

한국어 텍스트가 포함된 책표지 이미지 생성

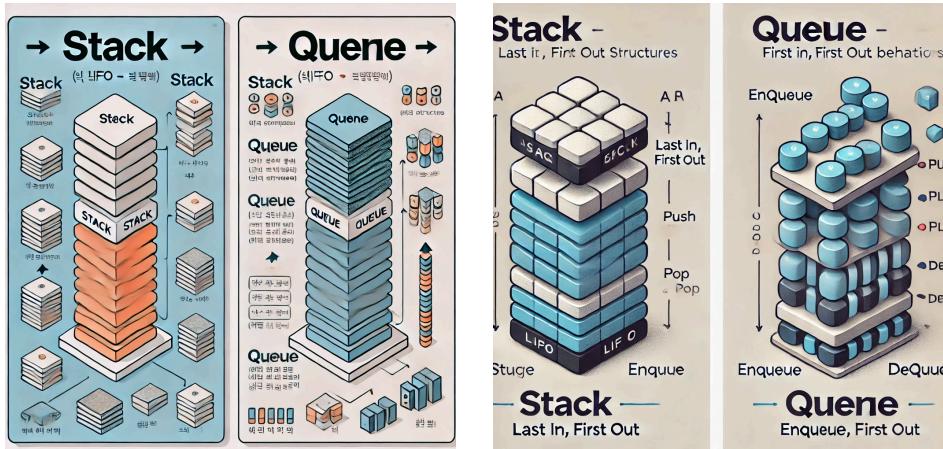
인공지능개론 기말 보고서

홍길동 2000106000

김철수 2000106000

1. 문제 정의

chatGPT의 Dall-E에서의 한국어 포함된 이미지 생성과 영어가 포함된 이미지 생성을 비교하였을 때, 한국어 text의 생성의 정확도가 확연히 낮은 것을 확인할 수 있었습니다.



<'스택과 큐를 한국어(좌측)/영어(우측)로 설명하는 이미지를 생성해줘' 프롬프트로 생성한 이미지>

위 그림에서 볼 수 있는 것과 같이, 동일한 프롬프트로 언어만 다르게 생성을 요청해보았을 때, 영어 텍스트에서는 ‘First in, First out’, ‘pop’ 등 큐와 스택의 주요 단어를 충분히 인식 가능한 정확도로 생성하는 반면, 한국어 텍스트의 경우 한글조차 생성되지 않는 것을 확인할 수 있었습니다.

따라서 본 프로젝트는 한글 텍스트가 포함된 이미지 생성에서 한글의 생성 정확도를 개선하는 것을 목표로 하였습니다.

2. 학습 준비

2-1. 모델 선정

Text Image Generation을 위해 본 프로젝트에서는 모델을 처음부터 학습시키는 것이 아닌, 사전 학습된 이미지 생성 모델을 활용하여 Transfer Learning을 하여 한글 텍스트 이미지 생성에 특화된 모델을 만드는 방식을 채택했습니다. 이미지 생성 모델이 좋은 이미지를 생성하기 위해서 수백만에서 수십 억 개의 이미지를 학습시켜야하기 때문에, 제한된 시간과 자원 내에서 가장 좋은 성능의 모델을 학습시키기 위해서 이미지 생성에 대해 이미 잘 학습이 된 모델을 이용하기로 결정하였습니다.

초기 계획은 위의 문제가 발생한 OpenAI의 DALL-E 모델을 파인튜닝하여 한국어도 생성할 수 있도록 개선하려고 하였으나, 공개되어 있는 DALL-E 모델 중 가장 최신 모델인 DALL-E 2 모델은 API 혹은 pretrained weight이 없는 GitHub 코드만 존재하기 때문에 사전 학습 모델로 사용하기에는 한계가 있었습니다. 따라서, 대안으로 유사한 텍스트-이미지 생성 모델을 선택하여 Transfer Learning을 진행하기로 계획했습니다.

Stable Diffusion 모델은 DALL-E 2와 유사하게 CLIP (Contrastive Language-Image Pre-training) 모델을 활용하여 텍스트와 이미지 간의 관계를 학습하여, 사용자의 텍스트 설명을 기반으로 이미지를 생성하는 작업에서 좋은 성능을 보이고 있는 오픈 소스 모델입니다.

Stable Diffusion은 오픈소스로 공개되어 있는 모델이기 때문에 이를 기반으로 여러 작업에 특화된 다양한 모델이 존재합니다. 이러한 모델 중, TextDiffuser 모델은 이미지에서 영어 텍스트를 생성하기 위해 최적화 된 Stable Diffusion 기반 알고리즘으로, 포스터와 책 표지 등의 다양한 데이터를 포함한 천 만개의 데이터셋(MARIO-10M)으로 사전 학습이 되어있습니다.

TextDiffuser는 2023년도 5월에 공개된 TextDiffuser와 같은 해 11월에 공개된 TextDiffuser2 2가지 버전이 있습니다. TextDiffuser는 영어 character 단위 segmentation mask를 사용하여 텍스트를 이미지로부터 구분하고, TextDiffuser2는 Bounding Box를 사용하여 텍스트를 이미지로부터 구분하는 모델입니다. 현실적으로, character-level segmentation mask를 포함한 데이터셋을 확보하는 것은 어려운 반면, Bounding Box 기반 데이터셋은 더 쉽게 접근 가능할 것이라는 판단 하에 TextDiffuser2를 최종적으로 선택하였습니다.

2-2. 데이터셋

a. 데이터셋 선정

AIHub 야외 실제 촬영 한글 이미지



AIHub의 ‘야외 실제 촬영 한글 이미지’는 일상에서 접할 수 있는 한글 이미지와 해당 이미지와 이미지에 포함되어 있는 텍스트에 대한 설명 및 좌표 등이 포함되어 있는 이미지 데이터셋입니다.

No	항목		길이	타입	필수여부	비고
	한글명	영문명				
1	데이터셋정보	info		Object		
1-1	데이터셋명	info.name	100	String	Y	
1-2	데이터셋설명	info.description	1000	String		
1-3	데이터셋URL	info.url	200	String		
1-4	데이터셋생성일자	info.date_created	100	String	Y	
1-5	데이터날씨	info.weather	100	String	Y	
1-6	밤낮	info.sun	100	String	Y	
2	이미지정보	images		List		
2-1	이미지식별자	images[].id	100	String	Y	
2-2	이미지너비	images[].width	4	Number	Y	
2-3	이미지높이	images[].height	4	Number	Y	
2-4	이미지파일명	images[].file_name	100	String	Y	
2-5	이미지라이선스	images[].license	100	String	Y	
2-6	이미지촬영일자	date_created	100	String	Y	
3	어노테이션정보	annotations		List		
3-1	어노테이션 식별자	annotations[].id	100	String	Y	
3-2	인식문자이미지식별자	annotations[].image_id	100	String	Y	
3-3	어노테이션 텍스트	annotations[].text	1000	String	Y	
3-4	어노테이션 속성	annotations[].attributes	1	Object		
3-5	어노테이션 바운딩박스	annotations[].bbox	4	List		
4	라이선스	licenses		List		
4-1	라이선스명	licenses.name	100	String	Y	
4-2	라이선스URL	licenses.url	200	String	Y	

‘야외 실제 촬영 한글 이미지’ 데이터셋은 45만개의 간판과 5만 개의 책 표지 데이터셋으로 구성되어 있습니다. 이 이미지들은 텍스트 인식이 비교적 쉬운 2D 백터 이미지가 아닌 실제 사진에 포함된 텍스트 이미지이면서도, 텍스트가 이미지에서 큰 비중을 차지하고 있기 때문에, 모델이 현실적인 환경에서의 텍스트와 이미지 간 관계를 학습하는데 적합하다고 판단하여 선정하였습니다.

제한된 기간 내에 최대한 좋은 성능을 달성하기 위하여 ‘한글이 포함된 책 표지 생성’이라는 작은 작업에 집중하는 것이 좋을 것이라고 판단하였고, 이에 따라 5만 개의 책 표지만 사용하기로 결정하였습니다.

b. 전처리 과정

데이터셋의 용량이 방대한 만큼 시간이 오래 소요되어 사용자가 진행상황을 알 수 있고, 중간에 문제가 생기더라도 다시 그 부분부터 실행할 수 있도록 코드를 작성하였습니다.

1. 결과 디렉토리 생성

스크립트: **makeResultDir.py**

- 역할:
 - 학습 결과를 저장할 디렉토리 구조를 생성.
 - 기존 디렉토리가 없는 경우 새로 생성하여 후속 작업의 기반을 마련.

코드 실행:

```
subprocess.run(['python', 'makeResultDir.py'], capture_output=True,
text=True)
```

- 성공 시 메시지 출력: "makeResultDir.py 스크립트 실행 완료."
- 실패 시 오류 메시지 출력 및 프로세스 종단.

2. 한글 이미지 압축 해제 및 파일 구조 정리

스크립트: korImgPro.py

- 역할:
 - 지정된 경로에서 ZIP 파일을 탐색하여 해제.
 - 폴더 이름과 ZIP 파일 이름 간 매핑을 통해 각 파일이 적절한 위치에 저장되도록 관리.
 - 한글 파일명을 올바르게 처리하여 파일 손상을 방지.

주요 로직:

1. 경로에 존재하는 ZIP 파일 목록 탐색.
2. ZIP 파일 수와 매핑된 이름 목록의 일치 여부 확인.
 - 초과 시 오류 메시지 출력.
3. 파일 해제:
 - 압축 파일 내 각 파일을 디코딩하여 올바른 위치에 저장.

코드 실행:

```
for folder in folders:
    zip_files = [f for f in os.listdir(folder) if f.endswith('.zip')]
    for zip_file in zip_files:
        with zipfile.ZipFile(zip_path, 'r') as zip_ref:
            ...

```

- 처리 완료 시 메시지 출력: "모든 ZIP 파일이 성공적으로 해제되었습니다."

3. 데이터 전처리 및 최종 파일 생성

스크립트: makeResult.py

- 역할:
 - 각 라벨 폴더를 순회하며 JSON 파일을 읽고, 데이터를 기반으로 OCR 파일 및 캡션 파일 생성.
 - 관련 이미지를 원본 폴더에서 찾아 결과 폴더로 복사.

주요 로직:

1. JSON 파일 읽기:
 - 각 JSON 파일에서 이미지 정보(images)와 주석 정보(annotations)를 추출.
2. ocr.txt 생성:

- 주석 정보에서 텍스트와 경계 상자 정보를 읽어 파일로 저장.
- 특정 조건(예: "xxx"인 텍스트 제외)을 적용하여 데이터 정제.

3. `caption.txt` 생성:

- 텍스트 주석을 결합하여 한 줄로 작성.

4. 이미지 복사:

- JSON 파일에 명시된 이미지 파일명을 기준으로 원본 폴더에서 결과 폴더로 복사.

코드 실행:

```
with open(json_path, "r", encoding="utf-8") as f:
    data = json.load(f)
    ...
    ocr_file.write(ocr_line)
    ...
shutil.copy2(origin_image_path, target_image_path)
```

- 처리 완료 시 메시지 출력: "모든 작업이 완료되었습니다."

Train/Val/Test Dataset: AIHub 데이터가 3개로 이미 나뉘어서 제공

Train: 야외실 제촬영한글이미지 Train 사용

Val: 야외실 제촬영한글이미지 Validation 사용

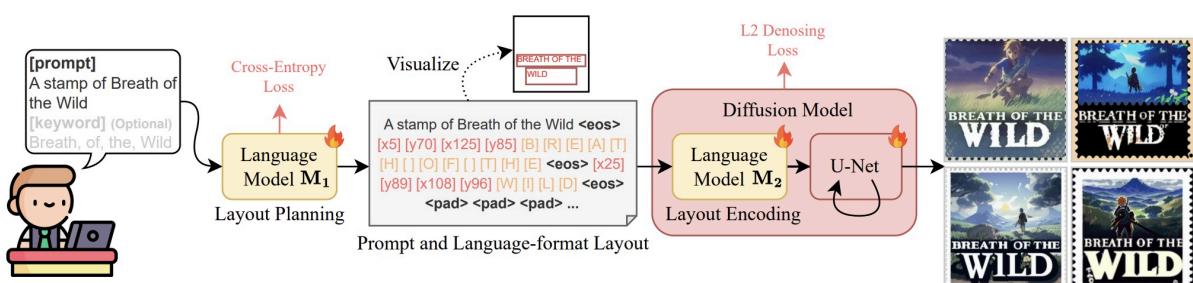
Test: 야외실 제촬영한글이미지 30사용

3. 모델 구축

본 프로젝트의 목표인 '사용자가 제공하는 책 제목 **Prompt**를 기반으로 한국어 텍스트가 포함된 책 표지 이미지 생성'을 위하여 모델 파이프라인을 구축하였습니다.

3-1) Pre-trained Model

본 프로젝트에서 **fine-tune**을 하기 위해 사용되었던 **text-diffuser2** 모델의 구조는 간단히 보면 다음과 같습니다.



