KUBIG
DATA SCIENCE & AI

CLAP2Diff 오디오 조건부 이미지 생성 모델

Team | CV 4조 21기 남동연
22기 신진섭
22기 공병승

KUBIG
DATA SCIENCE & AI

CONTENTS

KUBIG
DATA SCIENCE & AI

01. 프로젝트 개요

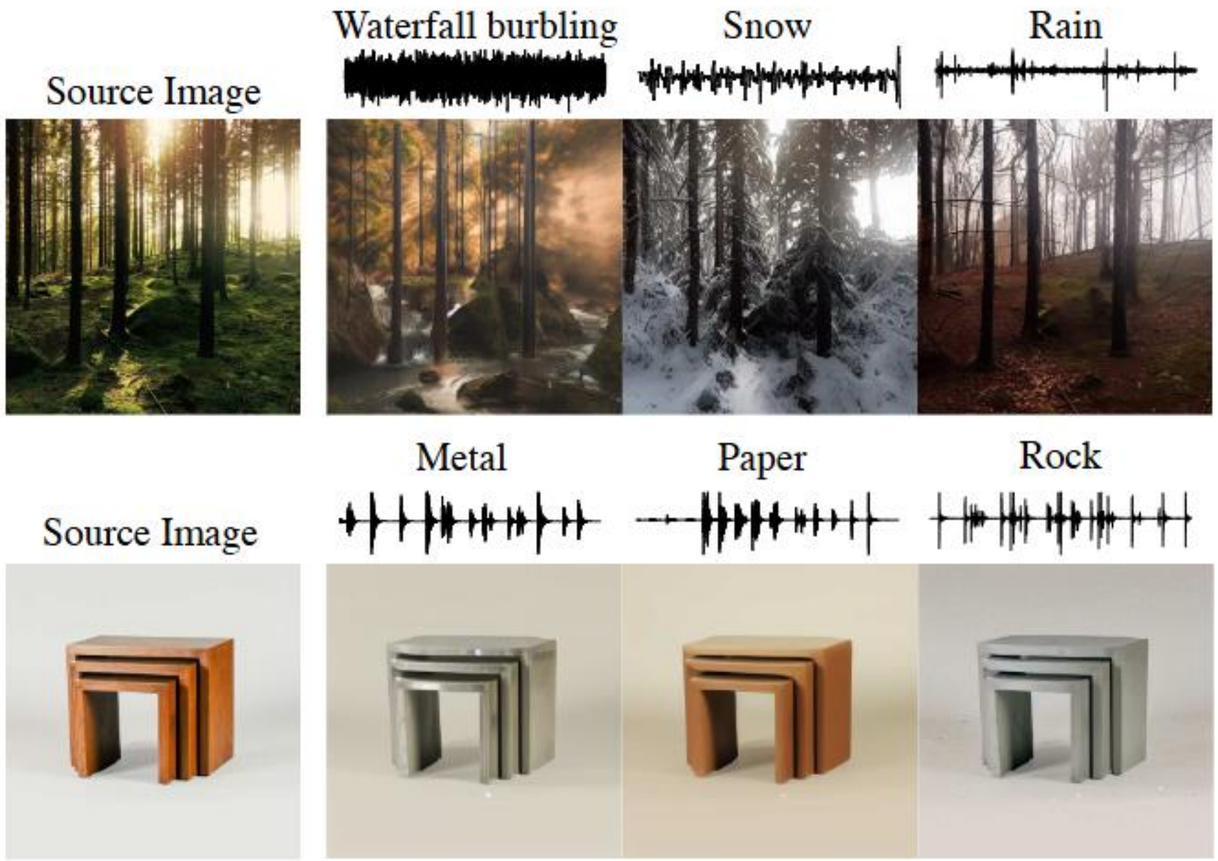
01. 프로젝트 개요

1. 오디오의 중요성

오디오는 텍스트가 놓치는

장면의 상태·시간 정보를

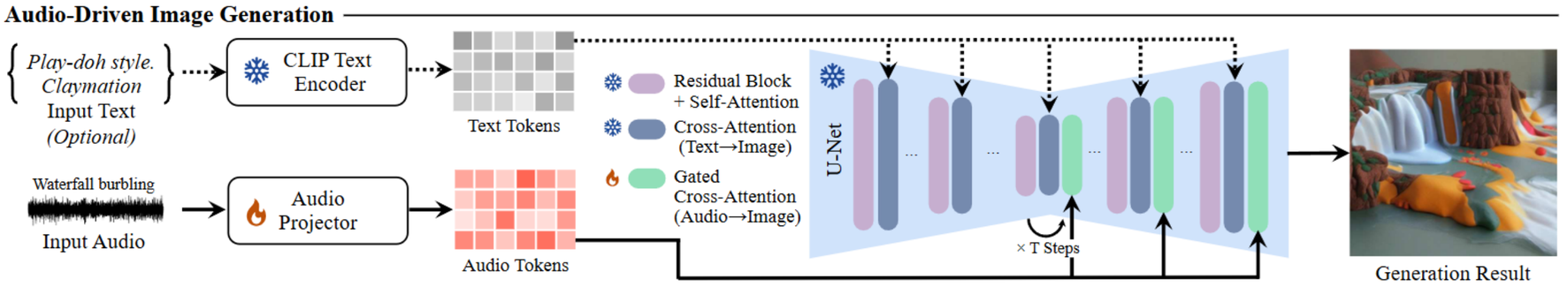
