since 1978
동서울대학교
DONG SEOUL UNIVERSITY

NLP4MusA 2021

Lyrics and Vocal Melody Generation conditioned on Accompaniment

Computer Software

장기태

Contents

1. Introduction

Lyrics and Vocal Melody Generation conditioned on Accompaniment
(NLP4MusA 2021)

Main contributions to the field are the following:

1. Generate lyrics and vocal melodies dependent on the accompaniment.
2. Optimize the Transformer architecture.
3. Propose an architecture that separates the generation of lyrics and vocal melodies.

Lyrics and Vocal Melody Generation conditioned on Accompaniment

Thomas Melistas
School of ECE
National Technical University of Athens
Greece
melistas.th@gmail.com

Theodoros Giannakopoulos
NCSR Demokritos
Greece
tyiannak@gmail.com

Georgios Paraskevopoulos
School of ECE
National Technical University of Athens
Greece
geopar@central.ntua.gr

Abstract

In this paper we present a previously unexplored task, the generation of lyrics and vocal melody for a given instrumental music piece in the symbolic domain. We model the above as a sequence-to-sequence task, using a memory efficient Transformer architecture, which we train on text event sequences that describe entire songs. Towards this end, we build a suitable dataset and apply musical analysis, compressing the instrumental part and making it key-independent. We further design a novel architecture to decouple lyrics and melody generation, making it possible to use pretrained language models and conditioning on lyrics. Finally, Mellotron is used to turn the generated sequences into singing audio.

1 Introduction

A significant part of research on singing has focused on information retrieval tasks, such as lyrics or melody transcription (Stoller et al., 2019; Nishikimi et al., 2019), as well as on singing voice synthesis (Nishimura et al., 2016). Generating the (symbolic) content of singing, namely lyrics and vocal melody, has only recently started gaining more attention. Relevant work has focused on generating lyrics for a specific music style or melody and lyrics-conditioned vocal melody generation.

Vocal music coexists with instrumental in most contemporary genres. However, despite the growing interest on studying the relation of lyrics and vocal melody, the connection of both to the accompaniment remains overlooked. In this work we aspire to fill this substantial gap.

We model the instrumental-conditioned generation of vocal melody and lyrics as a seq2seq task. First, we create pairs of text event sequences, which we use to train a baseline encoder-decoder Transformer architecture. We then propose a way to decouple lyrics from vocal melody generation, by inserting another decoder. Finally, we bring this symbolic output into the audio domain and perform a subjective evaluation study, using a singing voice synthesis model. In contrast to previous works that study vocal generation on the sentence level, we model full songs, which presents additional technical challenges.

Our main contributions to the field are the following: **(a)** We introduce the task of lyrics and vocal melody generation conditioned on the accompaniment. **(b)** We build a suitable dataset for this task by enforcing consistent tokenization. We apply musical analysis to compress the instrumental part up to 20% of the original, resulting to faster training. **(c)** We optimize the Transformer architecture in order to model full song sequences of up to 60k tokens in a single GPU. **(d)** We propose an architecture that decouples lyrics and vocal melody generation, providing the ability to use pretrained language models and predefined lyrics.

We release all code, datasets and some generated samples[1].

2 Related Work

Conditional Vocal Melody Generation In (Madhumani et al., 2020) a combination of word and syllable embeddings is used as input to an LSTM encoder (Hochreiter and Schmidhuber, 1997) that uses a vector to attend to three separate decoders, for note, duration and rest. Yu et al. (2021) use an LSTM that takes as input lyrics embeddings and noise vectors to sample MIDI sequences, which are provided to another LSTM alongside text embeddings and classified as real or fake. Another approach (Liu et al., 2020) studies singing voice generation without any melody or lyrics information, using GANs (Goodfellow et al., 2014) conditioned also on accompaniment.

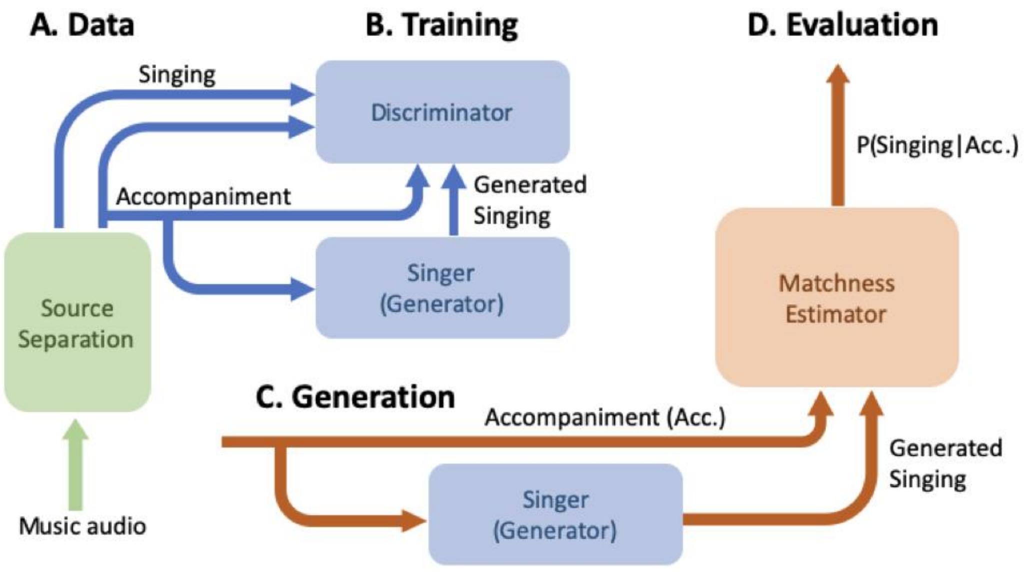
[1] github.com/gulnazaki/lyrics-melody

2. Related Work

Conditional Vocal Melody Generation

A combination of word and syllable embeddings is used as input to an LSTM encoder

Use an LSTM that takes as input lyrics embeddings and noise vectors to sample MIDI sequences



Lyrics Generation

Use an encoder-decoder LSTM to generate lyrics, taking into account the only rhythmic quality of the melody

3. Dataset Description

Lakh MIDI Dataset(LMD)

Creating A More Consistent Dataset

LMD is not oriented towards the analysis of vocals.

- Keeping only English lyrics.
- Remove any metadata.
- Deriving the vocal part, assign each lyric to the closest note
- Choosing the track with the most matches

Genre	Number of MIDI	Number of MIDI Images
Pop/Rock	8603	45,259
Country	962	5381
Electronic	694	4713
R&B	475	7557
Latin	293	1410
Jazz	280	1538
New Age	230	878
Rap	117	565
International	86	491
Reggae	69	344
Folk	64	281
Vocal	41	230
Blues	32	164
Total	11,946	68,811

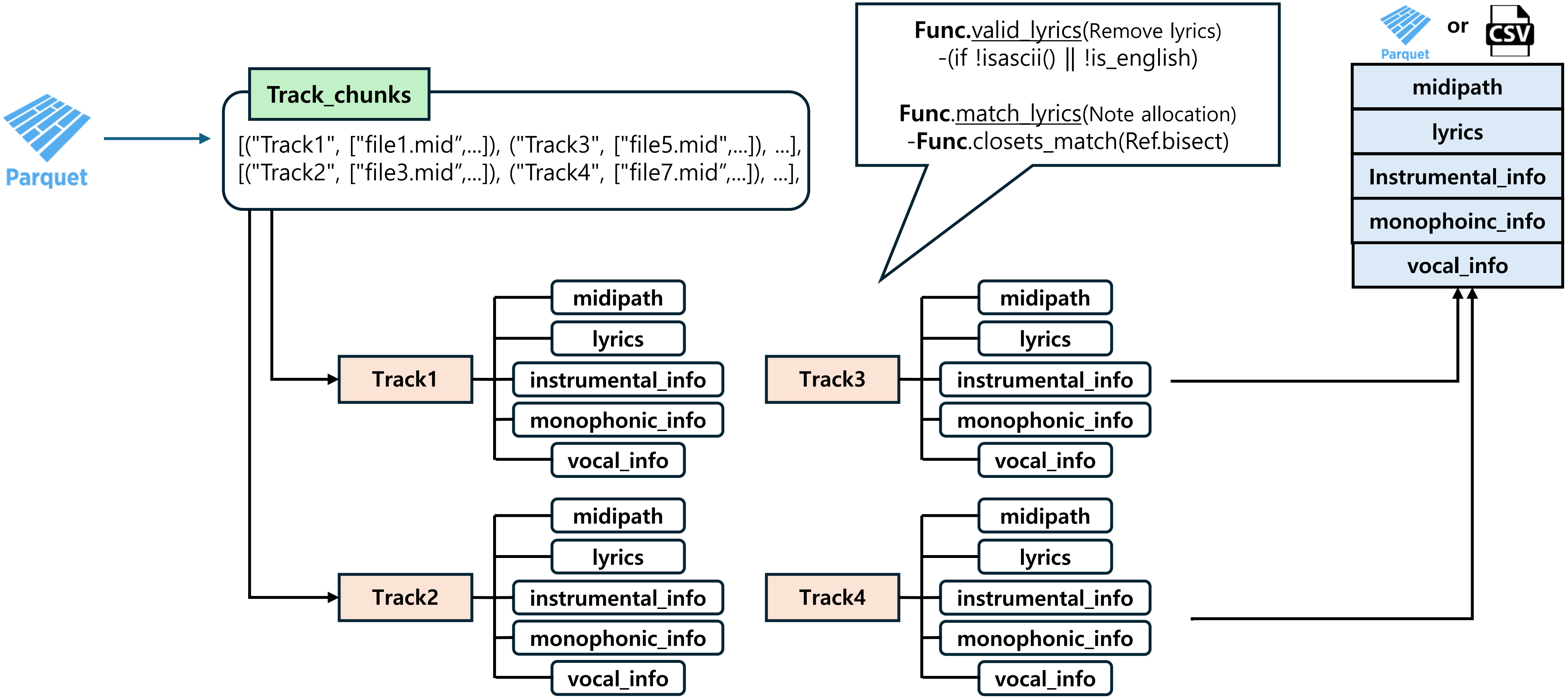
Modalities

Audio

Lyrics

3. Dataset Description

Dataset Processing



Made by장기태

3. Dataset Description

Text Event Format

All sequences consist of the following types of tokens

- Note on (a note of this pitch starts)
- Note off (a note ends)
- Wait time (time passed in MIDI ticks)