- 1) 사용자가 원하는 이미지를 설명하는 **prompt**를 입력합니다.
- 2) 대형 언어 모델(M_1)을 이용하여 **prompt**를 기반으로 키워드를 추출하고, 해당 키워드를 입력할 적절한 위치를 선정하여 키워드를 입력할 **layout**을 생성합니다.
- 3) 다른 대형 언어 모델(CLIP)을 이용하여
 - a) **prompt**의 의미를 전달하기 위해 단어 단위로 토큰화하고,
 - b) **layout**에서 생성된 키워드를 [] 내부에 한 글자씩 넣어 따로 토큰화 (예: [B], [R]) 한 후,
 - c) **layout**에서 생성된 좌표 토큰을 [] 내부에 삽입

- 4) 3번에서 생성된 새로운 CLIP text-encoding을 이용해서 UNet과 VAE로 Diffusion 진행

따라서 이 과정에서 총 4개의 인공지능 모델(M1, M2(CLIP), UNet, VAE)이 사용되었습니다.

3-2) 학습할 모델 선정

TextDiffuser2에서 사용되는 4개의 모델 중, 시간과 자원의 제약으로 인해 추가적으로 학습을 진행해야하는 모델을 최소화하는 것이 좋을 것이라고 판단하였습니다.

(1) M1 모델

M1 모델은 Llama 2라는 대형 언어모델을 shareGPT의 대화 내용 데이터를 통해 fine-tuned된 Vicuna라는 모델을 이용하여, 텍스트 이미지 생성에 관한 prompt를 받으면 layout을 생성하도록 다시 layout_planner_data_5k fine-tuning한 대형 언어모델입니다.

layout_planner_data_5k 데이터 예시

```
{  
    "id": "identity_0",  
    "conversations": [  
        {  
            "from": "human",  
            "value": "Given a prompt that will be used to generate an image, plan the layout of visual text  
            for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128,  
            including the coordinates of top, left, right, and bottom. All keywords are included in the caption. You dont need  
            to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let  
            us begin. Prompt: FirstService Residential builder"  
        },  
        {  
            "from": "gpt",  
            "value": "FirstService 0,91,127,112\nRESIDENTIAL 51,117,127,127\n"  
        }  
    ]  
},
```

위 데이터에서 확인할 수 있듯이, M1 모델은 영어 prompt로만 훈련이 되어있습니다. 따라서 한국어 대상으로는 정상적으로 작동하지 않으나, 실험 결과 그러나, 대부분의 prompt가 기존 영어 template에 맞게 훈련이 되어 있을 경우, 한국어인 책 제목을 넣어도 우리가 원하는 task (키워드 분리 및 좌표 생성)이 잘 수행되는 것을 확인할 수 있었습니다 (Appendix A).

본 언어 모델은 shareGPT와 5000개의 layout 생성 prompt로 학습한 거대 언어 모델로, 이를 언어를 변경하여 다시 학습하기 위해서는 많은 시간과 자원이 필요할 것이라고 생각하였습니다. 또한 해당 모델은 레이아웃만 자동으로 생성해주는 모델이기 때문에 레이아웃이 잘 생성이 된다면 문제가 없을 것이라고 판단하였습니다. 한글 책 표지만 생성하는 본 프로젝트에서는 기본 prompt는 영어로, 키워드인 제목만 한국어로 제공하여, 언어 모델을 변경 없이 그대로 재사용 할 수 있도록 하였습니다.

(2) CLIP 모델

CLIP 모델은 텍스트와 이미지를 같은 공간으로 임베딩 하여 이미지와 텍스트의 관계를 학습하는 모델입니다. 기존 CLIP 모델은 이미지와 영어 텍스트를 기반으로 Text Encoding이 학습되었을 것이기 때문에 이미지와 한국어 텍스트 간의 관계를 학습하기 위해 fine-tuning을 하기로 하였습니다.

(3) U-Net 모델

U-Net 모델은 CLIP의 Embedding을 받아서 노이즈에서 이미지를 생성하는 모델이기 때문에, CLIP 모델의 embedding이 변경되는 현 상황에서 학습이 필요하다고 판단하였습니다.

(4) VAE 모델

이미지를 latent space로 변환해주고 latent space를 다시 이미지로 복원해주는 모델로, 이미지 생성의 효율성을 높여주는 모델입니다. TextDiffuser2에서도 기존 Stable Diffusion의 VAE를 그대로 사용해주었기 때문에 추가 학습이 성능에 크게 영향을 미치지 않을 것이라고 판단하여 학습을 진행하지 않았습니다.

3-3) Full fine tuning 여부

초기 계획은 CLIP 모델을 새로운 한국어 CLIP 모델로 교체하는 것을 가정하고 훈련을 준비하였고, 1만 개의 많은 데이터를 사용하기 때문에 기존 가중치를 고정시키고 새로운레이어를 만들어주는 LoRA보다는 모든 가중치를 업데이트하는 Full fine tuning이 적절할 것이라고 생각하였습니다.

4. 모델 학습

4-1) 한국어 CLIP 모델 만들기 (CLIP_Generation.ipynb, CLIP_TEST.ipynb)

초기 단계에서는 영어 데이터 기준으로 제작된 CLIP 모델을 huggingface에 존재하는 한국어 CLIP 모델 (Bingsu)로 교체한 후 사용하려고 하였습니다.

그러나 M1 모델의 한계로 인해서 책 제목을 제외한 모든 text 입력이 영어로 들어가도록 계획이 변경되었고, TextDiffuser2의 U-Net이 이미 영어 CLIP으로 잘 학습되어 있기 때문에, 영어 텍스트에 대해서는 영어 CLIP의 weight도 최대한 활용할 수 있도록 할 방법을 고안해보았습니다.

따라서 영어 CLIP에 한국어 CLIP의 Token과 Embedding 추가한 custom CLIP 모델을 만드는 방법으로 계획을 변경하였습니다.

The new embeddings will be initialized from a multivariate normal distribution that has old embeddings' mean and covariance

한국어와 영어 CLIP token의 embedding 벡터 크기는 768로 동일하였으나, 크기가 동일하더라도 각 임베딩 값이 의미하는 내용은 다를 수 있을 것이라고 생각하였습니다. 그러나 한국어 토큰을 추가하고 embedding 값을 지정하지 않는다고 한다면, 기존 embedding 값의 통계 값에서 초기화되기 때문에 한국어 CLIP의 embedding을 사용하는 것이 비교적 단어와 단어 사이의 관계를 더 잘 나타낼 것이라고 생각하였습니다. 따라서 이 두 CLIP 모델을 병합할 때, 한국어 CLIP에서 가져온 Embedding은 parameter를 추가하여 이미 잘 학습되어 있는 기존 embedding들과 차이를 주게끔 만들어두었습니다.

```
# 추가된 단어에 대해 한국어 CLIP 모델의 단어
for token in new_tokens:
    token_id_ko = ko_tokenizer.convert_tokens_to_ids(token)
    token_id_en = tokenizer.convert_tokens_to_ids(token)
    en_embedding.weight.data[token_id_en] = (
        ko_embedding.weight.data[token_id_ko] + alpha * korean_language_tag
    )
```

그러나 이렇게 병합을 하였을 때 문제점이 존재하였는데 한국어 CLIP에 포함된 영어 토큰으로 인해 기존 Tokenizer에서는 정상적으로 처리되던 긴 영단어가 쪼개지는 문제점이

있었습니다 (Appendix B). 따라서 한국어 토큰을 추가할 때 영어 단어가 추가되지 않도록 처리해주었습니다.

```
# 한국어 Tokenizer의 토큰 중 영어 Tokenizer에 없고 영어가 아닌 토큰 new_tokens 생성 (최대한 기존 tokenizer & encoder 재사용)
new_tokens = [token for token in ko_tokens if token not in en_vocab and re.fullmatch(r'[a-zA-Z]+', token) is None]
```

또한 기존 TextDiffuser2에서는 CLIP에서 키워드의 각 글자와 좌표에 대하여 개별적으로 Tokenizing 및 임베딩을 거치기 때문에 각 단어와 좌표 (520 * 520 이미지 기준)을 token으로 추가해주었습니다. (키워드 토큰(예:[a], [감])과 좌표 토큰 [예:[1510]])

```
# 영어 TextDiffuser2 알파벳
import string
alphabet = string.digits + string.ascii_lowercase + string.ascii_uppercase + string.punctuation + ' ' # len(alphabet) = 95
...alphabet
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&^<>?@[\₩₩₩]~_`{|}~

# alphabet에 한국어 추가
alphabet += ''.join(chr(i) for i in range(ord('가'), ord('힝') + 1))

#### additional tokens are introduced, including coordinate tokens and character tokens
print('*****')
print(len(tokenizer))
for i in range(520):
    tokenizer.add_tokens(['l' + str(i)]) # left
    tokenizer.add_tokens(['t' + str(i)]) # top
    tokenizer.add_tokens(['r' + str(i)]) # width
    tokenizer.add_tokens(['b' + str(i)]) # height
for c in alphabet:
    tokenizer.add_tokens([f'{c}'])
print(len(tokenizer))
print('*****')

text_encoder.resize_token_embeddings(len(tokenizer))
```

이후 CLIP_Test.ipynb로 기존 CLIP, 한국어 CLIP, Custom CLIP가 실제 TextDiffuser 코드에서 사용될 때 동일하게 작동하는지 확인해보았습니다.

```
sentence = "This is a sentence."
test_clip_functionality(text_encoder, tokenizer, sentence)

== Testing CLIP Functionality ==
[Resize Token Embeddings Test] Passed.
[Encoding Prompts Test] Passed.
Encoded Prompt: tensor([[49406, 589, 533, 320, 12737, 269, 49407]])
[Decoding Prompts Test] Passed.
Decoded Prompt: <|startoftext|>this is a sentence . <|endoftext|>
[Padding Prompts Test] Passed.
Padded Prompt: [49406, 589, 533, 320, 12737, 269, 49407, 49407]
[Model Forward Pass (Conditional) Test] Passed.
Output Shape (Conditional): torch.Size([4, 77, 768])
[Model Forward Pass (Non-Conditional) Test] Passed.
Output Shape (Non-Conditional): torch.Size([4, 77, 768])
[Integration Test] Passed.
Composed Prompt: <|startoftext|>this is a sentence . <|endoftext|><|endoftext|>
== Testing Completed ==
```

```
# 기존 TextDiffuser에서 사용한 CLIP
test_clip_functionality(en_text_encoder, en_tokenizer, sentence)

==== Testing CLIP Functionality ====
[Resize Token Embeddings Test] Passed.
[Encoding Prompts Test] Passed.
Encoded Prompt: tensor([[49406, 589, 533, 320, 12737, 269, 49407]])
[Decoding Prompts Test] Passed.
Decoded Prompt: <|startoftext|>this is a sentence . <|endoftext|>
[Padding Prompts Test] Passed.
Padded Prompt: [49406, 589, 533, 320, 12737, 269, 49407, 49407]
[Model Forward Pass (Conditional) Test] Passed.
Output Shape (Conditional): torch.Size([4, 77, 768])
[Model Forward Pass (Non-Conditional) Test] Passed.
Output Shape (Non-Conditional): torch.Size([4, 77, 768])
[Integration Test] Passed.
Composed Prompt: <|startoftext|>this is a sentence . <|endoftext|><|endoftext|>
==== Testing Completed ===
```

```
# 한국어 CLIP
test_clip_functionality(ko_text_encoder, ko_tokenizer, sentence)

==== Testing CLIP Functionality ====
[Resize Token Embeddings Test] Passed.
[Encoding Prompts Test] Passed.
Encoded Prompt: tensor([[49406, 946, 561, 245, 23819, 295, 49407]])
[Decoding Prompts Test] Passed.
Decoded Prompt: <|startoftext|>this is a sentence . <|endoftext|>
[Padding Prompts Test] Passed.
Padded Prompt: [49406, 946, 561, 245, 23819, 295, 49407, 49407]
[Model Forward Pass (Conditional) Test] Passed.
Output Shape (Conditional): torch.Size([4, 77, 768])
[Model Forward Pass (Non-Conditional) Test] Passed.
Output Shape (Non-Conditional): torch.Size([4, 77, 768])
[Integration Test] Passed.
Composed Prompt: <|startoftext|>this is a sentence . <|endoftext|><|endoftext|>
==== Testing Completed ===
```

4-2) 사용한 데이터셋 관리 (DatasetCheckPoint.ipynb)

파일 단위로 전처리한 후 사용하게 된 1만 개의 데이터를 한 번에 로딩할 경우, 데이터셋 로딩 도중에 코랩이 끊기기 때문에 500개씩 불러와 사용해주었습니다. 이 때 런타임을 매번 초기화하기 때문에 DatasetCheckPoint.ipynb로 사용하지 않은 데이터셋 리스트를 별도의 파일 unused_dataset.txt를 생성하여 관리해주었습니다. 학습을 시작하기 전, DatasetCheckPoint에서 생성한 후, Train_TextDiffuser2.ipynb에서 학습이 정상적으로 끝나면 사용한 데이터 리스트를 pop 해주어 나머지 데이터셋 이름을 다시 unused_dataset.txt에 저장하는 형식입니다. 처음 저장할 때 목록을 한 번 shuffle 해주었습니다.

```
import random
random.shuffle(folder_names)

file_name = PIPELINE_PATH + 'unused_dataset.txt'
with open(file_name, "w") as f:
    for folder in folder_names:
        f.write(f"{folder}\n")
```

4-3) 모델 학습 주요 코드 설명 (Train_TextDiffuser2)

모델 학습 코드는 기존 TextDiffuser의 Fine-tuning 코드를 참고하여 준비하였습니다.