보완할수있다.



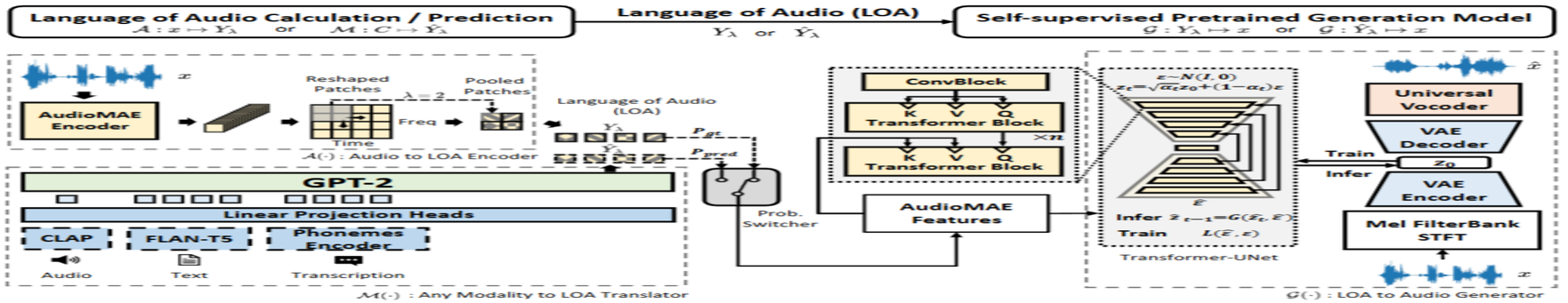
01. 프로젝트 개요

2. 모델 구조 참고

- SonicDiffusion: Audio-Driven Image Generation and Editing with Pretrained Diffusion Models (2023)



- AudioLDM2: Learning Holistic Audio Generation with Self-supervised Pretraining (2024)