Instrumental Text	Vocal Text
Limit the notes within the piano range(88 pitches)	Syllable/phoneme
Append the names of instruments to note tokens	Extension
	Boundary

Instrumental Text Event Representation

wait time event
note on event

'W_2000', 'W_84', 'GUITAR_ON_54', 'BASS_ON_42', 'DRUMS_ON_35', 'DRUMS_ON_51', 'W_4', 'DRUMS_OFF_35', 'DRUMS_OFF_51', 'W_4', 'GUITAR_OFF_54', 'W_34', 'BASS_OFF_42'

note off event

Vocal Text Event Representation

<in>
new word (boundary event)
<a>

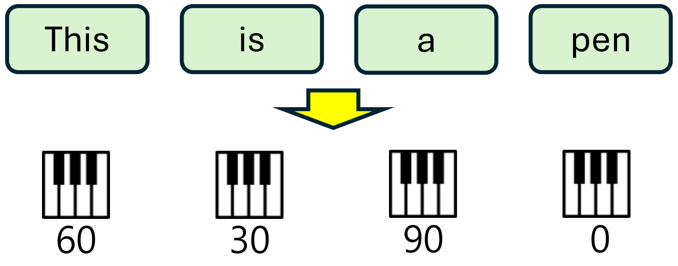
'IH0 N', 'ON_72', 'W_490', '_OFF_', 'W_100', 'N_W', 'AH0', 'ON_69', 'W_245', '_OFF_', 'W_245', 'N_W', 'M AA1', 'ON_72', 'W_300', '_OFF_', 'W_220', 'R K AH0 T', 'ON_72', 'W_245', '_OFF_', '_R_', 'ON_70', 'W_150', '_OFF_',

<rket>
<ma>
extension event

3. Dataset Description

Create Vocabs

- read based on whether vocabulary generation for music analysis is Enabled
- Depending on the type, **noteons** and **noteoffs** are configured.
- If the occurrence count of an event is equal to or greater than the specified threshold (thresh), the event is added to the **noteons** list




```python
if music_analysis:
    ext = os.path.splitext(dataset)[1]
    if ext != '.csv' and ext != ".parquet":
        exit("Provide a .csv or .parquet file for vocal vocabulary")
    try:
        df = pd.read_csv(dataset, usecols=[_type_]) if ext == '.csv' else pd.read_parquet(dataset, columns=[_type_])
    except ValueError:
        exit("The provided file doesn't have a {} column".format(_type_))
    except:
        exit("Problem with dataset file")
    if _type_ == 'instrumental':
        noteons = ['_DB_', '_B_']
        noteoffs = ['_REST_']
    else:
        noteons = []
        noteoffs = ['_OFF_']
    note_cnt = Counter([event for f in df[_type_] for event in json.loads(f) if event[:3] == 'ON_'])
    noteons += [tok for tok, cnt in note_cnt.most_common() if cnt >= thresh]
```

3. Dataset Description

Create Vocabs

- **noteons** are configured based on whether to include velocity information.
- **noteoffs** are configured based on whether it's monophonic or not.

```
else:
    if include_velocity:
        noteons = ['{}ON_{}_V{}'.format(instrument, pitch, velocity) for instrument in instruments
                                                                       for pitch in range(load_midi.MIN_PITCH, load_midi.MAX_PITCH + 1)
                                                                           for velocity in range(load_midi.VEL_QUANT >> 1, 128, load_midi.VEL_QUANT)]
    else:
        noteons = ['{}ON_{}'.format(instrument, pitch) for instrument in instruments
                                                        for pitch in range(load_midi.MIN_PITCH, load_midi.MAX_PITCH + 1)]

    if monophonic:
        if _type_ == 'instrumental':
            noteoffs = ['{}OFF'.format(instrument) for instrument in instruments]
        else:
            noteoffs = ['_OFF_']
    else:
        noteoffs = ['{}OFF_{}'.format(instrument, pitch) for instrument in instruments
                                                          for pitch in range(load_midi.MIN_PITCH, load_midi.MAX_PITCH + 1)]
```

3. Dataset Description

Create Vocabs

- **singing_tokens** represent specific events in vocal data, such as notes and rhythms.
- **for_decoupled** , **include_vowels** is true, additional vocal tokens are generated and added to **singing_tokens**.

```
if _type_ == 'vocal':
    singing_tokens = ['N_DL', 'N_L', 'N_W', '_C_', '_R_']
    if for_decoupled:
        if include_vowels:
            singing_tokens += [v + stress for v in ['AA', 'AE', 'AH', 'AO', 'AW', 'AY', 'EH', 'ER', 'EY', 'IH', 'IY', 'OW', 'OY', 'UH', 'UW'] for stress
    else:
        if not music_analysis:
            ext = os.path.splitext(dataset)[1]
            if ext != '.csv' and ext != ".parquet":
                exit("Provide a .csv or .parquet file for vocal vocabulary")
            try:
                df = pd.read_csv(dataset, usecols=["vocal"]) if ext == '.csv' else pd.read_parquet(dataset, columns=["vocal"])
            except ValueError:
                exit("The provided file doesn't have a 'vocals' column")
            except:
                exit("Problem with dataset file")

        singing_cnt = Counter([event for f in df['vocal'] for event in json.loads(f) if '_' not in event])
        singing_tokens += [tok for tok, cnt in singing_cnt.most_common() if cnt >= thresh]
```

3. Dataset Description

Chord Reduction

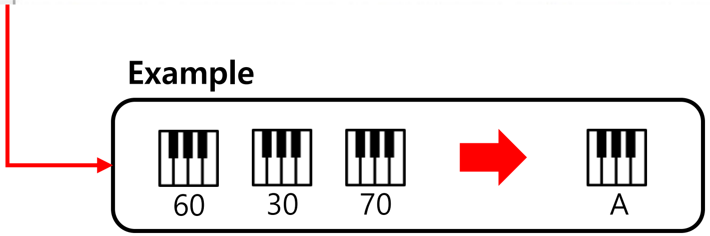
- The representation results in very long sequences when instrumental task
- Create a chord reduction of the instrumental, using the music21 library
- Individual instruments are merged and every new note results in the formation of a new chord

