 Loss인 Cross Entropy Loss가 적용되어 Loss가 계산됩니다.

logits = self.lm_head(hidden_states) # 12

logits = logits.float()

loss = None
if labels is not None:
    # Shift so that tokens < n predict n
    shift_logits = logits[..., :-1, :].contiguous() # 다음 토큰 예측을 위한 토큰 shift
    shift_labels = labels[..., 1:].contiguous() # 다음 토큰 예측을 위한 토큰 shift

    # Flatten the tokens
    loss_fct = CrossEntropyLoss()
    shift_logits = shift_logits.view(-1, self.config.vocab_size)
    shift_labels = shift_labels.view(-1)

    # Enable model parallelism
    shift_labels = shift_labels.to(shift_logits.device)
    loss = loss_fct(shift_logits, shift_labels) # 13, 14
```

| QLora

사용 가능한 자원의 한계(Colab Pro Plus, A100 40G 1대) 때문에 메모리 효율적인 fine-tuning에 대한 고민을 많이 했습니다. 이에 QLoRA: Efficient Finetuning of Quantized LLMs 를 통해 제안된 QLoRA 기법을 사용했습니다.

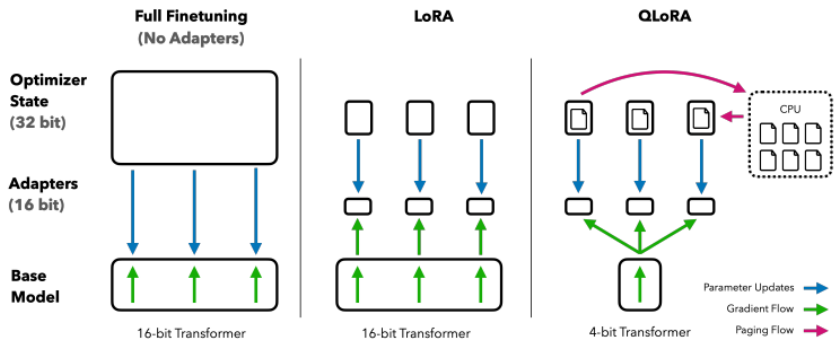


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

QLoRA: Efficient Finetuning of Quantized LLMs

위 기법은 양자화된 PLM(Pretrained Language Model)을 freeze 하고 메모리를 많이 요구하는 Layer에 Adapter를 병렬적으로 붙여 fine-tuning하는 기법입니다. 즉, LoRA에 양자화를 적용한 버전이라고 볼 수 있습니다. 저는 QLoRA를 쉽게 적용하기 위한 라이브러리인 transformers, BitsAndBytes, peft 라이브러리를 이용해 구현하였습니다.

3. 양자화 config 설정

```
bnb_config = BitsAndBytesConfig(
        load_in_4bit=True, # 모델을 4bit 양자화
        bnb_4bit_use_double_quant=True, # 추가 메모리 절약을 위해 양자화 상수 양자화
        bnb_4bit_quant_type="nf4", # 4-bit NormalFloat 타입으로 PLM 양자화
        bnb_4bit_compute_dtype=torch.bfloat16 # 모델 연산 시에는 bfloat16 타입으로 계산)
```

4. 모델 불러오기

```
# CausalLM (=Head가 vocab_size 차원의 벡터로 사영하는 Linear Layer)
model = AutoModelForCausalLM.from_pretrained(
         args.model_id, # Huggingface 모델 이름
         quantization_config=bnb_config, # 앞서 정의한 양자화 config
         device_map="auto", # 사용하는 GPU 자동 배정
         revision = 'ece1eea3e0f1653eebac5016bdae1fbd37894078') # 가장 성능이 좋았던 버전 commit id
```

5. gradient checkpointing 및 LoRA 적용

```
model.gradient_checkpointing_enable() # 메모리 절약을 위한 gradient checkpointing 활성화
model = prepare_model_for_kbit_training(model) # 모델을 k-bit 학습에 맞게 변환