(a) 하이퍼파라미터 정의

```

train_batch_size = 3
num_epochs = 3
learning_rate = 1e-6
gradient_accumulation_steps = 2 #2 스텝 동안 gradient를 누적한 후 한 번의 업데이트 수행 (실질적인 배치 크기 = batch_size * gradient_accumulation_steps)
n_workers = 8 # DataLoader의 프로세서
max_train_steps = None
max_grad_norm = 1 #Max Gradient Norm

```

```

use_ema = True
allow_tf32 = True
snr_gamma = None #float, SNR weighting gamma to be used if rebalancing the loss. Recommended value is 5.0.
enable_xformers_memory_efficient_attention = True

#Optimizer 설정
adam_beta1 = 0.9 # default, The beta1 parameter for the Adam optimizer.
adam_beta2 = 0.999 #default The beta2 parameter for the Adam optimizer.
adam_weight_decay = 1e-2 #default, weight decay to use
adam_epsilon = 1e-08 #Epsilon value for the Adam optimizer

```

하이퍼파라미터를 정의해주었습니다. 컴퓨팅 단위의 제한으로 인해 실험 횟수에 제한이 있었기 때문에 하이퍼파라미터는 최대한 안전하게 기존 **TextDiffuser2** 모델과 비슷하게 하였으나, **batch_size**나 **epoch**은 colab의 한계로 인해 기존 **TextDiffuser**보다 낮게 설정해두었습니다.

(b) 모델 로드

모델 로드는 **Fine-tuning**을 처음 시작할 때와 이후로 나누었습니다.

<first-round>

```

# Scheduler
noise_scheduler = DOPMScheduler.from_pretrained('runwayml/stable-diffusion-v1-5', subfolder="scheduler")

#Load CLIP
from transformers import CLIPTextModel, CLIPTokenizer
import torch
from copy import deepcopy

if os.path.exists(CLIP_PATH):
    print(f"CLIP found in {MODEL_PATH}. Loading...")
    loaded_tokenizer = CLIPTokenizer.from_pretrained(CLIP_PATH)
    print(f"CLIP Tokenizer loaded successfully!")
    loaded_text_encoder = CLIPTextModel.from_pretrained(CLIP_PATH)
    print(f"CLIP Text Encoder loaded successfully!")
else:
    print(f"CLIP not found in {MODEL_PATH}. Please run CLIP_Generation.ipynb to generate custom CLIP Model first")

# TODO: 코드 실행 완료 후 deep copy 삭제하고 loaded_tokenizer와 loaded_text_encoder를 tokenizer와 text_encoder로 변경
text_encoder = deepcopy(loaded_text_encoder)
tokenizer = deepcopy(loaded_tokenizer)

vae = AutoencoderKL.from_pretrained('runwayml/stable-diffusion-v1-5', subfolder="vae").cuda()
unet = UNet2DConditionModel.from_pretrained('JingyeChen22/textdiffuser2-full-ft', subfolder='unet').cuda()

vae.requires_grad_(False) #VAE Train 안함
text_encoder.requires_grad_(True) # 새로운 토큰 학습 (ltrb & 한국어)

# Create EMA for the unet.
if use_ema:
    ema_unet = UNet2DConditionModel.from_pretrained('JingyeChen22/textdiffuser2-full-ft', subfolder='unet')
    ema_unet = EMAModel(ema_unet.parameters(), model_cvis=UNet2DConditionModel, model_config=ema_unet.config)

```

<0|후 round>

```

FINETUNED_CLIP_ENCODER = PIPELINE_SAVE_PATH + 'text_encoder/'
FINETUNED_UNET = PIPELINE_SAVE_PATH + 'unet/'

# CLIP & Scheduler 로드
# Scheduler
noise_scheduler = DOPMScheduler.from_pretrained('runwayml/stable-diffusion-v1-5', subfolder="scheduler")

#Load CLIP
from transformers import CLIPTextModel, CLIPTokenizer
import torch
from copy import deepcopy

if os.path.exists(CLIP_PATH):
    print(f"CLIP found in {MODEL_PATH}. Loading...")
    loaded_tokenizer = CLIPTokenizer.from_pretrained(CLIP_PATH)
    print(f"CLIP Tokenizer loaded successfully!")
    loaded_text_encoder = CLIPTextModel.from_pretrained(FINETUNED_CLIP_ENCODER)
    print(f"CLIP Text Encoder loaded successfully!")
else:
    print(f"CLIP not found in {MODEL_PATH}. Please run CLIP_Generation.ipynb to generate custom CLIP Model first")

# TODO: 코드 실험 완료 후 deepcopy 삭제하고 loaded_tokenizer와 loaded_text_encoder를 tokenizer와 text_encoder로 변경
text_encoder = deepcopy(loaded_text_encoder)
tokenizer = deepcopy(loaded_tokenizer)

vae = AutoencoderKL.from_pretrained('runwayml/stable-diffusion-v1-5', subfolder="vae").cuda()
unet = UNet2DConditionModel.from_pretrained(FINETUNED_UNET).cuda()

vae.requires_grad_(False) #VAE Train 안함
text_encoder.requires_grad_(True) # 새로운 토큰 학습 (lrb & 한국어)

# Create EMA for the unet.
if use_ema:
    ema_unet = UNet2DConditionModel.from_pretrained(FINETUNED_UNET, subfolder="unet")
    ema_unet = EMAModel(ema_unet.parameters(), model_cls=UNet2DConditionModel, model_config=ema_unet.config)

```

first-round에는 CLIP Tokenizer와 Encoder를 Custom CLIP에서 바로 가지고 왔고, U-net을 TextDiffuser2의 huggingface에서 바로 이용해주었습니다. 이후 round들에서는 CLIP Encoder와 U-net을 Fine-tune한 path에서 가져와주었습니다.

학습 중 모델 가중치 업데이트 여부 설정을 해주었습니다. (CLIP, U-Net: requires_grad(True), VAE: requires_grad(False))

학습 안정성을 위하여 ema를 활성화해주었습니다.

(c) Optimizer 정의

```

# Initialize the optimizer
optimizer_cls = torch.optim.AdamW

#### train the u-net and text encoder
#### the lr for training u-net is set to a lower value (1e-5) following textdiffuser-1
optimizer = optimizer_cls(
    [
        {'params': text_encoder.parameters(), 'lr': 1e-5},
        {'params': unet.parameters(), 'lr': 1e-5},
    ],
    lr=learning_rate,
    betas=(adam_beta1, adam_beta2),
    weight_decay=adam_weight_decay,
    eps=adam_epsilon
)

```

Optimizer는 가중치 감소가 가능한 Adam Optimizer인 AdamW를 사용해주었으며, CLIP과 U-Net의 parameter 모두 학습이 필요하기 때문에 unet.parameters() & learning rate, text_encoder.parameters() & learning rate 각각 optimizer에 전달하였습니다. learning rate 외의 나머지 파라미터 값은 textdiffuser2와 동일하게 설정해주었습니다.

(d) 데이터셋 전처리

`unused_dataset.txt`에서 아직 사용이 되지 않은 데이터명을 최대 500개까지 불러와서 전처리를 해주었습니다. 불러온 데이터는 리스트에서 `pop`을 해주었고, 해당 리스트는 학습이 정상적으로 종료된 후 다시 `unused_dataset.txt`를 업데이트하는데 사용해주었습니다 (학습이 정상적으로 끝나지 않을 경우 해당 데이터로 다시 학습을 재개하기 위함)

```
if not folder_names:
    print('Data Loading Completed')
    break
if count > 500:
    break
count += 1
#print(folder, folder_names[0])
_ = folder_names.pop(0)
```

이전 파일 단위 전처리에서 `caption.txt`에는 ‘어학원’ 등 이미지 내부의 텍스트 정보만 포함되게 하였기 때문에. ‘book cover with a title of’와 같이 책 표지라는 `context`를 조금 더 추가해주었습니다.

```
# 학습하는 모든 데이터가 book cover이기 때문에 caption에 해당 내용 추가
caption = 'book cover with a title of ' + caption
```

`ocr.txt`에 있는 값들을 좌표 토큰으로 변환해주었습니다. 좌표 토큰은 이미지가 512*512 사이즈라고 가정하여 `l,r,t,b` 모두 520까지 정의가 되어 있으나, 실제로는 이미지가 더 큰 경우가 많았기 때문에 CLIP에 존재하지 않는 좌표 토큰이 생성될 위험이 있었습니다. 따라서 이미지 크기가 512 사이즈의 이미지로 변했을 때의 `scale_factor`를 계산해주어 `ocr.txt`의 값들을 변환해준 후 `encoding`을 진행해주었습니다.

```
ocrs = open(f'{DATASET_PATH}/{folder}/ocr.txt'),readlines()

ocrs_temp = []
for line in ocrs:
    #print(line)
    line = line.strip()
    items = line.split(',')
    #items = box.split(',')
    pred = ""
    for item in items[0:-9]:
        pred += item
    #print(pred, pred)
    #print(items)
    scale_factor_x = width / 512
    scale_factor_y = height / 512
    x1, y1, x2, y2, x3, y3, x4, y4 = int(items[-9]), int(items[-8]), int(items[-7]), int(items[-6]), int(items[-5]), int(items[-4]), int(items[-3]), int(items[-2])
    x_min = min(x1, x2, x3, x4)
    y_min = min(y1, y2, y3, y4)
    x_max = max(x1, x2, x3, x4)
    y_max = max(y1, y2, y3, y4)
    x_center = (x_min + x_max) // 2
    y_center = (y_min + y_max) // 2
    x1, x2, x3, x4 = int(x1 // scale_factor_x), int(x2 // scale_factor_x), int(x3 // scale_factor_x), int(x4 // scale_factor_x)
    y1, y2, y3, y4 = int(y1// scale_factor_y), int(y2// scale_factor_y), int(y3// scale_factor_y), int(y4// scale_factor_y)
    ocrs_temp.append([x_center, y_center, x_min, y_min, x_max, y_max, pred])
ocrs_temp = sorted(ocrs_temp, key=lambda x: x[0])
ocrs_temp = merge_boxes(ocrs_temp)
ocrs_temp = sorted(ocrs_temp, key=lambda x: x[1])
```

```

ocr_ids = [] ##### concat with the prompt tokens
for line in ocrs_temp:
    x_center, y_center, x_min, y_min, x_max, y_max, pred = line

    x_left = x_min
    y_top = y_min
    x_right = x_max
    y_bottom = y_max
    x_left = x_left // (512 // granularity)
    y_top = y_top // (512 // granularity)
    x_right = x_right // (512 // granularity)
    y_bottom = y_bottom // (512 // granularity)
    x_left = np.clip(x_left, 0, granularity)
    y_top = np.clip(y_top, 0, granularity)
    x_right = np.clip(x_right, 0, granularity)
    y_bottom = np.clip(y_bottom, 0, granularity)
    ocr_ids.append(f'{x_left}{y_top}{x_right}{y_bottom}{pred}')

char_list = list(pred)
char_list = [f'[{i}]' for i in char_list]
ocr_ids.extend(char_list)
ocr_ids.append(tokenizer.eos_token_id)
ocr_ids.append(tokenizer.eos_token_id)

ocr_ids = tokenizer.encode(ocr_ids)

```

U-Net을 위한 조건부 prompts_cond = caption + ocr + padding, 비조건부 prompts_nocond = padding 생성한 후, prompts_train은 90% 확률로 prompts_cond를, 10% 확률로 prompts_nocond를 선택해주어 U-Net이 대부분 prompt에 맞는 이미지를 생성하고, 10% 확률로 prompts가 없이 이미지 자체의 생성을 학습하도록 해주었습니다.

```

prompt = caption_ids + ocr_ids
prompt = prompt[:??]
while len(prompt) < ??:
    prompt.append(tokenizer.pad_token_id)

prompts_cond.append(prompt)
prompts_nocond.append([tokenizer.pad_token_id]*??)

#### classifier-free guidance
if random.random() < 0.1:
    prompts_train.append([tokenizer.pad_token_id]*??)
else:
    prompts_train.append(prompt)

if count%10==0:
    print(f'Number of Data Loaded: {count}')

```

마지막으로 torch.Transform을 이용하여 이미지를 ToTensor와 Resize(512, 512)를 해주었고, 이후 모델에 맞는 값으로 이미지를 정규화 해주었습니다.

```

[train_transforms(image).sub_(0.5).div_(0.5) for image in images]

train_transforms = transforms.Compose(
    [
        transforms.Resize((512, 512)),
        transforms.ToTensor(),
    ]
)