→ Roman Numeral Analysis

Result

	Vocal	Instrumental	Reduced
median	1645	13041	3220
max	6115	59120	11730

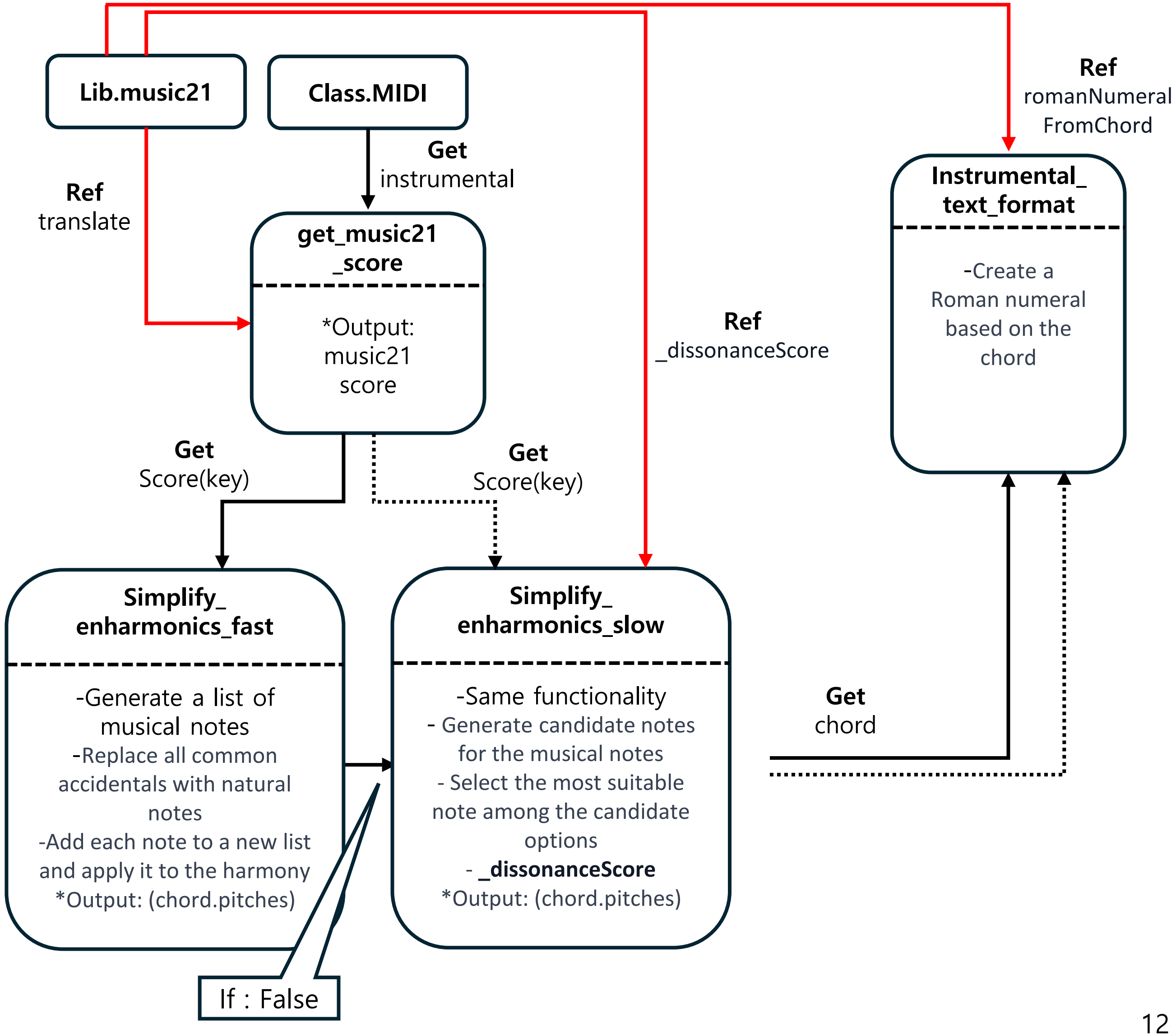
Example



3. Dataset Description

Chordify instrumental and use roman numerals

Function	Content
get_music21_score	Convert an instrument(by MIDI) To a music21 score
Simplify_enharmonics _fast	Simplify the notes of the chords To match the given key
Simplify_enharmonics _slow	Same functionality If Simplify_enharmonics_fast fails, proceed it
Instrumental_ text_format	Encode the chord to roman



Made by장기태

4. Proposed Method

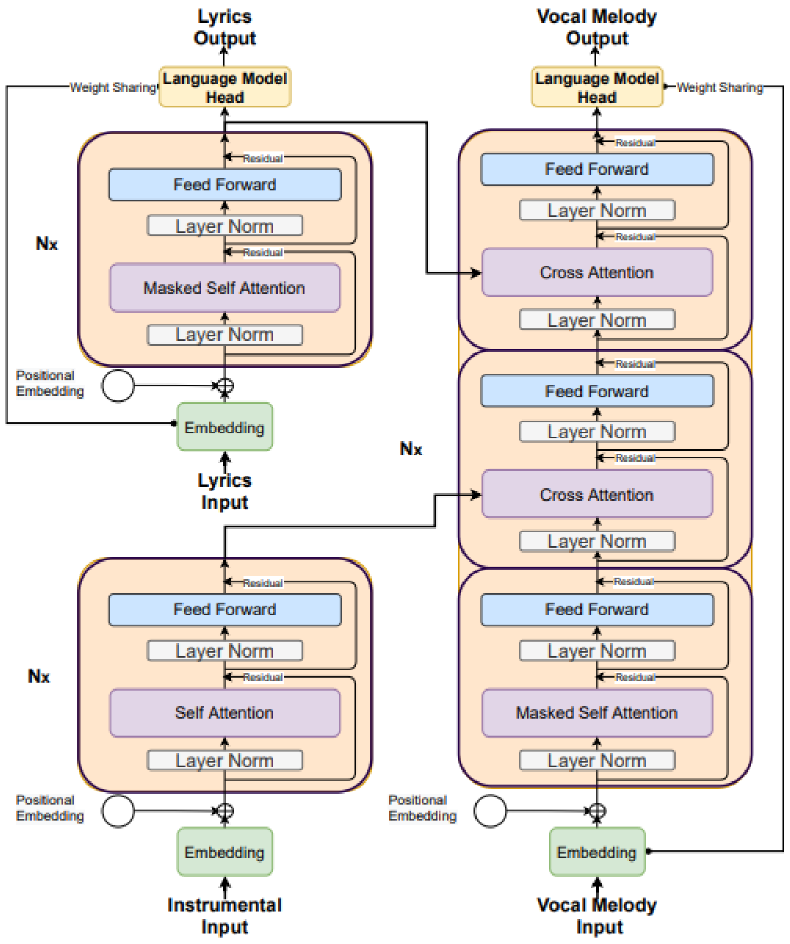
Encoder-Decoder Architecture

The memory footprint

- dot-product attention mechanism → Performer architecture
- Residual Network → Reversible Network
- Feed-Forward Layer → Feed-Forward Chunking

Decoupled Architecture

Add a separate decoder for lyrics
Cross Attention

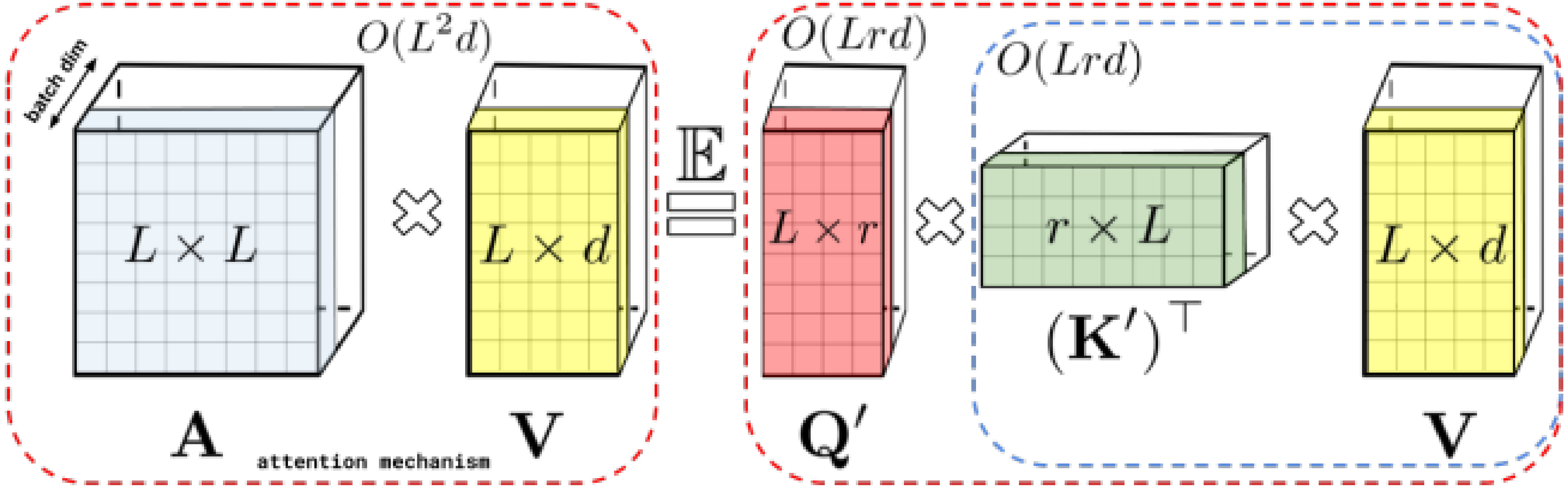


4. Proposed Method

dot-product attention mechanism → Performer architecture

- Instead of **SM** $(\frac{QK^T}{\sqrt{d}})V$,use $\boldsymbol{Q'} = \boldsymbol{\phi(Q)}$, $\boldsymbol{K'} = \boldsymbol{\phi(K)}$, where **SM** $(\frac{QK^T}{\sqrt{d}})$ = $\boldsymbol{(Q'\ K'^T)}$.
- Then **SM** $(\frac{QK^T}{\sqrt{d}})V$ = $\boldsymbol{(Q'K'^T)\ V}$ = $\boldsymbol{Q'\ (K'^T\ V)}$ by the commutative property of matrix multiplication
- This calculation is $\boldsymbol{O(N)}$ for sequences of length $\boldsymbol{N}$