KUBIG
DATA SCIENCE & AI

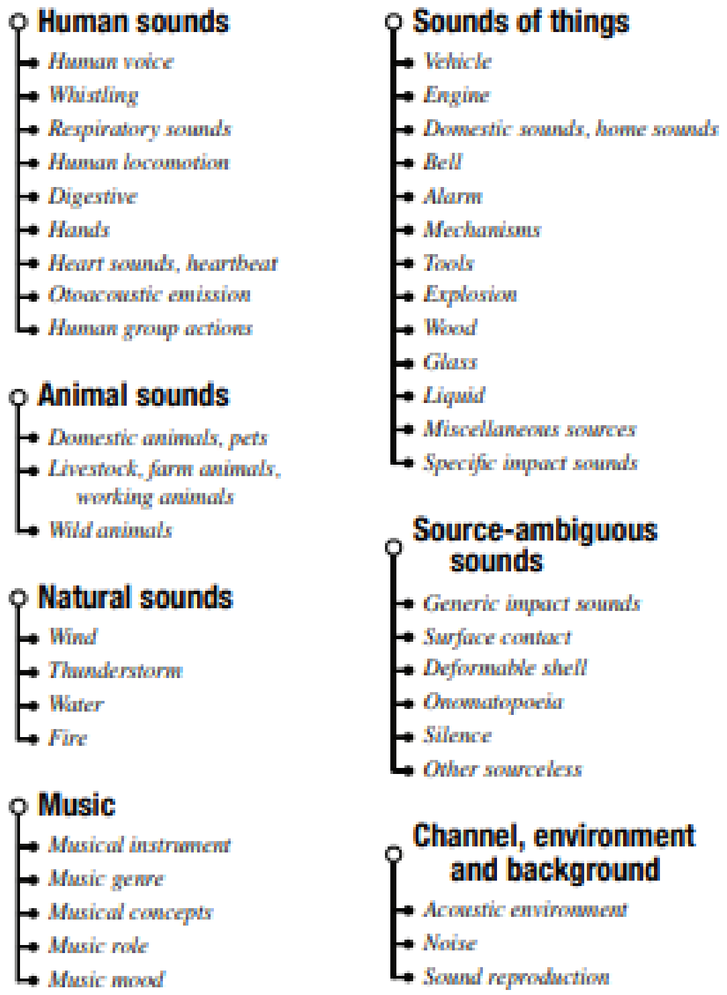
02. 프로젝트 내용

02. 프로젝트 내용

1. 데이터셋

- AUDIO SET: AN ONTOLOGY AND HUMAN-LABELED DATASET FOR AUDIO EVENTS(2017)
- 유튜브 영상 10s 분절 링크 + 오디오 클래스 데이터 구축

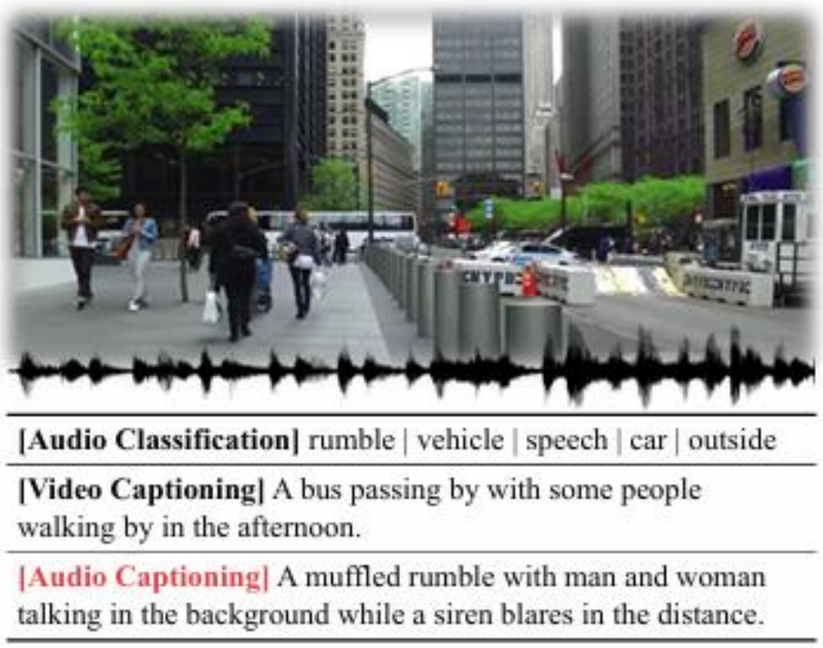
```
{
id: '/m/020bb7',
name: 'Bird vocalization, bird call, bird song',
description: 'Bird sounds that are melodious to
the human ear.',
citation_uri: 'http://en.wikipedia.org/wiki/Bird_
vocalization',
examples: 'youtu.be/vRg6EQm8pBw?start=30&end=40',
children: '/m/07pggtf,/m/07pggtn,/m/07sx8x_',
restrictions: ''
}
```



02. 프로젝트 내용

1. 데이터셋

- AudioCaps: Generating Captions for Audios in The Wild (2019) → 2025 New Data!
- Audio Set → AudioCaps : 10s 영상 + human-labeled 캡션 데이터 구축