```

(e) DataLoader 준비

Preprocess한 데이터를 Custom Dataset으로 만들어 iteration을 편리하게 만들어주고, 데이터를 shuffle하여 Dataloader를 생성해주었습니다. 맨 위에 unused_dataset.txt를 이용하여 남은 데이터의 개수가 나타나게 하였습니다. 아래 결과는 마지막으로 코드를 돌렸을 때이기 때문에 총 446개의 데이터가 남아있었고 이 데이터가 모두 로딩이 되어 남은 데이터가 0이 된 것을 확인할 수 있습니다. 데이터 로딩과 동시에 전처리를 진행하기 때문에 데이터 로딩 시간이 길었고, 그렇기 때문에 10개가 로딩될 때마다 상태를 표시해주었습니다.

```

from torch.utils.data import Dataset, DataLoader

class CustomDataset(Dataset):
    def __init__(self, examples):
        self.images = examples["images"]
        self.prompts_train = examples["prompts_train"]
        self.prompts_cond = examples["prompts_cond"]
        self.prompts_nocond = examples["prompts_nocond"]

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

    def __getitem__(self, idx):
        return {
            "images": self.images[idx],
            "prompts_train": self.prompts_train[idx],
            "prompts_cond": self.prompts_cond[idx],
            "prompts_nocond": self.prompts_nocond[idx],
        }

print('Initial Number of Leftover Datasets:', len(folder_names))
# Preprocess the data
with accelerator.main_process_first():
    examples = preprocess_train()

# Wrap the examples into a Dataset
train_dataset = CustomDataset(examples)
print("Train dataset length: {len(train_dataset)}")

# DataLoader creation
train_dataloader = DataLoader(
    train_dataset,
    shuffle=True,
    collate_fn=None, # collate_fn은 이미지 CustomDataset에서 데이터가 준비되므로 불필요
    batch_size=train_batch_size,
    num_workers=n_workers,
)

# 남은 데이터셋 업데이트
print('Remaining Number of Leftover Datasets:', len(folder_names))

Initial Number of Leftover Datasets: 446
Number of Data Loaded: 10
Number of Data Loaded: 20
Number of Data Loaded: 30

```

Number of Data Loaded: 440
 Train dataset length: 446
 Remaining Number of Leftover Datasets: 0

```

# Test the DataLoader
for idx, batch in enumerate(train_dataloader):
    print(f"Batch {idx}:")
    print(f"Images shape: {batch['images'].shape}")
    print(f"Prompts (train): {batch['prompts_train'].shape}")
    print(f"Prompts (cond): {batch['prompts_cond'].shape}")
    print(f"Prompts (no-cond): {batch['prompts_nocond'].shape}")
    break

Batch 0:
Images shape: torch.Size([3, 3, 512, 512])
Prompts (train): torch.Size([3, 77])
Prompts (cond): torch.Size([3, 77])
Prompts (no-cond): torch.Size([3, 77])

```

데이터 로더를 확인해주면 다음과 같이 이미지는 512*512의 RGB 이미지로, prompts는 모두 77개의 토큰으로 batch_size=3으로 잘 생성된 것을 확인할 수 있습니다.

(f) 학습

`Lr_scheduler`을 사용하여 점진적으로 learning rate를 줄여 처음에는 최적의 파라미터에 근접하게 해주고, 이후에는 최적의 파라미터를 지나치지 않도록 조정해주었습니다.

```
lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=lr_warmup_steps * gradient_accumulation_steps,
    num_training_steps=max_train_steps * gradient_accumulation_steps,
)
```

전체 모델 파이프라인을 accelerator에 넣어 준비해 주었습니다.

```
# Prepare everything with our `accelerator`.
unet, text_encoder, optimizer, train_dataloader, lr_scheduler = accelerator.prepare(
    unet, text_encoder, optimizer, train_dataloader, lr_scheduler
)

if use_ema:
    ema_unet.to(accelerator.device)
```

VAE를 이용하여 이미지를 latent space로 변환한 후, 랜덤한 노이즈를 추가해주었습니다. 이후 학습에 사용할 prompts인 prompts_train을 CLIP으로 인코딩 해주었습니다.

```
# Convert Images to latent space
latents = vae.encode(batch["images"].to(weight_dtype)).latent_dist.sample()
latents = latents * vae.config.scaling_factor

# Sample noise that we'll add to the latents
noise = torch.randn_like(latents)
if noise_offset:
    # https://www.crosslabs.org/blog/diffusion-with-offset-noise
    noise += noise_offset + torch.randn(
        (latents.shape[0], latents.shape[1], 1, 1), device=latents.device
    )
if input_perturbation:
    new_noise = noise + input_perturbation * torch.randn_like(noise)
bsz = latents.shape[0]
# Sample a random timestep for each image
timesteps = torch.randint(0, noise_scheduler.config.num_train_timesteps, (bsz,), device=latents.device)
timesteps = timesteps.long()

# Add noise to the latents according to the noise magnitude at each timestep
# (this is the forward diffusion process)
if input_perturbation:
    noisy_latents = noise_scheduler.add_noise(latents, new_noise, timesteps)
else:
    noisy_latents = noise_scheduler.add_noise(latents, noise, timesteps)

# Get the text embedding for conditioning
encoder_hidden_states = text_encoder(batch["prompts_train"])[0]

# Get the target for loss depending on the prediction type
if noise_scheduler.config.prediction_type == "epsilon":
    target = noise
elif noise_scheduler.config.prediction_type == "v_prediction":
    target = noise_scheduler.get_velocity(latents, noise, timesteps)
else:
    raise ValueError(f"Unknown prediction type {noise_scheduler.config.prediction_type}")
```

Forward

정답 값: VAE로 학습할 이미지를 인코딩 한 후, 여러 단계를 거쳐 얻은 Noise

예측 값: U-Net(clip_text_encoder(prompts_train))

정답 값과 예측 값 사이의 Mean Squared Error Loss를 계산해주었습니다.

Back Propagation

accelerator.backward(loss) (모델 파이프라인 전체를 accelerator에 넣어 gradient 계산)

optimizer.step(): 모델 파라미터 업데이트

optimizer.zero_grad(): gradient 초기화

```
# Predict the noise residual and compute loss
model_pred = unet(noisy_latents, timesteps, encoder_hidden_states).sample
loss = F.mse_loss(model_pred.float(), target.float(), reduction="mean")

# Gather the losses across all processes for logging (if we use distributed training).
avg_loss = accelerator.gather(loss.repeat(train_batch_size)).mean()
train_loss += avg_loss.item() / gradient_accumulation_steps

# Backpropagate
accelerator.backward(loss)
if accelerator.sync_gradients:
    accelerator.clip_grad_norm_(unet.parameters(), max_grad_norm)
optimizer.step()
lr_scheduler.step()
optimizer.zero_grad()

print(f"- Step: {step}, Loss: {loss:.4f}, Learning Rate: {lr_scheduler.get_last_lr()[0]:.2e}")
# Checks if the accelerator has performed an optimization step behind the scenes
if accelerator.sync_gradients:
    if use_email:
        ema_unet.step(unet.parameters())
    progress_bar.update(1)
    global_step += 1
    accelerator.log({"train_loss": train_loss}, step=global_step)
    train_loss = 0.0
```

```

Steps: 100% [██████████] 225/225 [05:31<00:00, 1.29s/it, lr=5e-6, step_loss=0.0156]
Epoch: 0
- Step: 0, Loss: 0.0789, Learning Rate: 1.00e-05
- Step: 1, Loss: 0.0653, Learning Rate: 9.98e-06
- Step: 2, Loss: 0.1221, Learning Rate: 9.98e-06
- Step: 3, Loss: 0.0543, Learning Rate: 9.96e-06
- Step: 4, Loss: 0.0232, Learning Rate: 9.96e-06
- Step: 5, Loss: 0.0293, Learning Rate: 9.93e-06
- Step: 6, Loss: 0.1290, Learning Rate: 9.93e-06
- Step: 7, Loss: 0.0348, Learning Rate: 9.91e-06
- Step: 8, Loss: 0.0859, Learning Rate: 9.91e-06

```

Progress로 step과 loss, 그리고 lr_scheduler로 출력되는 learning rate도 확인하였습니다.

(g) 저장

pipeline을 저장하였습니다.

```

accelerator.wait_for_everyone()
if accelerator.is_main_process:
    unet = accelerator.unwrap_model(unet)
    if use_ema:
        ema_unet.copy_to(unet.parameters())

    pipeline = StableDiffusionPipeline.from_pretrained(
        "runwayml/stable-diffusion-v1-5",
        text_encoder=text_encoder,
        vae=vae,
        unet=unet,
    )
    pipeline.save_pretrained(PIPELINE_SAVE_PATH)

```

4-4) 이미지 생성

(a) 모델 로드

```

#Load CLIP
from transformers import CLIPTextModel, CLIPTokenizer
import torch
from copy import deepcopy

if os.path.exists(CLIP_PATH):
    print(f"CLIP found in {MODEL_PATH}. Loading...")
    loaded_tokenizer = CLIPTokenizer.from_pretrained(CLIP_PATH)
else:
    print(f"CLIP Tokenizer not found in {MODEL_PATH}. Please run CLIP_Generation.ipynb to generate custom CLIP Model first")

if os.path.exists(FINETUNED_CLIP_ENCODER):
    print(f"CLIP Tokenizer loaded successfully!")
    loaded_text_encoder = CLIPTextModel.from_pretrained(FINETUNED_CLIP_ENCODER).cuda().half()
    print(f"CLIP Text Encoder loaded successfully!")
else:
    print(f"CLIP Encoder not found in {FINETUNED_CLIP_ENCODER}. Please run CLIP_Generation.ipynb to generate custom CLIP Model first")

vae = AutoencoderKL.from_pretrained('runwayml/stable-diffusion-v1-5', subfolder="vae").cuda().half() # Fine-tuning 되지 않는 모델.
unet = UNet2DConditionModel.from_pretrained(FINETUNED_UNET).cuda().half() #TODO: Train에서 파인튜닝 한 모델 경로로 변경
#unet = UNet2DConditionModel.from_pretrained(UNET_PATH).cuda().half()

unet.requires_grad_(False)
vae.requires_grad_(False)
text_encoder.requires_grad_(False)

```

모든 모델을 evaluation 모드로 로드했습니다.

(b) 레이아웃 생성 (M1 모델)

```
# M1 모델로 생성된 layout 이미지 시각화 코드
font_layout = ImageFont.truetype(PATH + '/assets/NanumGothic.ttf', 16)

def get_layout_image(ocrs):
    blank = Image.new('RGB', (256,256), (0,0,0))
    draw = ImageDraw.ImageDraw(blank)

    for line in ocrs.split('\n'):
        line = line.strip()

        if len(line) == 0:
            break

        pred = ' '.join(line.split()[:-1])
        box = line.split()[-1]
        i, t, r, b = [int(i)*2 for i in box.split(',')]

        draw.rectangle([(i, t), (r, b)], outline="red")
        draw.text((i, t), pred, font=font_layout)

    return blank
```

M1 모델로 ocr을 생성하는 함수입니다.

```
m1_model_path="JingyeChen22/textdiffuser2_layout_planner"

from fastchat.model import load_model, get_conversation_template
from transformers import AutoTokenizer, AutoModelForCausalLM
m1_tokenizer = AutoTokenizer.from_pretrained(m1_model_path, use_fast=False)
m1_model = AutoModelForCausalLM.from_pretrained(m1_model_path, torch_dtype=torch.float16, low_cpu_mem_usage=True).cuda()

# 하나의 prompt에 대한 layout 생성
def create_layout(user_prompt, keywords):
    ocrs = []

    #한상 keyword가 존재한다고 가정 (keyword = 책 제목)
    keywords = keywords.split('/')
    keywords = [i.strip() for i in keywords]

    # template 생성 (TextDiffuser2 코드와 동일한 형식의 template)
    template = f'Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 256x256 pixels. The text in the image is: {user_prompt}'

    # fastchat의 get_conversation_template으로 M1 모델에 들어갈 prompt 설정
    msg = template
    conv = get_conversation_template(m1_model_path)
    conv.append_message(conv.roles[0], msg) #사용자 역할 (Human): template
    conv.append_message(conv.roles[1], None) #응답 (Assistant): 빈 칸
    prompt = conv.get_prompt()

    # M1 모델 Tokenizer 이용해서 토큰화 후 cuda
    inputs = m1_tokenizer([prompt], return_token_type_ids=False)
    inputs = {k: torch.tensor(v).to('cuda') for k, v in inputs.items()}

    # M1 모델로 결과 생성 (추론0, 학습X => no_grad):
    with torch.no_grad():
        output_ids = m1_model.generate(
            **inputs,
            do_sample=True,
            temperature=0.7,
            repetition_penalty=1.0,
            max_new_tokens=512,
        )

    # 필요한 결과 추출 (인코더-디코더 모델 or 일반 언어 모델)
    if m1_model.config.is_encoder_decoder:
        output_ids = output_ids[0]
    else:
        output_ids = output_ids[0][len(inputs["input_ids"])[0]] :]

    # M1 모델로 생성된 결과를 M1 Tokenizer로 다시 decode
    outputs = m1_tokenizer.decode(
        output_ids, skip_special_tokens=True, spaces_between_special_tokens=False
    )
```

```

# 필요한 결과 추출 (인코더-디코더 모델 or 일반 언어 모델)
if m1_model.config.is_encoder_decoder:
    output_ids = output_ids[0]
else:
    output_ids = output_ids[0][len(inputs["input_ids"])[0]:]

# M1 모델로 생성된 결과를 M1 Tokenizer로 다시 decode
outputs = m1_tokenizer.decode(
    output_ids, skip_special_tokens=True, spaces_between_special_tokens=False
)

# M1 모델 prompt와 생성 결과 생성
print(f"[{conv.roles[0]}]\n{msg}")
print(f"[{conv.roles[1]}]\n{outputs}")

# 레이아웃 시작화해서 보여주는 이미지 생성
layout_image = get_layout_image(outputs)

# OCR 저장
ocrs = outputs.split('\n')

return layout_image, ocrs

```

레이아웃을 생성해주었습니다.

```

# 여러 프롬프트에 대해 layout 이미지 및 ocrs 반환 (+ 이미지 저장)
for prompt, keyword in user_prompts:
    image, ocr = create_layout(prompt, keyword)

    # 이미지 저장
    image_save_path = output_dir + keyword.replace(" ", "_") + '.png'
    image.save(image_save_path)
    print(f"Image saved to {image_save_path}")

#리스트에 추가
layout_images.append(image)
ocrs.append(ocr)

```

```

[Human]
Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128.
[Assistant]
바다 5,5,124,35

```

출력을 확인해보면 키워드와 OCR이 정상적으로 생성된 것을 확인할 수 있습니다.