→ **FAVOR+ (Fast Attention Via positive Orthogonal Random features)**



4. Proposed Method

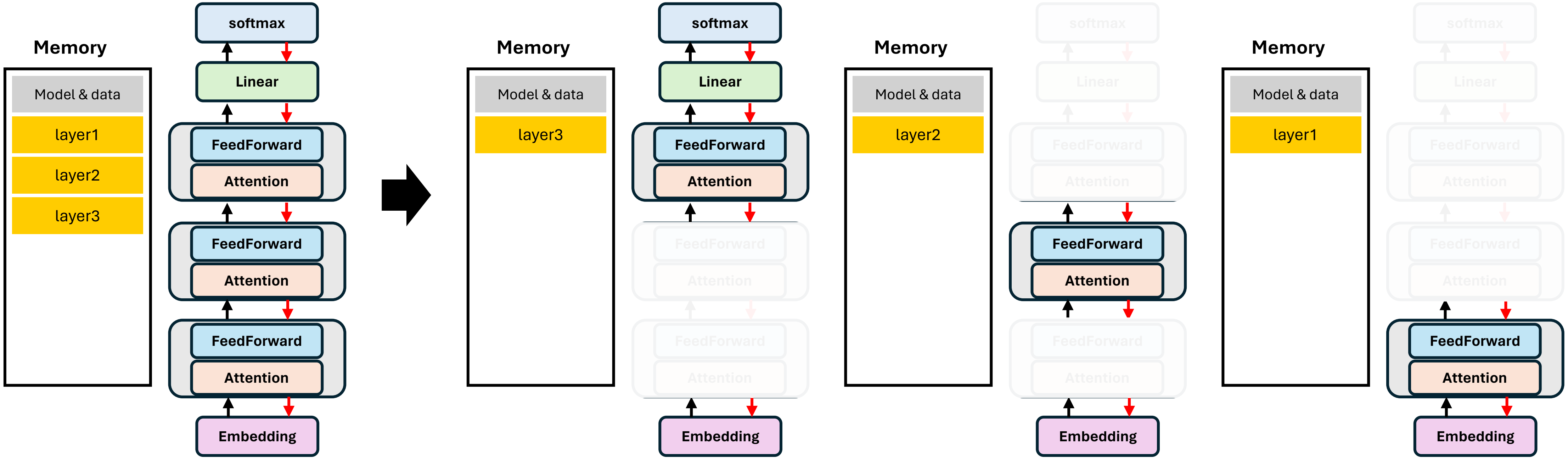
Residual Network → Reversible Network

Characteristics of Residual Block

- Repetition of Attention and Feed Forward Layers
- Intermediate results are stored for backpropagation

Idea

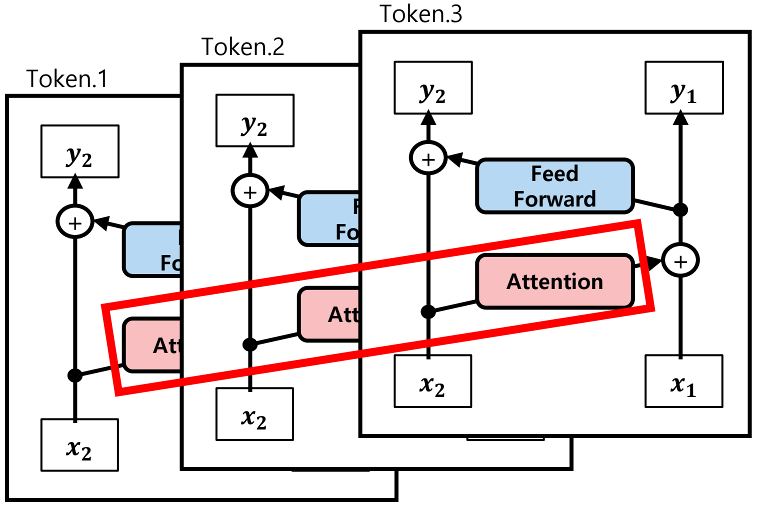
Through output(y) calculate the value of input(x)



Made by장기태

4. Proposed Method

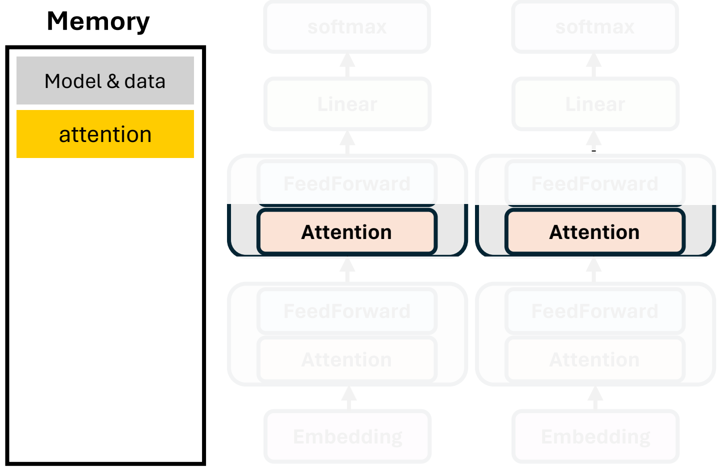
Feed-Forward Layer → Feed-Forward Chunking



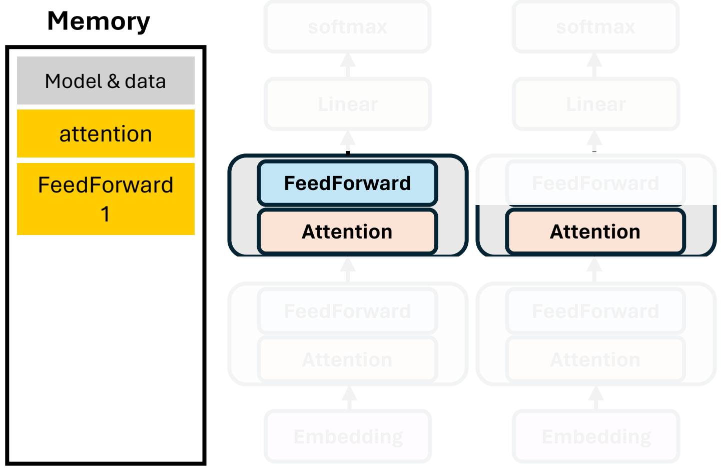
Characteristics of Feedforward Layer

- The size of the Feedforward Layer is large.
- **Unlike Attention, it can be computed regardless of the sequence's position.**

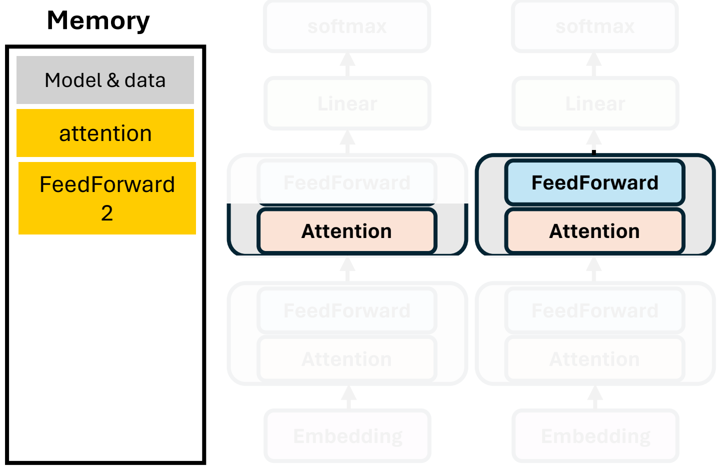
Compute Attention



Compute Feed-Forward1 & Memory release

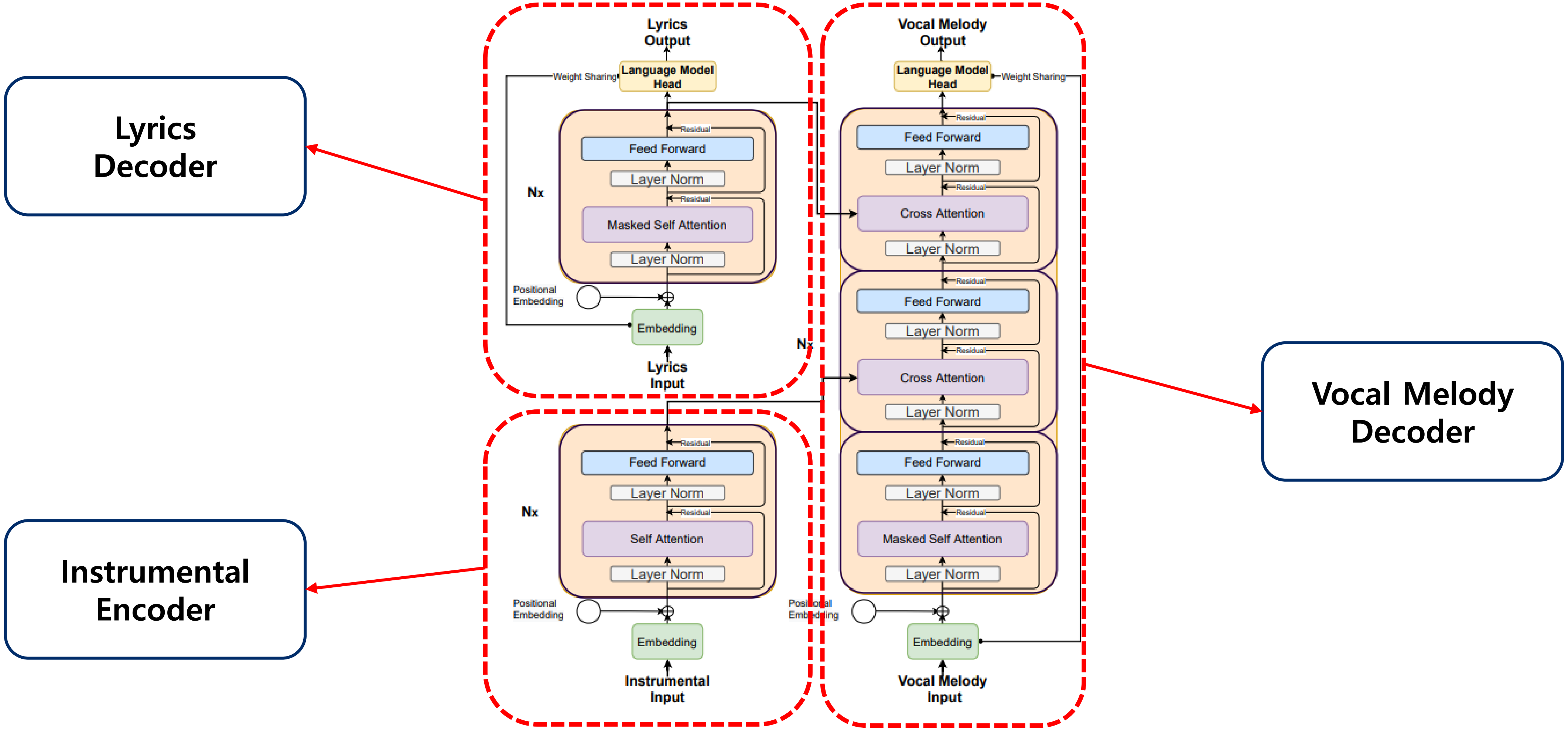


Compute Feed-Forward2 & Memory release



4. Proposed Method

Decoupled Architecture



5. Decoupled Performer

Utility Functions

lyrics_dict={‘Prefix_apple’ : 1}

group_dict_by_key(cond, d)