02. 프로젝트 내용

1. 데이터셋

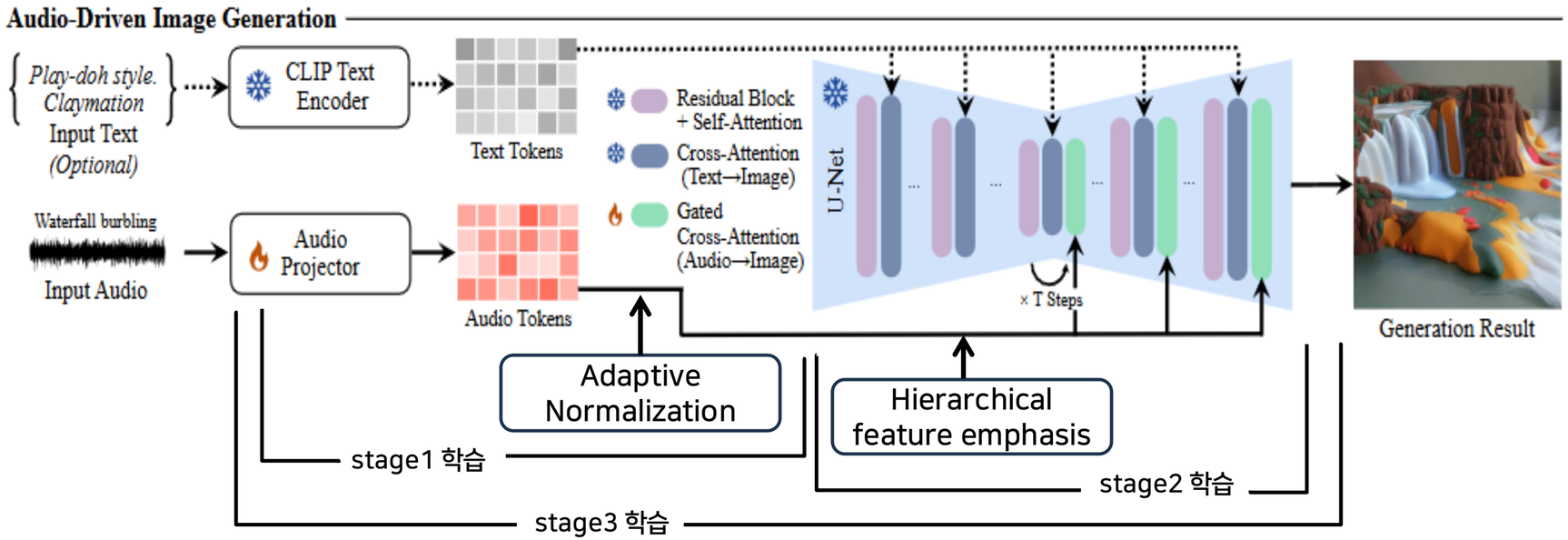
- Youtube 크롤링을 통해 해당 영상의 이미지 + 10s 오디오 데이터 추출 (yt-dlp / FFmpeg)
- 8500 쌍의 이미지 + 캡션(AudioCaps 제공) + 10s 오디오 데이터 확보

```python
def download_audio(self, audiocap_id, youtube_id, start_time):
    """Download 10-second audio clip from YouTube with 2025.08.20 improvements"""

    output_path = self.audio_dir / f"{audiocap_id}.wav"

    # Skip if already exists
    if output_path.exists():
        return True, "Already exists"

    url = f"https://www.youtube.com/watch?v={youtube_id}"
    end_time = start_time + 10

    # Try different player JS variants (2025.08.20 feature)
    player_variants = ['main', 'es6', 'es5']
    user_agents = [
        'com.google.android.youtube/19.45.36 (Linux; U; Android 12) gzip',
        'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
        'com.google.ios.youtube/19.45.38 (iPhone14,3; U; CPU iOS 17_6_1 like Mac O
    ]
```

```python
def extract_frame(self, audiocap_id, youtube_id, start_time):
    """Extract frame from YouTube video at specified time"""

    output_path = self.frames_dir / f"{audiocap_id}.jpg"

    # Skip if already exists
    if output_path.exists():
        return True, "Already exists"

    url = f"https://www.youtube.com/watch?v={youtube_id}"
    frame_time = start_time + 5.0  # Middle of 10-second clip

    try:
        # Get video URL using yt-dlp
        cmd = [
```