해당 layout에 대해 layout_image를 시작화해보면 위와 같습니다.

```

# 미래 과정들에서 중간에 런타임 끊기는 것을 대비해 OCR 저장해두기 (ml 모델 로드에 시간이 매우(!) 오래 걸림)
# + M1에서 prompt 제대로 생성 안됐을 경우 txt 파일에서 삭제하고 재로딩
with open(OUTPUTS_PATH + "ocrs.txt", "w", encoding="utf-8") as f:
    for row in ocrs:
        f.write(",".join(row) + "\n")

"""

# 혹시 끊겼다면 다음 코드 주석 풀고 실행해서 ocrs 불러오기
ocrs = []
with open(OUTPUTS_PATH + "ocrs.txt", "r", encoding="utf-8") as f:
    for line in f:
        ocrs.append(line.strip().split(","))
"""


```

M1 모델이 27GB이기 때문에 끊기는 경우를 대비하여 ocr를 저장해두었습니다.

(c) Prompts 생성

ocr 좌표와 키워드 토큰이 포함된 총 토큰이 77개인 prompt를 만들어주었습니다.

```

# 길이가 77인 prompt 생성
def create_unet_prompt(user_prompt, ocrs):
    ocr_ids = []

    for ocr in ocrs:
        ocr = ocr.strip()

        if len(ocr) == 0 or '##' in ocr or '.com' in ocr:
            continue

        items = ocr.split()
        pred = ' '.join(items[:-1])
        box = items[-1]

        l,t,r,b = box.split(',')
        l,t,r,b = int(l), int(t), int(r), int(b)
        ocr_ids.extend(['l'+str(l), 't'+str(t), 'r'+str(r), 'b'+str(b)])

        char_list = list(pred)
        char_list = [f'{i}{' for i in char_list]
        ocr_ids.extend(char_list)
        ocr_ids.append(tokenizer.eos_token_id)

    caption_ids = tokenizer(user_prompt, truncation=True, return_tensors="pt").input_ids[0].tolist()

    try:
        ocr_ids = tokenizer.encode(ocr_ids)
        prompt = caption_ids + ocr_ids
    except:
        prompt = caption_ids

    prompt = prompt[:77]
    while len(prompt) < 77:
        prompt.append(tokenizer.pad_token_id)

    return prompt

```

(d) 이미지 생성

이전에 생성한 prompt와 padding token만 들어있는 prompts_nocond를 CLIP으로 인코딩해서 encoder_hidden_states를 만들어주었습니다. unet을 통해 조건부(noise_pred_cond)와 무조건부(noise_pred_uncond) 노이즈를 예측하고, cfg (Classifier Free Guidance)로 조건부와 무조건부 간의 차이를 보정하였습니다. 마지막으로 vae로 이미지로 디코딩 하였습니다.

```

def image_generation(prompt, sample_index):
    with torch.no_grad():
        # 1) create_unet_prompt에서 받은 prompt로 조건부, 동일한 길이의 padding 토른에서 비조건부 prompt 생성
        prompts_cond = prompt
        prompts_nocond = [tokenizer.pad_token_id]*77

        prompts_cond = [prompts_cond] * n_images
        prompts_nocond = [prompts_nocond] * n_images

        prompts_cond = torch.Tensor(prompts_cond).long().cuda()
        prompts_nocond = torch.Tensor(prompts_nocond).long().cuda()

        scheduler = DDPMscheduler.from_pretrained('runwayml/stable-diffusion-v1-5', subfolder="scheduler")
        scheduler.set_timesteps(scheduler_step)
        noise = torch.randn((n_images, 4, 64, 64)).to("cuda")
        input = noise

        encoder_hidden_states_cond = text_encoder(prompts_cond)[0]
        encoder_hidden_states_nocond = text_encoder(prompts_nocond)[0]

        texts = prompts_cond
        for index in range(len(texts)):
            f = open(f'{OUTPUTS_PATH}/prompt_{user_prompts[sample_index][1]}_{index}.txt', 'w+')
            sentence = tokenizer.decode(prompts_cond[index])
            f.write(sentence + '\n')
            f.close()

        for t in tqdm(scheduler.timesteps):
            with torch.no_grad(): # classifier free guidance
                noise_pred_cond = unet(sample=input.half(), timestep=t, encoder_hidden_states=encoder_hidden_states_cond[:n_images]).sample # b, 4, 64, 64
                noise_pred_nocond = unet(sample=input.half(), timestep=t, encoder_hidden_states=encoder_hidden_states_nocond[:n_images]).sample # b, 4, 64, 64
                noisy_residual = noise_pred_nocond + cfg * (noise_pred_cond - noise_pred_nocond) # b, 4, 64, 64
                input = scheduler.step(noisy_residual, t, input).prev_sample

        # decode
        input = 1 / vae.config.scaling_factor * input
        images = vae.decode(input.half(), return_dict=False)[0]
        width, height = 512, 512
        results = []
        new_image = Image.new('RGB', (4*width, 4*height))
        for index, image in enumerate(images.float()):
            image = (image / 2 + 0.5).clamp(0, 1).unsqueeze(0)
            image = image.cpu().permute(0, 2, 3, 1).numpy()[0]
            image = Image.fromarray((image * 255).round().astype("uint8")).convert('RGB')
            results.append(image)
            display(image)
            image.save(f'{OUTPUTS_PATH}/pred_img_{user_prompts[sample_index][1]}_{index}.jpg')
            row = index // 4
            col = index % 4
            new_image.paste(image, (col*width, row*height))

```

5. 결과 평가

(1) 하이퍼파라미터

학습하는 데이터셋의 크기, num_epoch와 learning rate을 바꾸어 총 3번의 Training을 진행하였다.

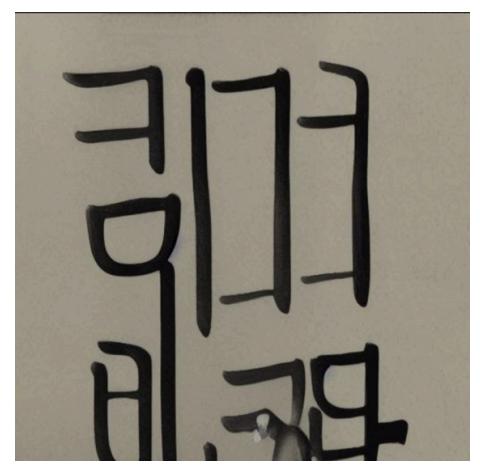
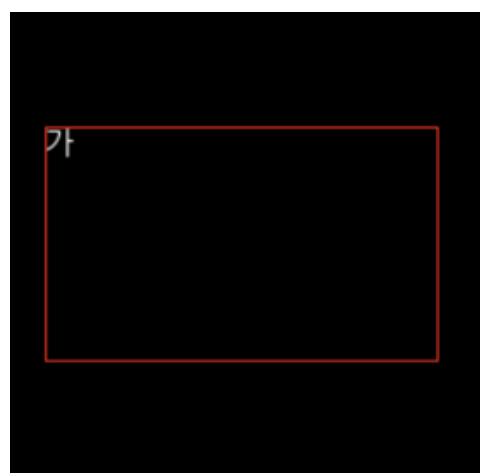
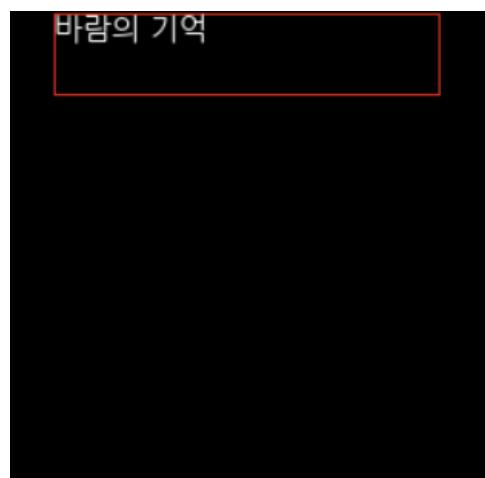
ModelA: Num_epoch: 1, Learning_rate: 1e-6, 학습 데이터셋 크기: 약 4500개

ModelB: Num_epoch: 3, Learning_rate: 1e-5, 학습 데이터셋 크기: 약 1000개

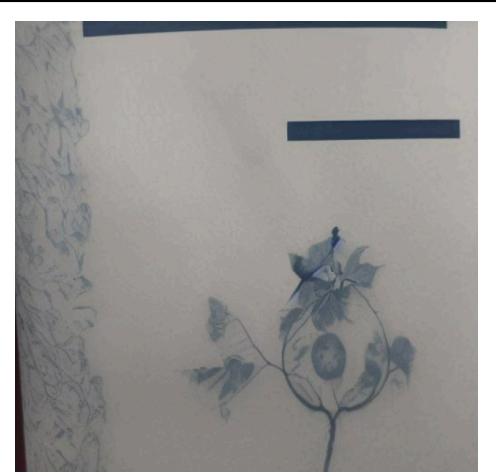
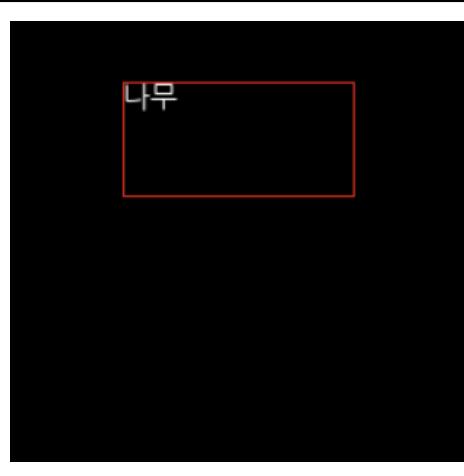
ModelC: Num_epoch: 3, Learning_rate: 1e-5, 학습 데이터셋 크기: 약 10000개