# LoRA Config 설정
lora_config = LoraConfig(
    r=config["lora_arch"]["r"], # low rank 사이즈
    lora_alpha=config["lora_arch"]["lora_alpha"], # adpater의 영향력을 조절하기 위한 alpha
    target_modules= ["q_proj", "k_proj", "v_proj", "o_proj"], # LoRA가 적용될 모델의 Layer
    lora_dropout=config["lora_arch"]["lora_dropout"], # LoRA에서 사용될 dropout rate
    bias="none", # bias는 사용하지 않음
    task_type="CAUSAL_LM", # Task : CAUSAL_LM(다음 토큰 맞추기)
)

# LoRA 모델 적용(adpapter 적용)
model = get_peft_model(model, lora_config)
```

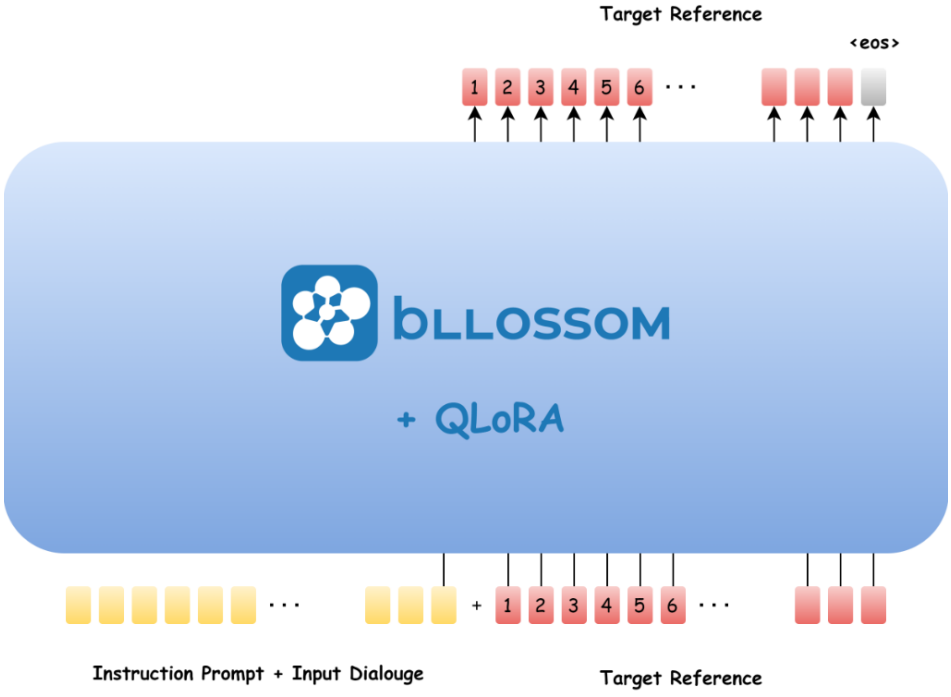
| Hyperparameter 세팅 및 최고 성능 모델 선정 기준

hyperparameter	value
batch size	1
gradient_accumulation_steps	64
warmup_steps	20
learning_rate	2e-5
epoch	25
weight_decay	0.1
lr_scheduler_type	cosine
max_seq_length	2048
r	16
lora_alpha	64
lora_dropout	0.1

최고 모델 선정 기준으로는, validation dataset의 eval_loss에 대한 early stopping(patience=3)을 적용하여 선정하였습니다.

재현 가능한 **최종 Inference**를 뽑아내기 위해서 **Greedy Decoding**으로 결과를 뽑아냈고, **repetition_penalty=1.1**을 적용해 반복되는 문장을 억제했습니다.

| 학습 방식



QLoRA 방식으로 양자화 후 LoRA Adapter를 추가한 이후에
앞서 구성한 **Instruction Prompt**와 **Input Dialouge**가 **입력**으로 들어가고,
그에 따라 **Target Reference**에 해당하는 **구조화된 요약문**의 Token들을
auto-regressive하게 예측해내도록 **fine-tuning**합니다.

5. 평가 결과

총 8개의 모델에 대한 리더보드 상 점수입니다.
- **baseline** : 대회 baseline
- **baseline (with SFT)** : 대회 baseline + SFT fine-tuning
- **TG** : 전반적인 요약(**T**otal Summary) 부분만 형식 통일(**G**eneralization) + Instruction Fine-tuning
- **TG + SG** : TG + 화자1, 2 요약(**S**peaker Summary) 형식 통일(**G**eneralization)
- **TG + SG + RR** : TG + SG + 반복 어구(단어) 제거 전처리(**R**emove **R**epeated phrase and words)
- **TG + SG + RR + Total** : TG + SG + RR 세팅으로 Train+Dev 데이터 전부(**Total**) 사용하여 학습
- **Ensemble_v1** : TG + SG + RR와 TG + SG + RR + Total 버전 중에서 짧은 문장을 선택
- **Ensemble_v2** : Ensemble_v1에서 화자 2 요약 부분만 생성하도록 따로 학습된 모델의 결과로 대체

Model	Evaluation Score	ROUGE-1	BERTScore	BLEURT
baseline	54.276	44.592	73.277	44.950