string_begins_with(prefix)
group_by_key_prefix(prefix, d)
extract_enc_lm_dec_kwargs(kwargs)

5. Decoupled Performer

Class Structure

Class Initialization

The constructor method receives key parameters and configurations.

Module Initialization

Encoder (enc), decoder (dec), and decoupled_decoder (lm) are initialized based on their respective configurations.

Generation Method

Performs forward pass to generate lyrics and vocals based on given input.

5. Decoupled Performer

Class Structure

Class Initialization

- Extract and group settings for encoders, decoders, and language models using the `extract_enc_lm_dec_kwargs` function.
- Initializes the class with the following

```python
def __init__(
    self,
    dim,
    tie_token_embeds = False,
    no_projection = False,
    pretrained_lm = "",
    **kwargs
```

Module Initialization

- Created by initializing the `PerformerLM` model for each part with the given keyword arguments.
- If a pre-trained language model is provided, load it and apply it to the language model.

```python
enc = PerformerLM(**enc_kwargs)
lm = PerformerLM(**lm_kwargs)
dec = PerformerLM(**dec_kwargs)
#keyword arguments 로 각각의 인자들의 키-쌍 값이 전달
```

```python
self.enc = enc
self.lm = AutoregressiveWrapper(lm)
self.dec = AutoregressiveWrapper(dec)
```

Generation Method

- Use the function to extract and set the parameters of the encoder, language model, and decoder. After that, perform the generation for each model and finally return the lyrics and vocals.

```python
def generate(self, instrumental, lyrics_start, lyrics_len,
             vocals_start, vocals_len, **kwargs):
    enc_kwargs, lm_kwargs, dec_kwargs, kwargs =
    extract_and_set_enc_lm_dec_kwargs(kwargs)
    instrumental_encodings = self.enc
    (instrumental, return_encodings = True, **enc_kwargs)
     lyrics_encodings, lyrics = self.lm.generate
     vocals = self.dec.generate
     return lyrics, vocals
```

Made by장기태

5. Decoupled Performer

Train Decoupled Performer

Define Command-Line Argument Parsing Function

Define Dataset Class

Define DataLoader - Related Function

Define Valid Structure Metric Calculation Function

Main Function

```
model = DecoupledPerformer(
    dim=768,  # 모델 내부의 임베딩 차원
    enc_heads=6,  # 인코더의 어텐션 헤드 수
    lm_heads=12,  # 언어 모델의 어텐션 헤드 수
    dec_heads=6,  # 디코더의 어텐션 헤드 수
    enc_depth=6,  # 인코더의 층 수
    lm_depth=6,  # 언어 모델의 층 수
    dec_depth=6,  # 디코더의 층 수
    enc_ff_chunks=10,  # 인코더 피드포워드 레이어에서 청크 수
    lm_ff_chunks=1,  # 언어 모델 피드포워드 레이어에서 청크 수
    dec_ff_chunks=10,  # 디코더 피드포워드 레이어에서 청크 수
    enc_reversible=True,  # 인코더의 레이어가 역방향 처리를 지원할지 여부
    lm_reversible=True,  # 언어 모델의 레이어가 역방향 처리를 지원할지 여부
    dec_reversible=True,  # 디코더의 레이어가 역방향 처리를 지원할지 여부
```

Made by장기태

ASK 2024

Annual Symposium of KIPS 2024

since 1978 동서울대학교 DONG SEOUL UNIVERSITY

한국정보처리학회 ASK 2024

T5 모델을 활용한 반주 기반 가사 생성 기법에 관한 연구

Computer Software

장기태

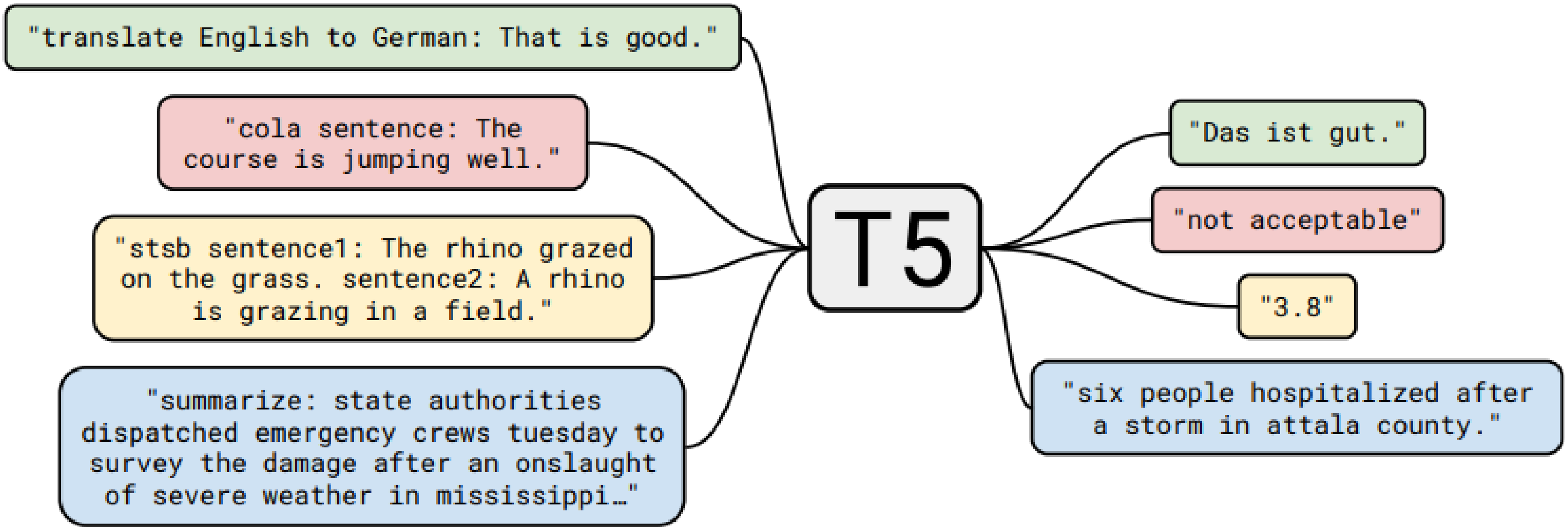
Contents

1. Introduction

What is T5

Text-To-Text Trasfer Transformer = T5

Versatile Transformer model for text-to-text tasks
T5 features a encoder-decoder architecture, enabling it to handle diverse text tasks with a single model.