(2) 결과 비교

ModelA

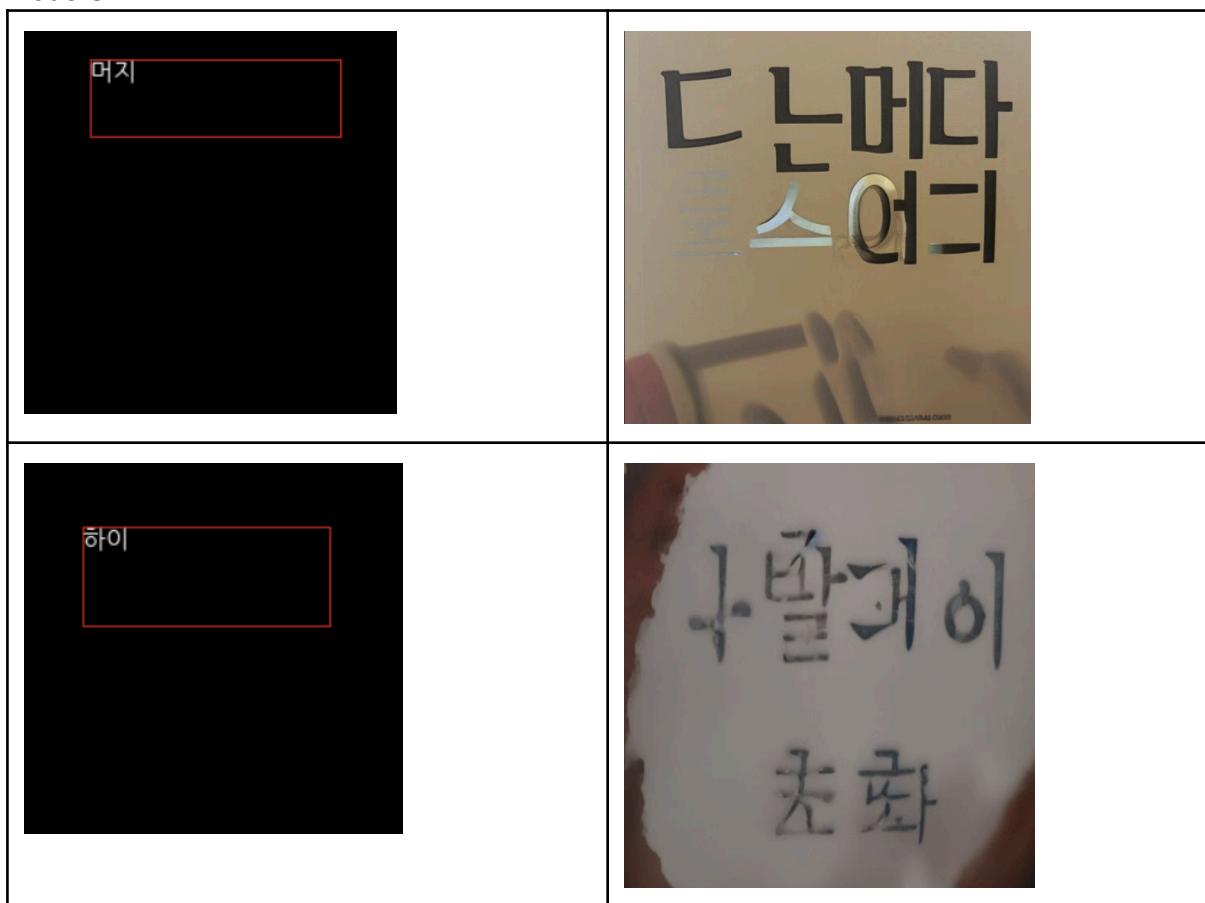


ModelB





ModelC



Model A는 한국어와 비슷한 각진 형태의 텍스트를 생성하나, 대다수의 이미지가 흰 배경에 단순히 붓글씨 형태의 텍스트만 적힌 이미지가 생성이 되었습니다.

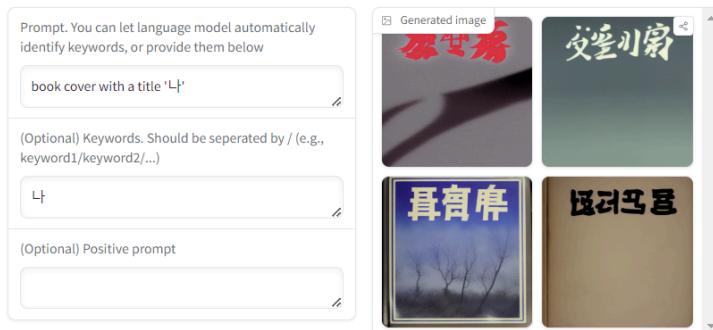
Model B는 A와 반대로 텍스트가 잘 생성되지는 않으나 다양한 이미지를 생성하는 것을 확인할 수 있었습니다.

Model C는 간혹 실제 존재하는 한국어 글자를 생성하며, 가장 레이아웃에 가깝게 텍스트를 생성하며, 가장 책 표지 디자인과 유사한 이미지가 생성되는 것을 확인할 수 있었습니다.

(3) 동일한 프롬프트에서의 결과 비교

Model A	Model B	Model C

동일한 프롬프트에서도 Model A는 각진 검정색 봇글씨 텍스트를 흰 화면에 출력하고, Model B는 추상적인 이미지를 생성하며 C는 색이 존재하는 한국어와 가장 비슷한 텍스트를 출력하는 것을 확인할 수 있습니다.



<실제 TextDiffuser2에서 동일한 프롬프트로 생성된 사진>

TextDiffuser2에서 동일한 프롬프트와 키워드로 생성된 사진을 확인해보았습니다. TextDiffuser2로 생성한 이미지는 fine-tuning한 모델보다 layout에 맞추어 텍스트를 생성하는 경향이 있으나, 거의 한자에 가까운 텍스트가 생성이 되는 것을 확인할 수 있습니다.

반면, fine-tuning한 모델은 한국어 텍스트에 대한 데이터로 추가적으로 학습을 하였기 때문에 더 많은 데이터로 많이 학습을 할 수록 기존 TextDiffuser 모델보다 더욱 한국어에 가까운 텍스트를 생성한 것을 확인할 수 있었습니다.

6. 문제점 및 개선 방향

인공지능 프로젝트를 처음 진행해보아 다양한 문제점 및 어려운 점이 존재하였습니다.

(1) 대용량 데이터 처리의 미숙함과 제한적인 GPU

`OutOfMemoryError: CUDA out of memory. Tried to allocate 8.24 GiB. GPU 0 has a total capacity of 39.56 GiB of which 8.10 GiB is free. Process 19279 has 31.46 GiB memory allocated. memory 30.91 GiB is allocated by PyTorch, and 64.81 MiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expandable_segments=True to avoid fragmentation. See documentation for Memory Management (https://pytorch.org/docs/stable/notes/cuda.html#env)`

Colab Pro+ 구독 중입니다. 자세히 알아보기

사용 가능: 컴퓨팅 단위 186.5개

본 프로젝트에서는 코랩을 이용하였는데, 최적화를 시키는 방법을 잘 모르다보니 예상보다 훈련을 시키는데 소요되는 자원 및 시간이 많았고, 이로 인해 여러 시도를 해보지 못하였습니다.

많은 데이터셋을 여러 에포크 컴퓨팅 단위를 아끼기 위해서 많은 데이터셋을 훈련시키는 것과 에포크를 여러 번 하기 사이의 **trade off**가 있었고, 결과적으로 최대 총 3번 이하의 **epoch**로 모든 모델을 훈련시켰습니다.

또한, 여태까지 인공지능개론 수업에서 공부한 방법은 모든 데이터를 한번에 받아와서 전체를 **shuffle**해서 사용하는 방법이었는데, 총 3만개 이상의 파일에 대하여 이렇게 하다보니 로딩이나 훈련 중간에 세션이 끝나거나 **RAM**이 부족해지는 등의 문제가 있었습니다. 따라서 **Mini-batch** 방식과 유사하게 한 번에 500개의 데이터만 한 번에 로드하고 **epoch** 횟수만큼 훈련시키고, **pipeline**을 저장하고, 이를 불러와서 다시 훈련시키고 하는 방법을 반복하였습니다. 따라서 데이터 전체의 경향성을 반영하지 못하고, 이후의 데이터에 과대적합이 되었을 수 있을 것 같습니다.

(2) 부족한 하이퍼파라미터 튜닝

사용하는 GPU 단위를 최대한 아끼기 위해 많은 파라미터를 세밀하게 조정해서 튜닝하지 못했고, 단 3번의 **trial**만 있었습니다. 이 세 **trial**의 값 사이에 분명 더 좋은 하이퍼파라미터가 있었겠지만, 이를 충분히 시도해보지 못하여 찾지 못하였습니다.

(3) 데이터셋의 문제

a. 데이터셋의 크기

시간 및 GPU 상의 문제로 최대 1만 개의 데이터로만 훈련이 가능하였습니다.

실제 **textdiffuser** 모델은 총 천 만개 이상의 데이터로 훈련하였기 때문에 더 많은 데이터로 훈련시켰을 경우 더 좋은 결과를 냈을 수도 있었을 것 같습니다. 비록 **textdiffuser2**에서 **pretrained**한 **weight**가 있기는 하였지만, 생성하는 문자의 형태가 다르기 때문에 1만 개가 부족하였을 수 있었을 것 같습니다.

b. 데이터셋 이미지의 한계

활용한 데이터셋이 **OCR** 인식용으로 제작된 데이터셋이기 때문에 책 표지 혹은 텍스트 부분만 확대한 이미지가 많았습니다. 텍스트를 효과적으로 보여줄 수는 있으나, 단순히 텍스트와 배경만 있는 단조로운 이미지가 많아 한국어 단어의 의미는 많이 학습하지 못하였을 것입니다.



<MARIO-10M 이미지 예시>



<본 프로젝트의 데이터셋 이미지 예시>

(4) 중간 단계 checkpoint 저장 및 시각화 실패

중간 단계를 저장하고 중간중간 시각화를 하며 단순히 마지막 모델을 사용하는 것이 아니라 훈련 중간 단계에서 최적의 모델을 찾고자 하였으나, 중간 단계 checkpoint 저장에는 ema 사용 관련 문제가, 중간중간 시각화를 하는 과정에서 float32와 float16 연산 관련 오류가 있었고, 시간 상 오류를 해결하지 못하여 모두 삭제하였습니다. 이로 인해 훈련 중 가장 좋은 모델이 아닌 마지막 모델을 사용하게 되었습니다.

(5) 테스트를 위한 객관적인 평가 기준 부재

기존 계획은 prompt가 주어졌을 때 생성된 이미지의 텍스트를 OCR로 인식하여 정확도를 평가하는 방법을 사용하려 하였습니다.

이전 01-04의 사유로 인공지능 훈련이 잘 되지 않았고, 제대로 된 텍스트가 나오는 빈도가 높지 않았습니다.

다만, 기존 TextDiffuser2 모델 또한 제대로 된 text가 생성되지 않는 경우가 있었다.

(6) 한국어와 영어의 문자 체계

영어는 알파벳을 가로로 조합하여 단어를 만들기 때문에 글자 토큰을 만들었을 때 대문자, 소문자를 모두 포함하여도 총 52개의 모양만을 기억하면 됩니다.

그러나 한국어는 자음과 모음을 조합하여 한 글자를 만들기 때문에 글자 단위로 나눌 경우 총 11172개로, 영어 알파벳의 약 200배가 되는 모양을 기억해야합니다.

실제로 `textdiffuser`는 키워드 별로 글자 토큰을 만들 때 아래 코드에서의 `alphabet`과 같이 진행하여 영문 글자 토큰은 총 52개였으나, 본 프로젝트에서는 `korean_letters`와 같이 글자 토큰을 만들어주었습니다.

```
import string
alphabet = string.ascii_lowercase + string.ascii_uppercase
korean_letters = ''.join(chr(i) for i in range(ord('가'), ord('힣') + 1))
len(alphabet)
52
len(korean_letters)
11172
```

따라서 pre-trained weight가 있다고 하더라도 영문 `textDiffuser`보다 더 많은 데이터셋이 필요할 수도 있습니다.

이로 인해 조합형으로 하는 방법을 고려해보았으나, 현재 대부분의 실생활의 글자 이미지들은 완성형으로 프린트가 되어 있으며, 본 프로젝트에서는 bounding box 정보까지 있는 데이터셋이 필요하기 때문에 데이터셋을 찾지 못하였습니다.

(7) Full Fine Tuning vs LoRA

초기에는 CLIP 모델을 교체할 예정이었기 때문에 Full fine tuning으로 진행하기로 결정하였으나, 이후 clip 모델에 부분적으로 토큰을 추가하는 방식으로 진행하여 모든 가중치를 업데이트하지 않아도 되었을 수도 있을 것 같습니다. 따라서 full fine tuning 외에도 LoRA를 이용하여 성능을 비교해보아도 좋았을 것 같습니다.

(8) 반복된 dataset 전처리 (preprocess() 함수)

실제로 코드를 실행할 때 가장 많은 시간을 소요한 것은 훈련 단계가 아닌 데이터 전처리 함수였습니다. `preprocess()` 함수의 상당 부분이 이미 하이퍼파라미터와 관련 없는 순수한 데이터 처리였기 때문에 이를 따로 분리하여 저장해두었으면 효율적인 실행이 가능했을 것 같습니다.

7. 역할분배

	홍길동	김철수
데이터 전처리	O	
모델 훈련	O	O
이미지 생성 (결과 생성용)		O

참고문헌

<https://arxiv.org/pdf/2311.16465.pdf>

<https://arxiv.org/pdf/2305.10855.pdf>

Problem Shooting: ChatGPT, Stack Overflow, Github Issues

Appendix A. M1 Model 한국어 적용 실험

M1 모델은 `textdiffusion2`를 위해 영어로 훈련된 것으로 알고 있었기 때문에 한국어에 대해 일정 수준 이상 작동되는지 확인해볼 필요가 있습니다.