02. 프로젝트 내용

2. 전체 프로세스 과정



텍스트 → CLIP Text Encoder → U-Net

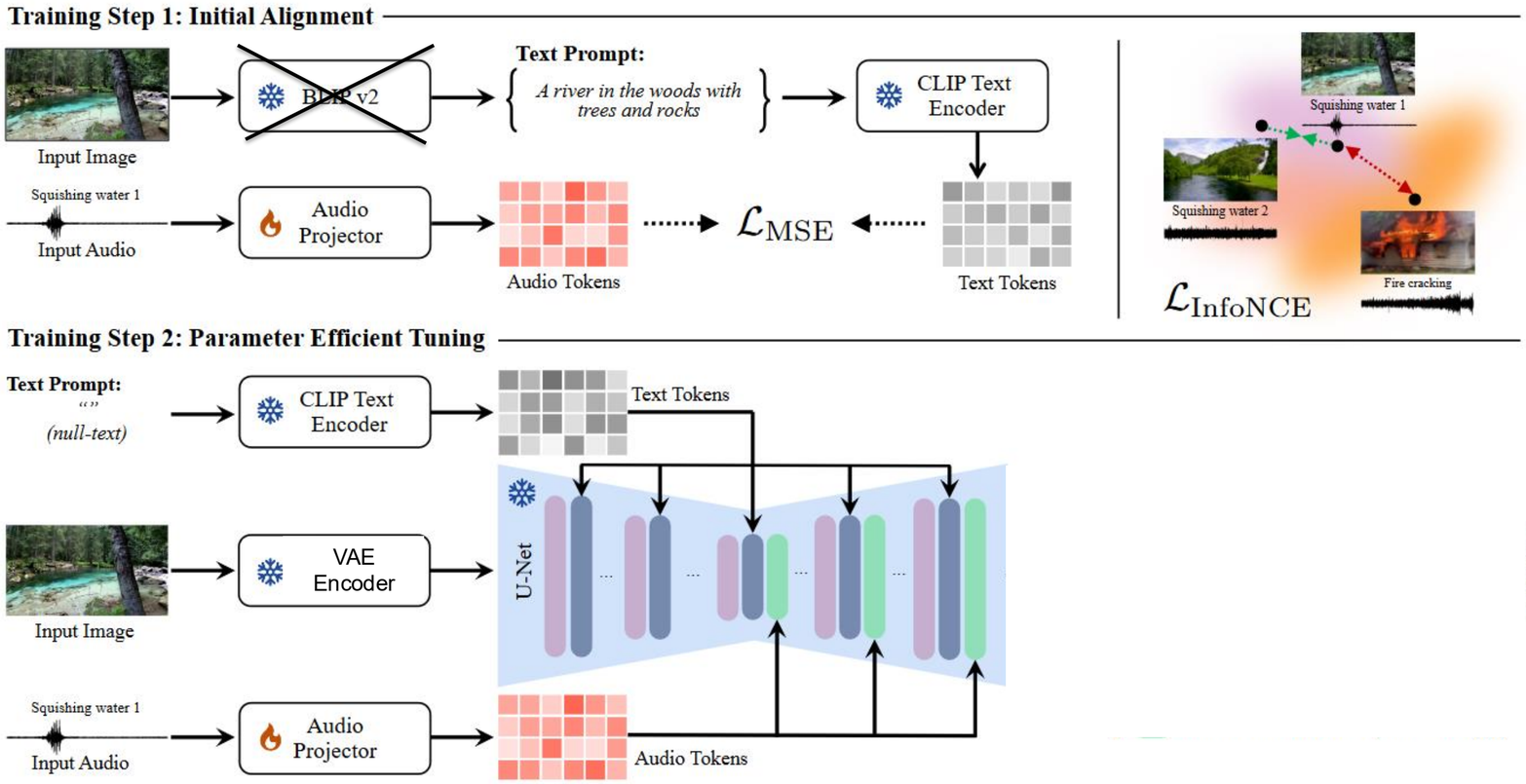
오디오 → Audio Projector (CLAP Audio Encoder + Adapter) → Adaptive Normalization

→ Hierarchical feature emphasis → U-Net

임베딩 된 텍스트와 오디오 토큰 → U-Net → VAE Decoder → Generation Result

02. 프로젝트 내용

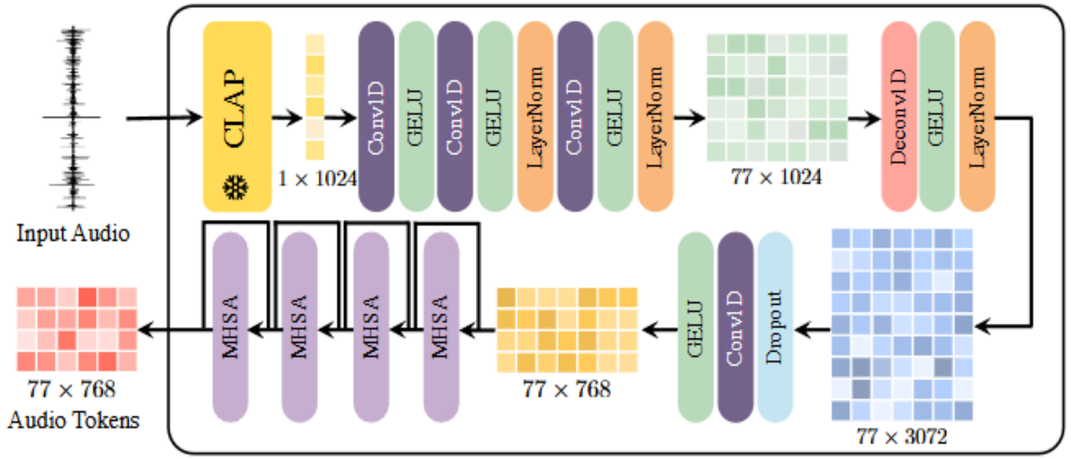
2. 전체 프로세스 과정



02. 프로젝트 내용

3. 각 프로세스 별 탐구

Audio Projector (CLAP Audio Encoder + Adapter)