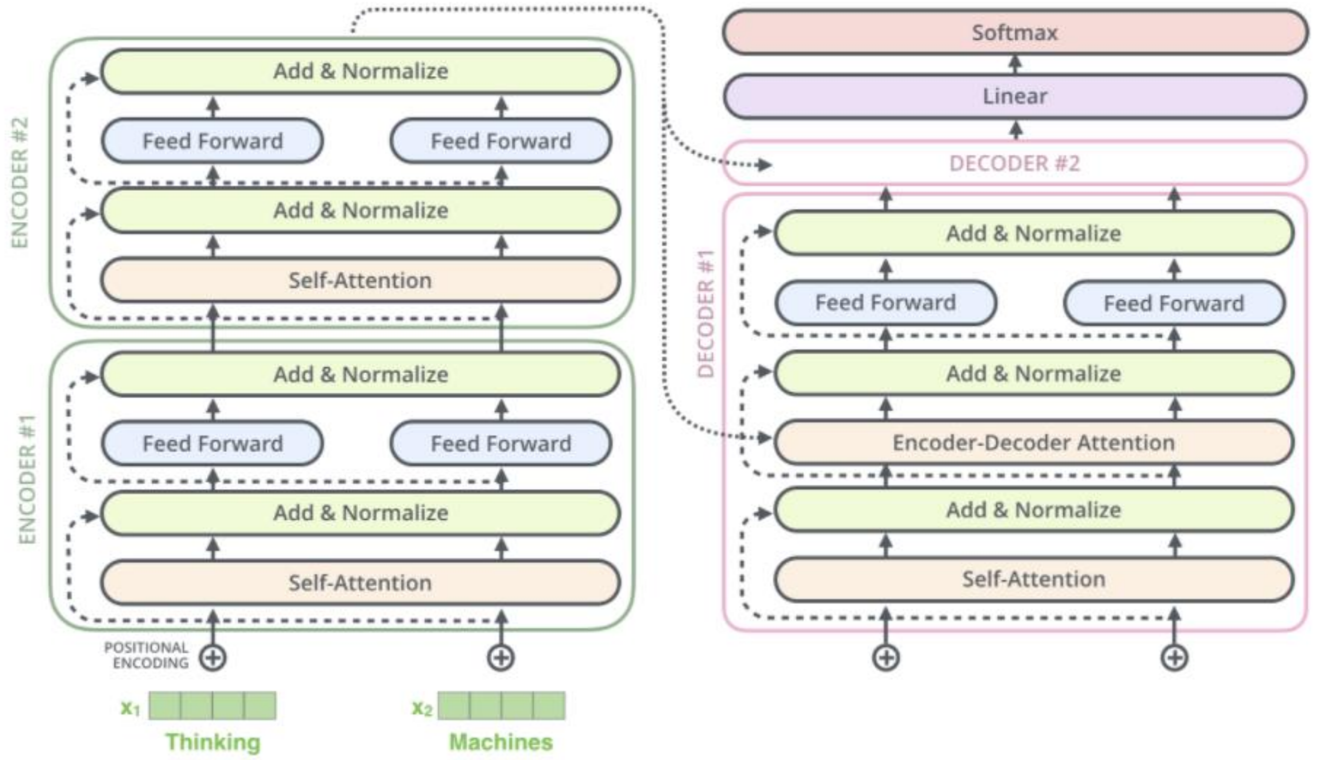
T5 Model Features

Follows the normal encoder-decoder structure

1. Using Relative Positional Embedding instead of Absolute Positional Embedding

2. Position Embedding Parameter Sharing

3. Using the c4 dataset for pre-training

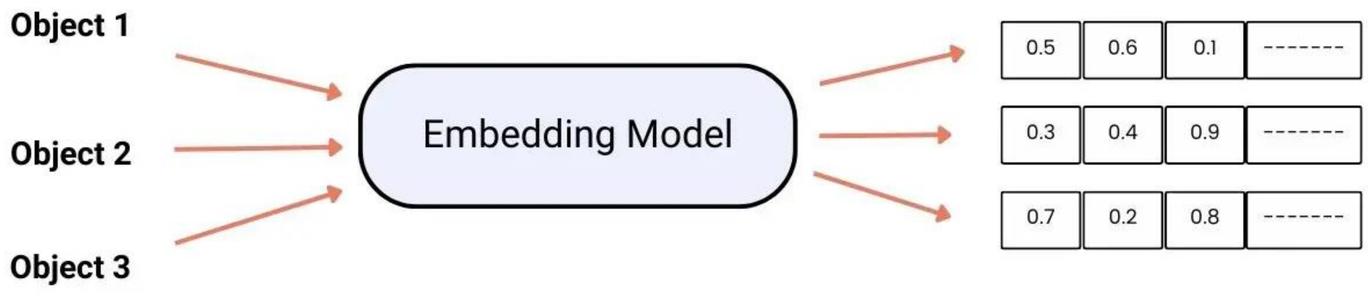


Relative Positional Embedding

Consider pair-wise relationship between inputs

Given the value 'relative position embedding' between key and query within self-attention

Self-attention score + relative position embedding as the final score

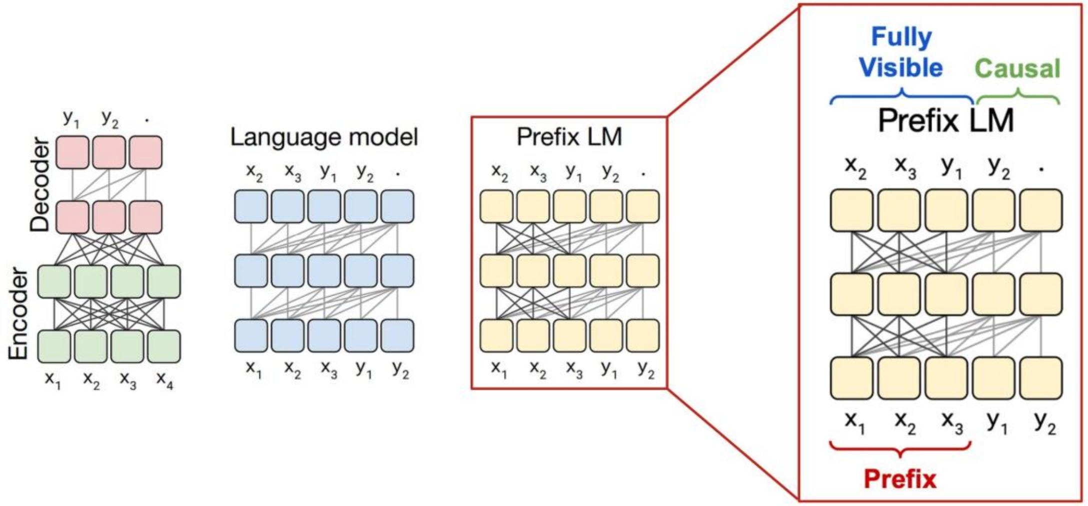


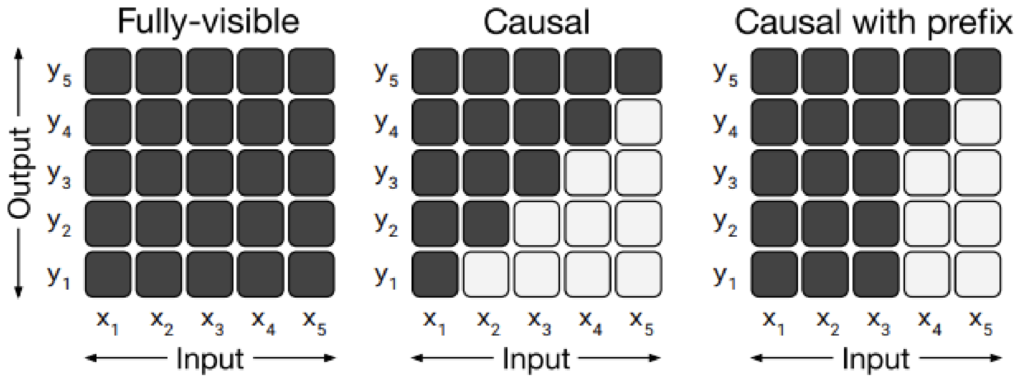
+ Relative position information

Causal Attention with Prefix

When causal attention performs attentions on tokens before the current position,

it performs additional attentions on tokens after a certain prefix.





Fully-visible
-Query can pay attention to any key

Casual
-Query can pay attention to Keys in timesteps prior to the current one

Casual with prefix
-The part given by prefix, the input text is fully visible, The output text is casual

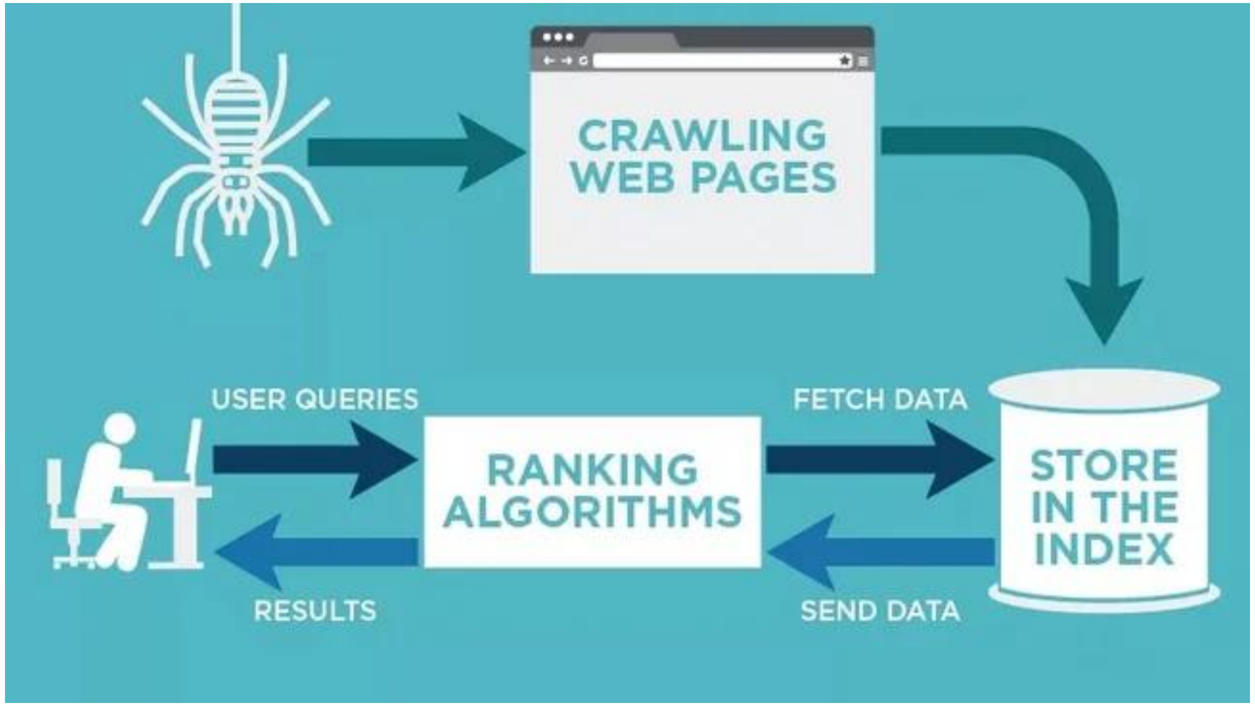
Made by장기태

3. Input Format

C4 Dataset

Colossal Clean Crawled Corpus = C4

Organize text from various web pages collected through web crawling.
Heuristic clean up to refine text