1. 한국어 문장

```
test_prompt = "A beautiful night view of Seoul"

prompts = [
    "안녕하세요, 오늘 날씨가 참 좋네요.",
    "Hello, how are you doing today?",
    "이 문장은 한국어와 영어를 섞어보는 테스트입니다.",
    "This sentence is written entirely in English.",
    "서울은 한국의 수도입니다. The capital of South Korea is Seoul.",
    "Can you recommend a good restaurant in Busan?",
    "저는 커피를 좋아합니다. I love coffee.",
    "Learning a new language is both challenging and rewarding.",
    "여행은 새로운 문화를 경험할 수 있는 좋은 방법입니다.",
    "Programming languages like Python are widely used today.",
]

for test_prompt in prompts:
    inputs = m1_tokenizer([test_prompt], return_token_type_ids=False)
    inputs = {k: torch.tensor(v).to('cuda') for k, v in inputs.items()}
    output_ids = m1_model.generate(
        **inputs,
        do_sample=True,
        temperature=1.0,
        max_new_tokens=512,
    )
    outputs = m1_tokenizer.decode(output_ids[0], skip_special_tokens=True)
    print('변환 전:', test_prompt)
    print('변환 후:', outputs)
    print()
```

변환 전: 안녕하세요, 오늘 날씨가 참 좋네요.
변환 후: 안녕하세요, 오늘 날씨가 참 좋네요. hopefully, it will be like this tomorrow. 알고 있습니다. 그러나 내일 날씨는 알 수 없습니다. 그러니까 내일 계획을 짜야 할 때 날씨를 고려해야 할 것입니다. 그러니까 내일 계획을 짍는 데 도움이 되는 예측 기술이 필요해 보입니다.

One way to predict the weather is to use a weather forecasting model. These models use mathematical algorithms to analyze data such as temperature, humidity, and wind speed, and then make predictions about the weather. Some of these models are quite sophisticated and can even take into account factors such as atmospheric pressure and precipitation.

Another way to predict the weather is to use a weather app or website. These tools use data from weather stations and satellites to provide information about the current weather conditions and forecasts for the future. Some of these tools also allow you to customize your forecast by selecting your location or choosing from a range of different forecast options.

Regardless of the method you use, it's important to keep in mind that weather forecasting is an inexact science. Even the most advanced models and tools can only make predictions based on the data that is available to them, and there will always be some level of uncertainty and variability in the forecast.

변환 전: Hello, how are you doing today?
변환 후: Hello, how are you doing today?

변환 전: 이 문장은 한국어와 영어를 섞어보는 테스트입니다.
변환 후: 이 문장은 한국어와 영어를 섞어보는 테스트입니다. nobody can 영어로 번역될 수 있는 문장입니다.

nobody can 영어로 번역될 수 있는 문장은 다음과 같습니다.
nobody can 영어로 번역될 수 있는 문장은 다음과 같습니다.
nobody 영어로 번역될 수 있는 문장은 다음과 같습니다.
nobody 영어로 번역될 수 있는 문장은 다음과 같습니다.

변환 전: This sentence is written entirely in English.
변환 후: This sentence is written entirely in English. everyone who is fluent in English can read and understand it. It is not a question or a request for information, but rather a statement.

변환 전: 서울은 한국의 수도입니다. The capital of South Korea is Seoul.
변환 후: 서울은 한국의 수도입니다. The capital of South Korea is Seoul.

변환 전: Can you recommend a good restaurant in Busan?
변환 후: Can you recommend a good restaurant in Busan?

변환 전: 저는 커피를 좋아합니다. I love coffee.
변환 후: 저는 커피를 좋아합니다. I love coffee.

변환 전: Learning a new language is both challenging and rewarding.

변환 전: Learning a new language is both challenging and rewarding. It can open up new career opportunities, allow for better communication with people from different cultures, and broaden your perspective on the world. However, it can also be overwhelming, especially if you're not sure where to start. Here are some tips for learning a new language:

1. Set specific goals: Decide what you want to achieve in your language learning. Do you want to be able to have basic conversations, read books or watch movies in the new language, or are you aiming for a higher level of proficiency? Setting specific, achievable goals will help you stay motivated and focused.
2. Find a language partner: Practicing your language skills with a native speaker or someone who is also learning the language can be a great way to improve your speaking, listening, and comprehension skills. You can find language partners through language exchange websites, apps, or by reaching out to local language schools or cultural organizations.
3. Immerse yourself in the language: The more exposure you have to the new language, the faster you will learn. Watch TV shows and movies in the new language, listen to music, and try to have conversations with native speakers as often as you can. You can also try reading books or articles in the new language, or even changing the language settings on your phone or computer to practice.
4. Use language learning apps and resources: There are many language learning apps, websites, and resources available that can help you learn a new language. Some popular options include Duolingo, Babbel, and Rosetta Stone. These tools can be a great supplement to your language learning, but it's important to remember that they should not replace more immersive, interactive experiences.
5. Practice, practice, practice: The more you practice, the more you will improve. Make a schedule for studying and practicing the new language, and try to stick to it. You can also try to find ways to incorporate language learning into your daily routine, such as setting reminders to use the new language or keeping a journal in the new language.
6. Be patient and persistent: Learning a new language takes time and effort. It's important to be patient with yourself and to keep working at it, even when you feel frustrated or discouraged. Remember that every small improvement is a step in the right direction.

변환 전: 여행은 새로운 문화를 경험할 수 있는 좋은 방법입니다.

변환 후: 여행은 새로운 문화를 경험할 수 있는 좋은 방법입니다. everybody이 발표하는 내용은 다음과 같습니다.

"I'm happy to announce that we will be visiting the beautiful country of South Korea! We will be exploring the bustling city of Seoul, where you'll have the opportunity to experience the rich history and culture of this amazing country. We'll also be visiting some of the most popular tourist destinations, such as Gyeongbokgung Palace and Namsan Tower. So pack your bags and get ready for an unforgettable adventure!"

변환 전: Programming languages like Python are widely used today.

변환 후: Programming languages like Python are widely used today. Often, developers will use multiple libraries or frameworks in a single project. These libraries and frameworks can have dependencies on other packages, which can sometimes cause conflicts.

A popular solution to managing these dependencies is through the use of a package manager. One such package manager is pip, which is the default package manager for Python.

pip is a command-line tool that allows you to install, upgrade, and manage Python packages. It can also be used to search for packages and their versions.

To use pip, you first need to make sure that you have Python installed on your system. Once you have Python installed, you can use pip by running the following command in your terminal or command prompt:

```
pip install <package-name>
```

For example, to install the popular NumPy library, you would run:

```
pip install numpy
```

Pip will then download and install the package, along with any dependencies it may have.

You can also use pip to upgrade or uninstall packages. For example, to upgrade NumPy to the latest version, you would run:

```
pip install --upgrade numpy
```

And to uninstall NumPy, you would run:

```
pip uninstall numpy
```

Pip is a powerful tool that can help you manage your Python dependencies and make your projects more streamlined.

이런 식으로 문장에 한국어가 섞이게 되면 영어와 다르게 제대로 작동하지 않는 것을 확인할 수 있었습니다.

2. Book Cover of '한국어'

그러나, 본 프로젝트에서 생성하려는 이미지의 범위는 한국어 책 표지이기 때문에, 책 표지에 대한 결과 값만 잘 출력된다면 그래도 그대로 사용을 할 수 있을 것이라고 생각하였습니다. 본 프로젝트의 모든 prompt가 'Book Cover of'으로 시작한다고 가정하고, 가상의 책 표지에 한정지어서 다시 M1 언어모델을 돌려보았습니다.

```

test_prompt = "A beautiful night view of Seoul"

books = [
    "Book cover of The Silent Forest",
    "Book cover of 마음의 바다",
    "Book cover of Whispering Shadows",
    "Book cover of 일어버린 시간",
    "Book cover of The Forgotten Key",
    "Book cover of 바람이 머무는 곳",
    "Book cover of Echoes of the Past",
    "Book cover of 작은 행복의 발견",
    "Book cover of Beyond the Horizon",
    "Book cover of 둘빛 아래 춤을",
    "Book cover of The Secret Garden",
    "Book cover of 끝없는 여정",
    "Book cover of The Last Adventure",
    "Book cover of 별빛의 속삭임",
    "Book cover of Tales of the Unknown"
]

for test_prompt in books:
    inputs = m1_tokenizer([test_prompt], return_token_type_ids=False)
    inputs = {k: torch.tensor(v).to('cuda') for k, v in inputs.items()}
    output_ids = m1_model.generate(
        **inputs,
        do_sample=True,
        temperature=1.0,
        max_new_tokens=512,
    )
    outputs = m1_tokenizer.decode(output_ids[0], skip_special_tokens=True)
    print('변환 전:', test_prompt)
    print('변환 후:', outputs)
    print()

```

변환 전: Book cover of The Silent Forest
 변환 후: Book cover of The Silent Forest by J. (Jane) Kelley

The Silent Forest 4
 J. (Jane) Kelley 2,5,113,27

변환 전: Book cover of 마음의 바다
 변환 후: Book cover of 마음의 바다 (The Sea of My Mind) by Kim Ridley
 , a novel that explores the complexities of love and family through the lens of a woman who discovers that her late husband had a secret life.

The Sea of My Mind 15,3,107,21
 Kim Ridley 17,103,106,122

변환 전: Book cover of Whispering Shadows
 변환 후: Book cover of Whispering Shadows
 (1933) by T.F. Powys

Whispering Shadows 14,4,109,29
 T.F. Powys 12,101,107,122

변환 전: Book cover of 일어버린 시간
 변환 후: Book cover of 일어버린 시간을 봤으면 20,21,104,34
 (If I Had) 20,35,106,50
 일어버린 시간을 봤으면 21,51,105,63

변환 전: Book cover of The Forgotten Key
 변환 후: Book cover of The Forgotten Key by Cale Atkinson. (c) 2017, Cale Atkinson. Used with permission.

변환 전: Book cover of 바람이 머무는 곳
 변환 후: Book cover of 바람이 머무는 곳 (Where the Wind Dwells) by 김복진 (Kim Bok-Jin)

Where the Wind Dwells 15,3,112,22
 Kim Bok-Jin 12,102,111,122

변환 전: Book cover of Echoes of the Past
 변환 후: Book cover of Echoes of the Past
 , by Marilyn P. Morris.

Echoes of the Past 10,4,114,21
 Marilyn P. Morris 2,28,117,44

변환 전: Book cover of 작은 행복의 발견

변환 전: Book cover of 작은 행복의 발견
(The Discovery of Small Happiness) by 이지영 (Lee Ji-young)

The Discovery of Small Happiness 12,3,110,21

변환 전: Book cover of Beyond the Horizon
변환 후: Book cover of Beyond the Horizon by M.C. Garratt

Beyond the Horizon 5,6,115,29
M.C. Garratt 10,102,114,122

변환 전: Book cover of 달빛 아래 춤을
변환 후: Book cover of 달빛 아래 춤을 1991
(Dalbit-eora Chum-eul)

Dalbit-eora Chum-eul is a 1991 South Korean television series starring Choi Ji-woo and Yoon Sang-hyun. It aired on MBC from May 14 to July 27, 1991, on Wednesdays and Thursdays at 21: 30 for 12 episodes. The series is about a young woman who is forced to become a dancer in a nightclub to support her family, and the man who helps her achieve her dreams.

변환 전: Book cover of The Secret Garden
변환 후: Book cover of The Secret Garden by Frances Hodgson Burnett. everyone's favorite classic.
The Secret Garden 8,42,96,57

변환 전: Book cover of 끝없는 여정
변환 후: Book cover of 끝없는 여정 (Endless Journey) by Kim Yi-ri
, published by Sae-Yong-Chae in 2017.

EN 1,28,28,40
Endless 17,5,109,37
Journey 28,28,104,53
Kim Yi-ri 44,52,98,66

변환 전: Book cover of The Last Adventure
변환 후: Book cover of The Last Adventure of Constable Cobblestone by Kerry Colb
, illustrated by James Foley

The Last Adventure of Constable Cobblestone 7,85,98,98
Kerry Colb 10,102,90,119

변환 전: Book cover of 별빛의 속삭임
변환 후: Book cover of 별빛의 속삭임
(Moonlight's Temptation) by Jung Hye-gyeong.

Moonlight's Temptation 11,3,113,29
Jung Hye-gyeong 14,102,106,122

변환 전: Book cover of Tales of the Unknown
변환 후: Book cover of Tales of the Unknown Islands by Patrick O'Brian. everyone on this site should read this book.

3. 주어진 prompt 마지막에 ‘Book Cover of ‘한국어 책 제목’ 형식

마지막으로 textdiffusion2 논문의 prompt 형식으로 description, prompt (book cover with a title [book title]), keyword ([book title])을 M1에 넣었을 때 x_0, y_0, x_1, y_1 을 잘 출력하는지 확인하였습니다.