(1) CLAP Audio Encoder

- 입력: [132,300] → 출력: [1, 1024]
- 1024차원 임베딩으로 압축

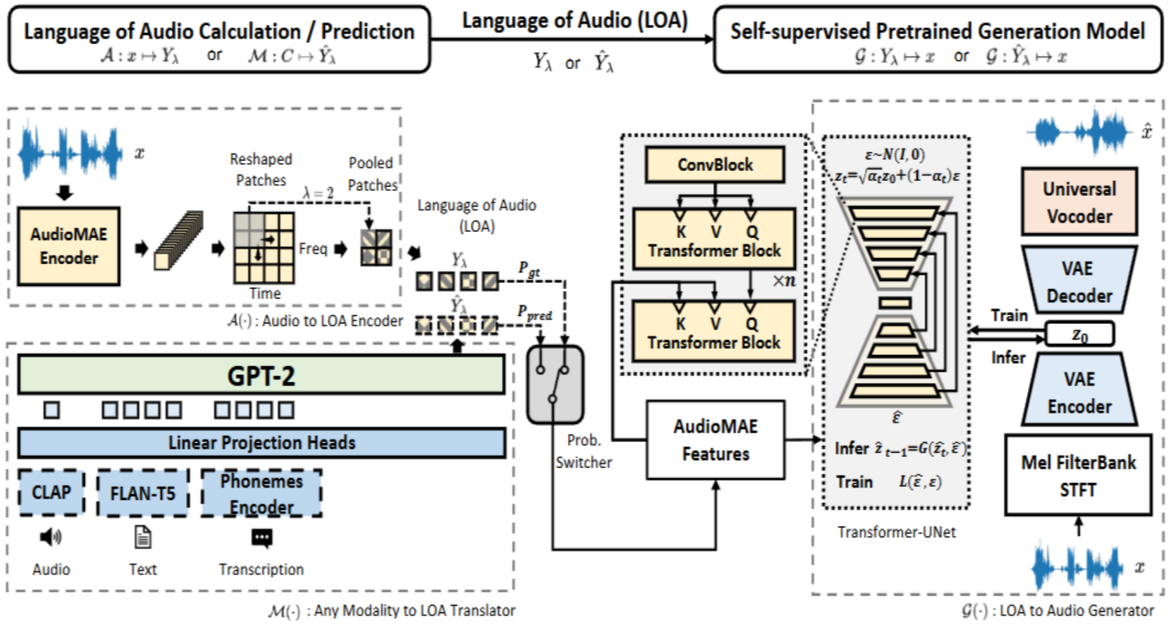
(2) Audio Projection

- 입력: [1, 1, 1024] → 출력: [1, 77, 768]
- Conv1D(1024 → 768), Deconv1D(768 → 77×3072),
  Conv1D(77×3072 → 77×768),
  4× TransformerBlock with Self-Attention
- CLAP 임베딩을 SD 텍스트 토큰 공간으로 변환

02. 프로젝트 내용

3. 각 프로세스 별 탐구

Hierarchical feature emphasis



(1) Hierarchical feature 의 필요

Why? 혼합된 오디오 데이터

- "Woman laughing and cat meowing"
- "Thunder with rain and wind"
- 데이터셋의 약 62%가 복합 오디오
- 평균 2.3개의 독립적 음원 포함

(2) AudioLDM2 의 구조 참고

AudioMAE → CLAP → FLAN-T5

(acoustic feature) (Semantic) (Detail)

(3) Hierarchical feature emphasis

Low-freq → Mid-freq → High-freq

(Ambience) (background) (Foreground)