- only retained lines that ended in a terminal punctuation mark
- discarded any page with fewer than 3 sentences and only retained lines that contained at least 5 words.
- removed any page that contained any word on the “List of Dirty, Naughty, Obscene or Otherwise Bad Words”

3. Input Format

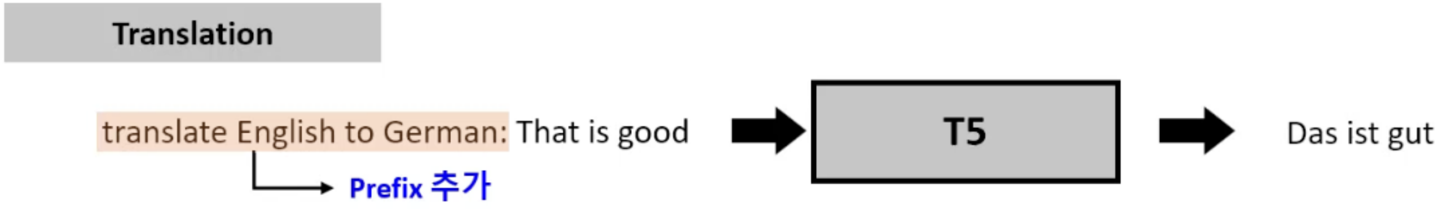
Prefix

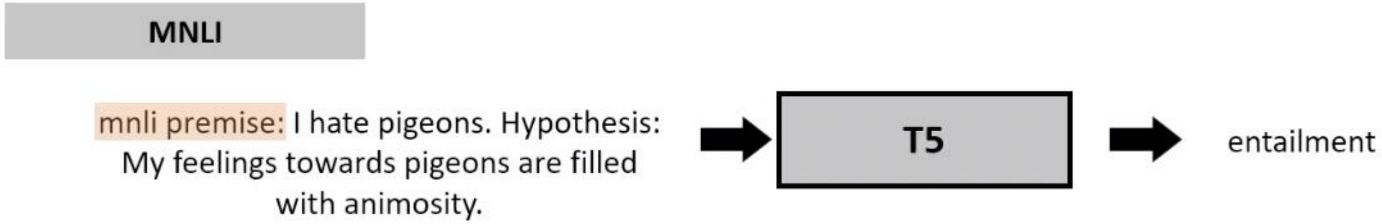
Add prefix in original input sequence

In order to process all tasks in a "text-to-text" model, certain tasks use some tricks.

Ex) MNLI (Multi-Genre Natural Language Inference corpus)

If a word like "hamburger" is generated outside of the correct candidate label, we simply say that the answer is incorrect and do not take any other action





3. Input Format

Task : input_text

Original input
Sentence: I am great man
Original target : 1

Processed input
CoLA sentence: I am great man
Processed target : acceptable

Original input
Sentence: It is so fun
Original target : 1

Processed input
SST2 sentence : It is so fun
Processed target : positive

4. Our Proposed Method

Preprocessing - Same as the existing process

Dataset Refinement

1. Keeping only English lyrics.
2. Remove any metadata.
3. Deriving the vocal part, assign each lyric to the closest note
4. Choosing the track with the most matches
5. Extract 5 columns

Chord Reduction

1. Create a chord reduction of the instrumental, using the music21 library
2. every new note results in the formation of a new chord(with. Romen Nemeral Analysis)

Create Vocab

Depending on the type, noteons and noteoffs are configured.

Model Training and Lyrics Generation - Applying T5

T5 Tokenize

- Tokenizer specialized for T5
- Instrumental_text(Chord), Lyric_Text

T5 Fine-Tuning

- T5-small model load
- Vanila_transformer → Fine-Tuning
- Save model(“pt”)

Lyrics Generation

- Generated from our proposed model
- Input the Instrumental and Output Lyric
- Subjective Evaluation

Made by장기태

4. Our Proposed Method

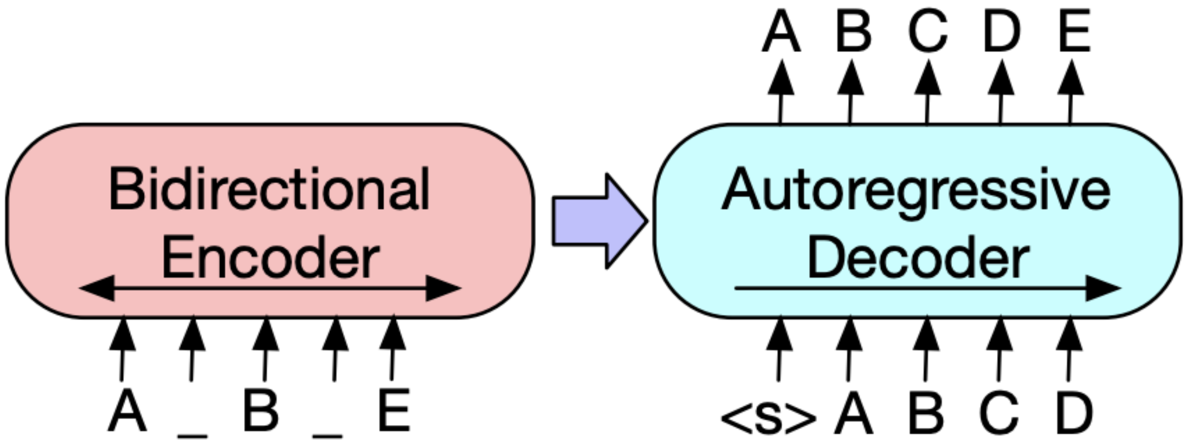
Why T5?

GPT

Generating Lyrics through Song Outlines

BART

Similar Structure, Lack of Prior Knowledge
Ex)Melody Emotion, Lyric Writing Techniques



Line needed:
I don't know, do you,
I can't see what I'm running to,
Any step could mean disaster
Line 3 needed: *7 syllables, end rhymes with* ***"to"***

Line needed:
'Cause they know freedom's not a race
And what they chase, it can't escape
Line 3 needed: *12 syllables, end rhymes with* ***"race"***

Made by장기태

4. Our Proposed Method

Why T5?

Input : Instrumental

wait time event

note on event

'W_2000', 'W_84', 'GUITAR_ON_54', 'BASS_ON_42', 'DRUMS_ON_35', 'DRUMS_ON_51', 'W_4', 'DRUMS_OFF_35', 'DRUMS_OFF_51', 'W_4', 'GUITAR_OFF_54', 'W_34', 'BASS_OFF_42'

note off event

Model

output : Lyric

```
That have to me to score a while I have you open your little thing I skipped by chance,
put ahh such a disillusionment's the sunset's
Break away
ev'cause throughout you need more  love,  right now,
But now
and all the clouds and stronger coming back home and runnin'cause tonight
So I chose to theodo do you put an end of the fight
'Tears have been waiting on the road when you,  I know I'm so much tonight it's dancing in your children

With the startout of the street
When all the one more than just like you once more blondesus really have always treat me each day
impossible as one to being unkind just loves me walking in the breeze
```

If GPT(decoder only)

Generating Lyrics through Song Outlines

GPT processes unidirectional information

-It means that the model can proceed with generation without considering the entire context.

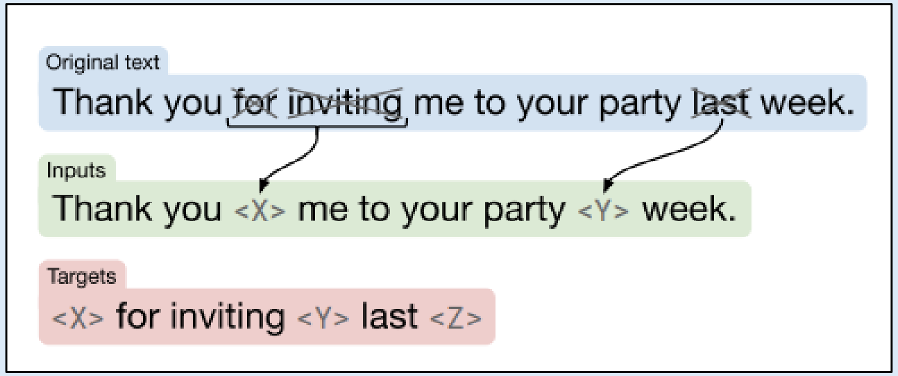
-It faces constraints in comprehensively considering various input sources, such as accompaniment information

Made by장기태

4. Our Proposed Method

Why T5?

T5 - Span corruption



Span Corruption method selectively masks sequences of consecutive words (spans) in text and requires the model to predict these masked spans.