baseline (with SFT)	58.982	**54.759**	**79.154**	43.035
TG	59.148181	53.729551	78.3912502	45.3237418
TG + SG	59.6248691	53.4625498	78.5050607	46.9069969
TG + SG + RR	59.8890253	53.8221605	78.5433122	47.3016033
TG + SG + RR + Total	59.8531238	54.1609176	78.535594	46.8628598
Ensemble_v1	60.0270675	53.7108599	78.5558022	**47.8145403**
Ensemble_v2	**60.0316536**	53.7539715	78.5728804	47.7681089

5.1. 모델 자체 평가

저의 모델은 기존의 ROGUE와 BERTScore의 단점을 보완하고 장점을 결합한 BLEURT 점수에서 특히 주목할 만한 성과를 거두었습니다.

비록 ROUGE-1과 BERTScore에서 baseline (with SFT) 대비 약간의 점수 하락이 있었지만, BLEURT 점수에서는 크게 개선된 결과를 보여주었습니다. 이를 통해, 모델이 실제로 매우 명확하고 간결한 요약문을 생성할 수 있음을 확인할 수 있었습니다.

또한, **Inference** 시 **Greedy Decoding** 기법을 사용함으로써, beam search와 같은 추가적인 메모리 오버헤드 없이 뛰어난 재현 능력을 발휘할 수 있었습니다. 이 모든 요소들이 결합되어, 저의 모델은 높은 효율성과 품질을 동시에 실현하고 있습니다.

5.2. 경진대회 이후 모델 보완 및 추가 연구에 대한 방향성

제안된 모델은 명확하고 간결한 요약을 제공하는 데 강점이 있지만, ROGUE와 BERTScore에서 낮은 점수를 받은 것은 Reference(정답) 요약문에 포함되어야 할 중요한 토큰들이 충분히 생성되지 않았음을 시사합니다.

| Decoding 하이퍼파라미터 최적화
이는 Greedy Decoding의 한계에서 비롯된 것으로 보이며, 향후 Decoding 하이퍼파라미터 최적화를 통해 더욱 풍부하면서도 정확한 토큰을 담은 요약문을 생성할 수 있을 것입니다.

| Loss 변형
(Liu et al., 2022)에 따르면, 추상 요약(Abstractive Summarization)에서 문장 생성을 위한 fine-tuning 시 MLE(Maximum Likelihood Estimation)로 학습이 수행되는데, 이는 inference 시 성능 하락으로 이어질 수 있다고 합니다. 이를 보완하기 위해 token-level이 아닌 sentence-level에서 요약문의 성능을 평가하고, 이를 loss에 반영하여 학습하면, inference 시 뽑히는 더 높은 확률의 문장이 실제 평가지표에서도 좋은 점수를 얻을 수 있도록 할 수 있습니다.

| 데이터 증강
또한, 데이터 증강이 허용되는 (나)형의 특성을 고려하여, 다음과 같은 방법으로 모델을 개선할 수 있습니다

- 적은 데이터셋의 한계를 극복하기 위해 **외부 데이터셋을 추가 학습**에 활용하거나
- 데이터셋에 대한 정성 분석을 통해 중요한 문장이나 단어에 **태그**를 달아, 모델이 핵심에 더 집중할 수 있도록 유도할 수 있습니다.

이번 경진대회에서 새롭게 제안한 두 가지 기법은 다음과 같습니다.

- **음성 전사 과정에서 생기는 반복 어구(단어) 노이즈를 효과적으로 전처리하는 기법**
- **출력(output) 형식을 통일하여 데이터 효율적인 LLM fine-tuning 기법**

이 기법들은 일상 대화 요약뿐만 아니라 다양한 분야에도 적용 가능하며
음성 전처리 분야나, 기업에 최적화된 LLM(대규모 언어 모델)을 제작하는 데 유용하게 활용될 것으로 기대됩니다.

6. 모델 사용설명서

1. 레포지토리를 clone합니다.