02. 프로젝트 내용

3. 각 프로세스 별 탐구

Hierarchical feature emphasis

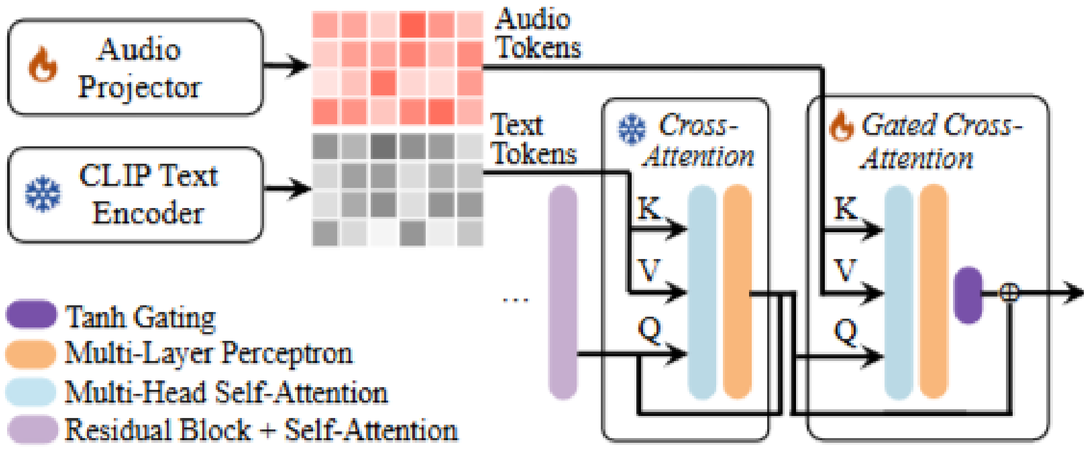
```python
def hierarchical_audio_processing(audio):

    # CLAP Encoding
    audio_emb = clap.encode(audio)  # [1, 1024]

    # Audio Projection (77 tokens)
    audio_tokens = projector(audio_emb)  # [1, 77, 768]
    # 원본 norm: 154.3 (문제!)

    # Normalization (Sweet Spot 발견!)
    current_norm = audio_tokens.norm()  # 154.3
    target_norm = 60.0  # 실험으로 발견한 최적값
    scale_factor = target_norm / current_norm  # 0.3888
    audio_tokens = audio_tokens * scale_factor

    # Hierarchical Decomposition (V4 실제 구조)
    # 77 tokens → 3 levels (soft assignment)
    audio_features = {
        'foreground': audio_tokens[:, :35, :],   # 35 tokens - 주요 소리
        'background': audio_tokens[:, 35:56, :], # 21 tokens - 배경
        'ambience': audio_tokens[:, 56:, :]      # 21 tokens - 분위기
    }

    # Level별 가중치 적용 (학습된 값)
    audio_features['foreground'] = audio_features['foreground'] * 0.7  # 강한 영향
    audio_features['background'] = audio_features['background'] * 0.5  # 중간
    audio_features['ambience'] = audio_features['ambience'] * 0.3     # 약한 영향

    return audio_features
    # 출력: 3개 레벨 dictionary
```

```python
def timestep_aware_generation(latent_noise, text, audio_features):
    """
    Diffusion process: T=50 (noise) → T=0 (image)
    각 timestep에 다른 audio level 사용
    """

    x_t = latent_noise  # [1, 4, 64, 64]

    for t in range(50, 0, -1):
        # Timestep별 audio feature 선택
        if t > 30:  # 초기: 전체 구조 (60%)
            audio_context = audio_features['ambience']
            # 어두운 톤, 공간감 설정

        elif t > 15:  # 중간: 주요 객체 (30%)
            audio_context = audio_features['background']
            # 파도, 구름 형태

        else:  # 후기: 세부사항 (10%)
            audio_context = audio_features['foreground']
            # 번개, 빗방울 텍스처

        # UNet with Audio Cross-Attention
        noise_pred = unet_with_audio_adapters(
            x_t, t, text, audio_context
        )
        x_t = scheduler.step(noise_pred, t, x_t)

    return x_t  # [1, 4, 64, 64] clean latent
```

02. 프로젝트 내용

3. 각 프로세스 별 탐구

U-Net



(1) Cross-Attention

- 다른 데이터 간 관계 학습
  (이미지↔텍스트, 이미지↔오디오)

(2) Gated Cross-Attention

- 문제: 오디오를 얼마나 반영해야 할까?
  너무 약하면 → 오디오 효과 없음
  너무 강하면 → 이미지 품질 저하
- 해결: 학습 가능한 Gate로 자동 조절
- 장점: 자동 균형 조절, 과도한 영향 방지, 이미지 품질 유지
  빠른 학습