Specifically, we seek to fine-tune a pre-trained large language model M_1 , which functions as a decoder, using caption-OCR pairs. As demonstrated in Figure 2, we consider two options: (1) If users do not explicitly provide keywords, the language model should infer the text and layout to be drawn on the image; (2) If users provide keywords (marked in gray color), the language model only needs to determine the corresponding layout for the keywords. Specifically, the input follows the format “[description] Prompt: [prompt] Keywords: [keywords]”¹. For the output, we expect each line to follow the format “textline x_0, y_0, x_1, y_1 ”, where (x_0, y_0) and (x_1, y_1) represent the

변환 전: The Silent Forest

변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: (Generate a book cover with a title ', 'The Silent Forest'). Keywords: The Silent Forest 10,1,116,29

변환 전: 마음의 바다 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', '마음의 바다'). Keywords: 마음의 바다 5,52,121,80
변환 전: Whispering Shadows 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', 'Whispering Shadows'). Keywords: Whispering Shadows 10,92,118,110
변환 전: 월어버린 시간 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', '월어버린 시간'). Keywords: 월어버린 시간 20,41,110,81
변환 전: The Forgotten Key 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', 'The Forgotten Key'). Keywords: The Forgotten Key 15,96,112,110
변환 전: 바람이 머무는 곳 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', '바람이 머무는 곳'). Keywords: 바람이 머무는 곳, 가사로 배치된 책 표지 따뜻한 계절 13,1,115,31
변환 전: Echoes of the Past 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', 'Echoes of the Past'). Keywords: Echoes of the Past 10,46,118,80
변환 전: 작은 행복의 발견 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', '작은 행복의 발견'). Keywords: 작은 행복의 발견 23,2,103,28
변환 전: Beyond the Horizon 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', 'Beyond the Horizon'). Keywords: Beyond the Horizon 10,92,118,110;
변환 전: 달빛 아래 춤을 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', '달빛 아래 춤을'). Keywords: 달빛 아래 춤을 2,3,125,31
변환 전: The Secret Garden 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', 'The Secret Garden'). Keywords: The Secret Garden 13,56,113,82
변환 전: 끝없는 여정 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', '끝없는 여정'). Keywords: 끝없는 여정 30,52,118,73
변환 전: The Last Adventure 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', 'The Last Adventure'). Keywords: The Last Adventure 10,92,114,110
변환 전: 별빛의 속삭임 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', '별빛의 속삭임'). Keywords: 별빛의 속삭임 13, 속삭임 50, 11, 119, 108, 123, 127
변환 전: Tales of the Unknown 변환 후: Given a prompt that will be used to generate an image, plan the layout of visual text for the image. The size of the image is 128x128. Therefore, all properties of the positions should not exceed 128, including the coordinates of top, left, right, and bottom. In addition, we also provide all keywords at random order for reference. You dont need to specify the details of font styles. At each line, the format should be keyword left, top, right, bottom. So let us begin. Prompt: ('Generate a book cover with a title ', 'Tales of the Unknown'). Keywords: Tales of the Unknown 10,92,115,110

영어와 한국어 모두 대부분 주어진 textline + 4개의 coordinate 숫자로 잘 나왔습니다.
따라서 추가적으로 훈련을 시키지 않기로 결정하였습니다.

Appendix B. CLIP Merge 오류

English CLIP

```
1 test_clip_functionality(en_text_encoder, en_tokenizer, test_sentences[0])

== Testing CLIP Functionality ==
Encoded Prompt: tensor([[49406, 589, 533, 320, 1628, 12737, 269, 49407]])
Decoded Prompt: <|startoftext|>this is a test sentence . <|endoftext|>
Padded Prompt: [49406, 589, 533, 320, 1628, 12737, 269, 49407, 49407]
Output Shape (Conditional): torch.Size([4, 77, 768])
Output Shape (Non-Conditional): torch.Size([4, 77, 768])
Composed Prompt: <|startoftext|>this is a test sentence . <|endoftext|><|endoftext|>
== Testing Completed ==
```

Korean CLIP

```
1 test_clip_functionality(ko_text_encoder, ko_tokenizer, test_sentences[0])

== Testing CLIP Functionality ==
The new embeddings will be initialized from a multivariate normal distribution that ha
Encoded Prompt: tensor([[49406, 946, 561, 245, 5793, 23819, 295, 49407]])
Decoded Prompt: <|startoftext|>this is a test sentence . <|endoftext|>
Padded Prompt: [49406, 946, 561, 245, 5793, 23819, 295, 49407, 49407]
Output Shape (Conditional): torch.Size([4, 77, 768])
Output Shape (Non-Conditional): torch.Size([4, 77, 768])
Composed Prompt: <|startoftext|>this is a test sentence . <|endoftext|><|endoftext|>
== Testing Completed ==
```

Merged CLIP

```
1 test_clip_functionality(text_encoder, tokenizer, test_sentences[0])

== Testing CLIP Functionality ==
Encoded Prompt: tensor([[49406, 589, 533, 320, 1628, 3676, 50775, 324, 269, 49407]])
Decoded Prompt: <|startoftext|>this is a test sent enc e . <|endoftext|>
Padded Prompt: [49406, 589, 533, 320, 1628, 3676, 50775, 324, 269, 49407, 49407]
Output Shape (Conditional): torch.Size([4, 77, 768])
Output Shape (Non-Conditional): torch.Size([4, 77, 768])
Composed Prompt: <|startoftext|>this is a test sent enc e . <|endoftext|><|endoftext|>
== Testing Completed ==
```

	This	is	a	test	sentence	.	
en_tokenizer	589	533	320	1628	12737	269	
ko_tokenizer	946	561	245	5793	23819	295	
merged	589	533	320	1628 3676, 50771, 324		269	

한국어 CLIP과 영어 CLIP에서 모두 정상적으로 sentence가 하나의 토큰으로 토크나이징 되나 Merged clip에서는 sent enc e 3개의 토큰으로 나뉘는 것을 확인하였다.

긴 단어가 분리되는 현상을 조사해보았다. 처음 1000개에서는 sentence가 sent enc e로 분리가 되지 않기 때문에 특정 토큰(들)의 추가가 문제가 된 것이라고 생각하였다.

```
1 import torch.nn as nn
2 import re
3
4
5 embedding_dim = text_encoder.config.hidden_size
6 korean_language_tag = nn.Parameter(torch.randn(embedding_dim))
7 alpha = nn.Parameter(torch.tensor(1.5))
8 ko_vocab = ko_tokenizer.get_vocab()
9 ko_tokens = list(ko_vocab.keys())
10
11 # A 토크나이저의 현재 사전 가져오기
12 en_vocab = tokenizer.get_vocab()
13
14 # B의 토큰 중 A에 없는 토큰 추가
15 new_tokens = [token for token in ko_tokens if token not in en_vocab]
16
17 count = 0
18
19 for new_token in new_tokens:
20 # 새 토큰 추가
21 tokenizer.add_tokens(new_token)
22
23 #text_encoder.resize_token_embeddings(len(tokenizer))
24 count += 1
25 #print("Text encoder resized to match new tokenizer size.")
26 if count == 1000:
27 encoded_prompt = tokenizer.encode(test_sentences[0], truncation=True, return_tensors="pt")
28 decoded_prompt = tokenizer.decode(encoded_prompt[0])
29 print(f"tokens added: {new_tokens}, decoded_prompt")
30 count = 0
31
32 # A의 토큰을 허용할 가능도
33 en_embedding = text_encoder.get_input_embeddings()
34 ko_embedding = ko_text_encoder.get_input_embeddings()
35
36 # 모음의 일부로 허용할 가능도
37 for token in new_tokens:
38 if token in ko_vocab:
39 token_id_ko = ko_encoder(token)
40 token_id_en = en_encoder.convert_tokens_to_ids(token)
41 en_embedding.weight.data[token_id_en] = ko_embedding.weight.data[token_id_ko] + alpha + korean_language_tag
42 #for token in ko_vocab:
43 #    en_embedding.weight.data[token_id_ko], en_embedding.weight.data[token_id_en]
```

코드를 확인해보면 정확히 enc가 tokenizer에 추가될 때 sentence가 sent enc e로 분리되는 것을 확인할 수 있었다.

```
1 import torch.nn as nn
2 import re
3
4
5 embedding_dim = text_encoder.config.hidden_size
6 korean_language_tag = nn.Parameter(torch.randn(embedding_dim))
7 alpha = nn.Parameter(torch.tensor(1.5))
8 ko_vocab = ko_tokenizer.get_vocab()
9 ko_tokens = list(ko_vocab.keys())
10
11 # A 토크나이저의 현재 사전 가져오기
12 en_vocab = tokenizer.get_vocab()
13
14 # B의 토큰 중 A에 없는 토큰 추가
15 new_tokens = [token for token in ko_tokens if token not in en_vocab]
16
17 count = 0
18
19 for i in range(1360,1365):
20 # 새 토큰 추가
21 tokenizer.add_tokens(new_tokens[i])
22
23 #text_encoder.resize_token_embeddings(len(tokenizer))
24 #print("Text encoder resized to match new tokenizer size.")
25 encoded_prompt = tokenizer.encode(test_sentences[0], truncation=True, return_tensors="pt")
26 decoded_prompt = tokenizer.decode(encoded_prompt[0])
27 word = tokenizer.decode(tokenizer.encode(new_tokens[i]), truncation=True, return_tensors="pt")[0]
28 print(f"New tokens added: {new_tokens[i]}, decoded_prompt, word)
```

→ New tokens added: <|startoftext|>this is a test sentence . <|endoftext|> <|startoftext|> <|endoftext|>
New tokens added: <|startoftext|>this is a test sentence . <|endoftext|> <|startoftext|> <|endoftext|>
New tokens added: <|startoftext|>this is a test sentence . <|endoftext|> <|startoftext|> <|endoftext|>
New tokens added: enc <|startoftext|>this is a test sent enc e . <|endoftext|> <|startoftext|> enc <|endoftext|>
New tokens added: i(pi)e(g</w> <|startoftext|>this is a test sent enc e . <|endoftext|> <|startoftext|> i(pi)e(g</w> <|endoftext|>

CSV 파일로 new_tokens에서 추가되는 단어들을 다시 확인해보았다. enc 외에도 여러 영어 단어들이 존재하였다.

bingsu tokenizer가 한국어를 위한 tokenizer이기는 하나, 여러 한국어 문서에도 영어가 존재하기 때문에 영어에 대한 token이 존재하며, 한국어 중심의 tokenizing 방식이 영어 중심의 tokenizing 방식과는 차이가 있을 것이기 때문에 실제 단어가 아닌 단어의

	A	B	C	D	E	F	G
1359	챙?깡?깡?						
1360	챘?진?진? 짤?칼?						
1361	챙?깡?깡? 짤?째?						
1362	챙?깡?깡? 짤?짭?						
1363	챙?깡?깡? 짤?짭?						
1364	챠?침?깡?						
1365	enc						

따라서 이러한 영어 token이 존재할 때의 충돌을 방지하기 위해 new_token을 생성할

```
# B의 토큰 중 A에 없는 토큰 추가
new_tokens = [token for token in ko_tokens if token not in en_vocab and re.fullmatch(r'[a-zA-Z]+', token) is None]
```

실험 결과

1. 영어 문장: Hello, this is a test.

English	Encoded Prompt: [49406, 3306, 267, 589, 533, 320, 1628, 269, 49407] Decoded Prompt: < startoftext >hello , this is a test . < endoftext > Composed Prompt: < startoftext >hello , this is a test . < endoftext >< endoftext >
Korean	Encoded Prompt: [49406, 30584, 286, 946, 561, 245, 5793, 295, 49407] Decoded Prompt: < startoftext >hello , this is a test . < endoftext > Composed Prompt: < startoftext >hello , this is a test . < endoftext >< endoftext >
Merged	Encoded Prompt: [49406, 3306, 267, 589, 533, 320, 1628, 269, 49407] Decoded Prompt: < startoftext >hello , this is a test . < endoftext > Composed Prompt: < startoftext >hello , this is a test . < endoftext >< endoftext >

2. 한국어 문장: 안녕하세요, 이것은 테스트 문장입니다.

English	Encoded Prompt: [49406, 16071, 230, 167, 227, 243, 33992, 15074, 116, 18541, 498, 267, 12286, 35555, 225, 8276, 478, 169, 227, 234, 20305, 39820, 167, 105, 116, 20849, 98, 20849, 227, 33708, 13094, 353, 269, 49407] Decoded Prompt: < startoftext >안녕하세요 , 이것은 테스트 문장입니다 . < endoftext > Composed Prompt: < startoftext >안녕하세요 , 이것은 테스트 문장입니다 . < endoftext >< endoftext >
Korean	Encoded Prompt: [49406, 378, 182, 139, 179, 195, 389, 374, 90, 397, 224, 286, 379, 381, 177, 365, 210, 141, 179, 186, 476, 830, 139, 79, 90, 369, 72, 369, 179, 644, 367, 222, 295, 49407] Decoded Prompt: < startoftext >안녕하세요 , 이것은 테스트 문장입니다 . < endoftext > Composed Prompt: < startoftext >안녕하세요 , 이것은 테스트 문장입니다 . < endoftext >< endoftext >
Merged	Encoded Prompt: [49406, 16071, 230, 167, 227, 243, 33992, 15074, 116, 18541, 498, 267, 12286, 35555, 225, 8276, 478, 169, 227, 234, 20305, 39820, 167, 105, 116, 20849, 98, 20849, 227, 33708, 13094, 353, 269, 49407] Decoded Prompt: < startoftext >안녕하세요 , 이것은 테스트 문장입니다 . < endoftext > Composed Prompt: < startoftext >안녕하세요 , 이것은 테스트 문장입니다 . < endoftext >< endoftext >