```
git clone https://github.com/DonghaeSuh/korean_dialouge_summarization.git
```

2. conda 또는 venv를 통한 가상환경을 만듭니다

```
# conda로 가상환경 생성 (Python 3.10.12)
conda create -n korean_dialogue_summarization python=3.10.12 -y
# 가상환경 활성화
source activate korean_dialogue_summarization


# venv로 가상환경 생성 (Python 3.10.12이 이미 시스템에 설치되어 있어야 함)
python3 -m venv korean_dialogue_summarization
# 가상환경 활성화
source korean_dialogue_summarization/bin/activate
# 가상환경에서 pip 업그레이드
pip install --upgrade pip
```

3. 의존성 라이브러리를 설치합니다.

```
# requirements.txt에 정의된 라이브러리 설치
pip install -r requirements.txt
```

빠른 세팅

```
source setup.sh
```

레포지토리를 clone한 이후, 위 스크립트를 실행하면

- korean_dialogue_summarization라는 이름의 **가상환경을 생성하고 활성화**한 이후
- **필요 의존성 패키지들이 설치**되고
- **checkpoints라는 폴더에 3가지 chekcpoint 폴더**가 생깁니다
  - checkpoint-85 : "전반적인 요약" + "speaker 2 요약" 만 따로 학습한 모델입니다
  - checkpoint-115 : TG + SG + RR 버전 모델입니다
  - checkpoint-150 : TG + SG + RR + Total 버전 모델입니다

4. Inference

빠른 추론

TG + SG + RR 버전과, **TG + SG + RR + Total** 버전에 해당하는 checkpoint를 **setup.sh**를 통해 불러왔다면

```
source run_fast_inference.sh
```

위 스크립트를 통해 한번에 아래의 총 5가지 결과물을 results 폴더에서 확인하실 수 있습니다.

- **ensemble_1.json**
- **TG_SG_RR_result_postprocessed.json**
- TG_SG_RR_result.json
- **TG_SG_RR_Total_result_postprocessed.json**
- TG_SG_RR_Total_result.json

4.1 추론

추론하고자 하는 checkpoint의 경로를 adapter_checkpoint_path에 전달하고,
저장할 경로, 사용하는 모델 id(허깅페이스)를 전달해주고 실행합니다

```
python -m run.test \
    --output {저장할 경로.json} \
    --model_id {사용하는 모델 id} \
    --device cuda:0 \
    --adapter_checkpoint_path {체크포인트 경로}
```

4.2 후처리

본 레포지토리의 방법론을 사용할 시, output 내에 (## 전반적인 요약, ## speaker 1 요약, ## speaker 2 요약)과 같은 Header 들이 달리게 됩니다.