Advantages
Contextual Understanding: The Span Corruption method enhances the model's ability to infer missing information from surrounding context.
Precise Text Generation: A model trained to predict consecutive word sequences generates lyrics that are more accurate and natural.

Precision and Consistency

BART - Denoising

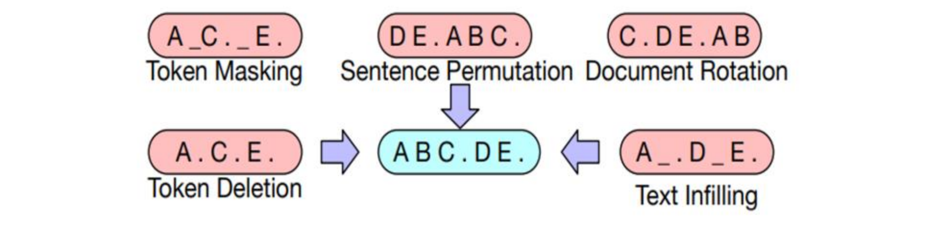


Figure 2: Transformations for noising the input that we experiment with. These transformations can be composed.

The noise recovery method involves introducing various forms of noise (e.g., token masking, sentence shuffling, text deletion, etc.) into the input text and then training the model to recover the original text.

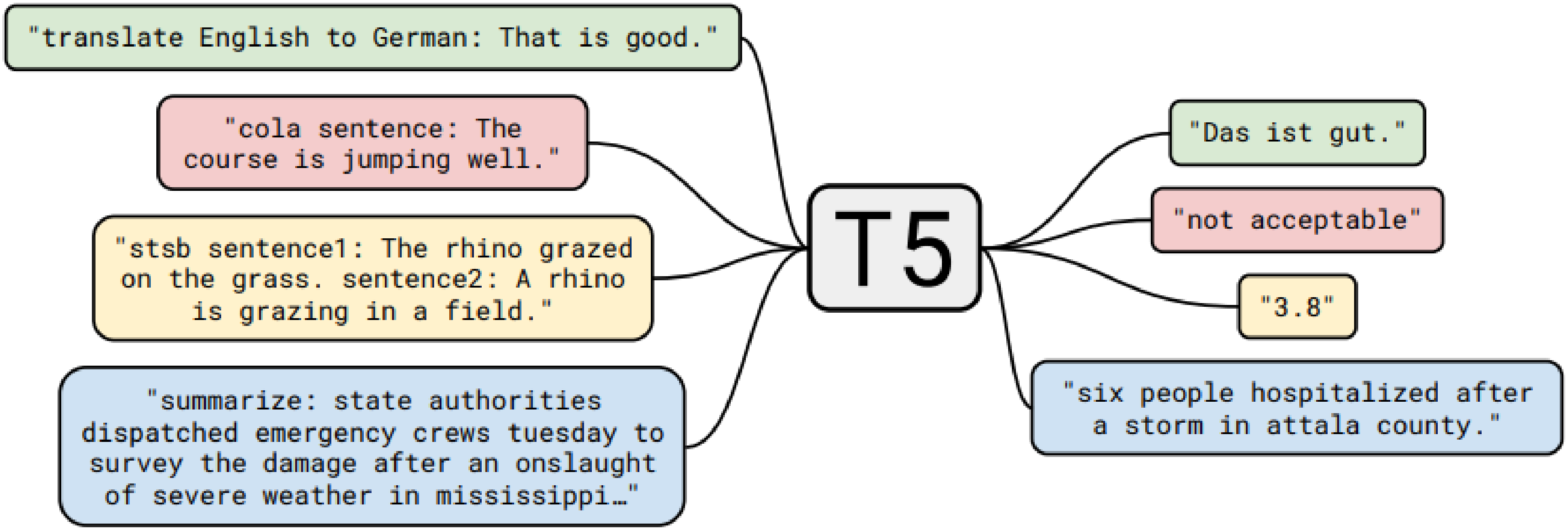
Advantages
Learning Complex Sentence Structures: Introducing various types of noise (token masking, sentence reordering, etc.) and training the model to recover from them enables the model to better understand and process complex language structures. Flexibility: The noise recovery method exposes the model to a wide range of language variations, allowing it to handle language more flexibly.

Creativity and Diversity

Similar Structure, Lack of Prior Knowledge

4. Our Proposed Method

Why T5?



1. **Precision and Consistency based on Detailed Accompaniment Content – Bidirectionality + Autoregressive Approach**
2. **Precise, Consistent – Span corruption**
3. **Creative, Diverse – C4 dataset, allowing various input formats**
4. **Scalability – mT5's diverse language processing, koT5, T5-japanese**

4. Our Proposed Method

T5 Fine-Tuning

1. Importing libraries and modules

```
import pandas as pd
import torch
from torch.utils.data import DataLoader, Dataset, random_split
from transformers import T5Tokenizer, T5ForConditionalGeneration, AdamW
import sacrebleu
from tqdm.auto import tqdm
import csv
```

2. Class definition : Dataset

```
class MidiDataset(Dataset):
    def __init__(self, tokenizer, dataframe, vocab, max_length=512):
        self.tokenizer = tokenizer
        self.data = dataframe
        self.vocab = vocab  # 반주 어휘
        self.max_length = max_length

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

    def __getitem__(self, idx):
        item = self.data.iloc[idx]
        instrumental_data = eval(item['instrumental'])  # 리스트 형태로 저장된 문자열을 리스트로 변환
        lyrics_data = item['lyrics']

        # 반주 데이터를 어휘 인덱스로 변환
        instrumental_tokens = [str(self.vocab[token]) if token in self.vocab else '0' for token in instrumental_data]
        instrumental_str = ' '.join(instrumental_tokens)

        source = f"instrumental: {instrumental_str}"
        target = f"lyrics: {lyrics_data}"

        source_encodings = self.tokenizer(source, max_length=self.max_length, padding='max_length', truncation=True, return_tensors="pt")
        target_encodings = self.tokenizer(target, max_length=self.max_length, padding='max_length', truncation=True, return_tensors="pt")

        return source_encodings.input_ids.squeeze(), target_encodings.input_ids.squeeze()
```

4. Our Proposed Method

T5 Fine-Tuning

3. New) Model Load

```
# 모델 설정
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = T5ForConditionalGeneration.from_pretrained('t5-small').to(device)
optimizer = AdamW(model.parameters(), lr=1e-4)
```

4. Learning code→ Fine-Tuning code

```
model.train()
for epoch in range(50):  # 학습 에포크
    for source, target in tqdm(train_loader, desc=f"Training Epoch {epoch}"):
        source, target = source.to(device), target.to(device)
        optimizer.zero_grad()
        outputs = model(input_ids=source, labels=target)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
```

Lyric generation : [instrumental]

Task_token   Input_text

→ Target : Lyric

5. Experiment

BELU SCORE

```
model.eval()
predictions, actuals = [], []
with torch.no_grad():
    for source, target in tqdm(val_loader, desc="Validating"):
        source = source.to(device)
        outputs = model.generate(
            input_ids=source,
            max_length=512,
            num_beams=5,
            repetition_penalty=2.0,
            no_repeat_ngram_size=4
        )
        pred_text = tokenizer.batch_decode(outputs, skip_special_tokens=True)
        true_text = tokenizer.batch_decode(target, skip_special_tokens=True)

        predictions.extend(pred_text)
        actuals.extend([true_text])

# BLEU-2gram 스코어 구하기
bleu_2_score = sacrebleu.corpus_bleu(predictions, actuals, weights=(0.5, 0.5)).score

# BLEU-3gram 스코어 구하기
bleu_3_score = sacrebleu.corpus_bleu(predictions, actuals, weights=(0.33, 0.33, 0.33)).score

print(f"BLEU-2 Score: {bleu_2_score}")
print(f"BLEU-3 Score: {bleu_3_score}")
```

Emotion Analysis Consistency

```
import pandas as pd
from transformers import pipeline

# 데이터 로드
file_path = 'predictions.csv'  # 예시 경로, 실제 파일 경로로 수정해야 합니다.
df = pd.read_csv(file_path)

# 감정 분석 파이프라인 설정
sentiment_pipeline = pipeline("sentiment-analysis")

# 감정 분석 함수 정의
def analyze_sentiments(texts):
    results = sentiment_pipeline(texts)
    return [result['label'] for result in results]

# 감정 분석 실행
df['Generated Sentiment'] = df['Generated Text'].apply(lambda x: analyze_sentiments([x])[0])
df['Actual Sentiment'] = df['Actual Text'].apply(lambda x: analyze_sentiments([x])[0])

# 일치도 평가
def evaluate_match(df):
    match_count = (df['Generated Sentiment'] == df['Actual Sentiment']).sum()
    total_count = len(df)
    match_rate = match_count / total_count
    return match_rate

match_rate = evaluate_match(df)
print(f"감정 일치도: {match_rate:.2%}")

# 선택적: 결과 확인
print(df[['Generated Sentiment', 'Actual Sentiment']].head())
```

평가방식 / 적용모델	BLEU Score (2-gram)	BLEU Score (3-gram)	감정분석 일치도
Transformer	8.61	7.97	52.71%
GPT-2	4.18	3.68	62.19%
BART	15.12	14.50	74.49%
제안한 방식	27.20	26.48	78.50%

Result

Made by장기태