```
python -m postprocess \
--path "results/TG_SG_RR_Total_result.json" \
--output_path "results/TG_SG_RR_Total_result_postprocessed.json"
```

이를 없에주기 위해 위 코드를 통해 postprocess.py 모듈로 후처리할 수 있습니다

4.3 앙상블

앙상블(더 짧은 문장을 선택)을 하기 위해서는 후처리된 두 개의 json파일을 results 폴더에 마련한 이후

```
python -m postprocess \
--path "results/TG_SG_RR_result_postprocessed.json" \
--ensemble_path_1 "results/TG_SG_RR_Total_result_postprocessed.json" \
--output_path "results/ensemble_1.json"
```

위 코드를 통해 ensemble_1 을 얻어볼 수 있습니다

빠른 ensemble_2

가장 최고 점수의 모델인 ensemble_2를 재현하기 위해서는 총 3가지 단계를 거쳐야 합니다

```
# checkpoint-85 모델로 test.py 실행
python -m run.test \
    --output "results/TG_SG_RR_only_speaker_2_result.json" \
    --model_id "MLP-KTLim/llama-3-Korean-Bllossom-8B" \
    --tokenizer "MLP-KTLim/llama-3-Korean-Bllossom-8B" \
    --device "cuda:0" \
    --adapter_checkpoint_path "checkpoints/checkpoint-85" \
    --is_test "yes" \
    --only_speaker_2 "yes"

# TG_SG_RR_only_speaker_2_result.json에 대해 후처리
python -m postprocess \
    --path "results/TG_SG_RR_only_speaker_2_result.json" \
    --output_path "results/TG_SG_RR_only_speaker_2_result_postprocessed.json"


# 두 후처리 결과 파일을 앙상블
python -m postprocess \
    --path "results/ensemble_1.json" \
    --ensemble_path_2 "results/TG_SG_RR_only_speaker_2_result_postprocessed.json" \
    --output_path "results/ensemble_2.json"
```

이를 한 번에 수행할 수 있는 스크립트는 아래와 같습니다

```
source make_ensemble_2.sh
```

이제 results 폴더 내에 있는 후처리된 파일(~postprocessed.json)과 ensemble_1, ensemble_2를 제출하면 순위표(리더보드)에 성적이 반영됩니다

7. 참고 자료(References)

Ranzato, M., Chopra, S., Auli, M., & Zaremba, W. (2015). **Sequence level training with recurrent neural networks.** *arXiv*. https://arxiv.org/abs/1511.06732

Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015). **Scheduled sampling for sequence prediction with recurrent neural networks.** *arXiv*. https://arxiv.org/abs/1506.03099

Xu, X., Tao, C., Shen, T., Xu, C., Xu, H., Long, G., & Lou, J.-g. (2023). **Re-reading improves reasoning in large language models.** *arXiv*. https://arxiv.org/abs/2309.06275

Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). **QLoRA: Efficient finetuning of quantized LLMs.** *arXiv*. https://arxiv.org/abs/2305.14314

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). **LoRA: Low-rank adaptation of large language models.** *arXiv*. https://arxiv.org/abs/2106.09685

Liu, Y., Liu, P., Radev, D., & Neubig, G. (2022). **BRIO: Bringing order to abstractive summarization.** *arXiv*. https://arxiv.org/abs/2203.16804

[붙임 2. 모델 정성 평가 기준]

ㅇ 모델의 재현성 검증 평가 기준

- 재현성 검증은 1) 학습의 재현성 검증과 2) 모델의 재현성 검증으로 구분됩니다.
- 단, 올해 대회에서 참가팀이 사용한 학습 데이터와 고정된 'init parameter'를 사용하여 학습을 수행하는 과정을 실제로 재현하는 검증(학습의 재현성 검증)은 소스 코드를 정성적으로 평가하는 방법으로 대체하며, 운영위원회에서는 학습된 모델의 추론 결과가 순위표(리더보드)의 점수와 일치하는지 검증할 계획입니다.

※ 모델 재현성 검증 정성 평가 기준

1점	2점	3점	4점	5점
매우 미비	미비	보통	우수	매우 우수

항목	구분	평가 항목
1	학습의 재현성	참가팀의 소스 코드는 학습 코드를 포함하여 재현이 가능한가? (소스 코드 분석을 통한 재현성)
2		참가팀의 모델은 라이선스에 문제가 없고 공개된 데이터를 사용하였는가? (데이터 활용의 재현성)
3	모델의 재현성	참가팀의 소스 코드는 공개되어 다른 사람이 바로 사용할 수 있는 수준인가? (환경 설정 등을 포함한 소스 코드)
4		참가팀의 모델은 제출된 경진대회의 결과물을 재현하는가? (순위표의 점수를 재현하는지 평가)

ㅇ 모델의 우수성 평가 기준

- 모델의 우수성은 아래의 항목을 기준으로 평가합니다.

※ 모델의 우수성 검증 정성 평가 기준

1점	2점	3점	4점	5점
매우 미비	미비	보통	우수	매우 우수

항목	평가항목	분류
1	참가팀은 본 과제의 성격을 잘 이해하고 있는가?	문제 인식 및 방법론
2	참가팀이 풀고자 하는 문제는 얼마나 중요한가?	
3	참가팀이 제시한 방법론은, 참가팀이 풀고자 하는 문제에 정당한가?	
4	참가팀이 제안한 알고리즘은 참가팀의 방법론에 적합한 방법인가?	알고리즘의 우수성
5	참가팀이 제안한 알고리즘은 최신의 연구 동향을 반영한 우수한 방법인가?	
6	참가팀이 제안한 알고리즘은 독창성이 있는가?	
7	참가팀이 제출한 소스코드는 방법론 및 알고리즘을 충실히 구현하였는가?	프로그램의 우수성
8	참가팀이 제출한 소스코드는 가독성이 높고 간결하게 작성되어 활용 가능성이 높은가?	

[붙임 3. 일상 대화 요약 전문가 정성 평가 기준]

ㅇ 정성 평가 세부 기준(정성 평가 대상은 제출 결과물 중 60건)

평가 기준		평가 내용	평가
내용 (60)	충실성	요약문 내 **주제문**이 전체 내용을 잘 포괄하도록 작성되었는가?	예/아니요
		요약문이 '대화 주제별 요약'이라는 과제에 맞게 **주요 내용의 누락 없이 충실히** 요약되었는가?	예/아니요
		대화문 원문을 5어절 이상 **그대로 베낀 부분** 없이 요약을 잘 수행하였는가?	예/아니요
	정확성	원래 대화문에 비추어 보았을 때 대화 내용을 **틀리게** 요약하였거나, 정확성 판별에 있어 **모호**한 내용 없이 정확하게 요약되어 있는가?	예/아니요
조직 (30)	체계성	원래 대화에 비추어 보았을 때 해당 대화의 **전개 흐름을 반영**하여 요약문이 구성되었는가?	예/아니요
	완결성	일상 대화 요약이라는 목적에 맞게 **주제문과 화자별 요약문**을 갖춘 **하나의 요약문**으로 작성되었는가?	예/아니요
표현 (10)	응집성	**담화 표지(응집성 장치)**를 사람이 사용하는 것과 같이 자연스럽고 적절하게 사용하여 요약문을 작성하였는가?	예/아니요
	유창성	**맞춤법 규범상 오류나 오타, 어색한 표현, 비문** 없이 대화 요약문이 구성되었는가?	예/아니요