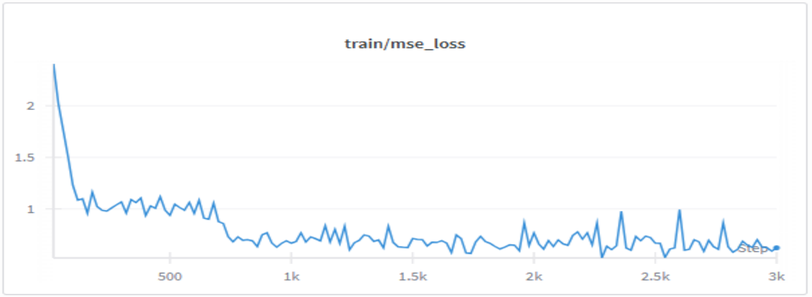
KUBIG
DATA SCIENCE & AI

03. 프로젝트 결과

03. 프로젝트 결과

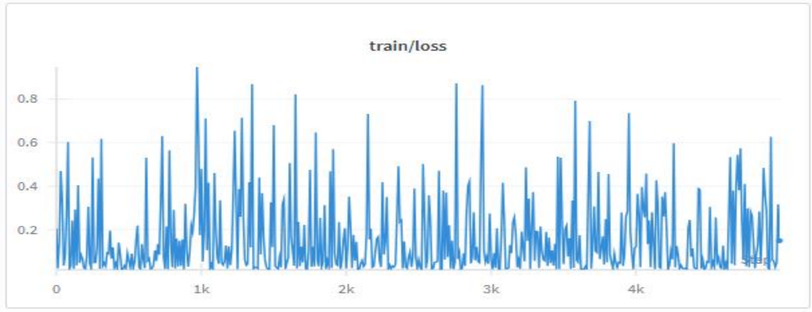
1. 학습 후 로스 값 비교

Stage 1 (Audio Projection):
- 초기: 1.82
- 1000 steps: 0.94
- 3000 steps: 0.67



Stage 2 (Hierarchy + U-Net):
- 초기: 0.52
- 1000 steps: 0.38
- 2000 steps: 0.31

Stage 3 (Fine-tuning):
- 초기: 0.20
- 2500 steps: 0.18
- 5000 steps: 0.14



03. 프로젝트 결과

2. Gradio 구현

03. 프로젝트 결과

3. 이미지 생성 결과 (성공)

입력 오디오

Rainy and Thunder

입력 텍스트

a city

이미지 생성

03. 프로젝트 결과

3. 이미지 생성 결과 (성공)

입력 오디오

Rainy and Thunder

입력 텍스트

a beach

이미지 생성

03. 프로젝트 결과

3. 이미지 생성 결과 (성공)

입력 오디오

Rainy and Thunder

입력 텍스트

a forest

이미지 생성

03. 프로젝트 결과

3. 이미지 생성 결과 (실패)

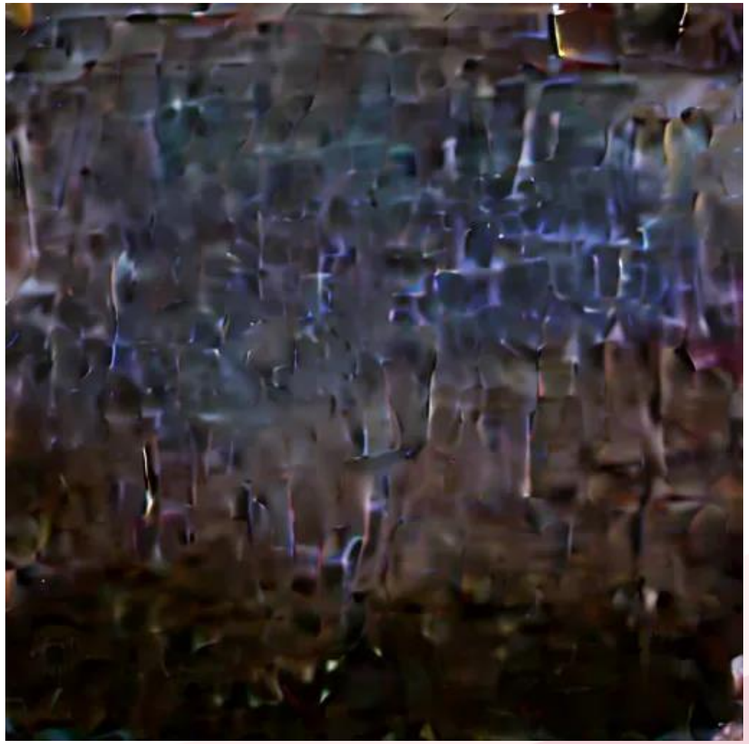
입력 오디오

Rainy and Thunder

입력 텍스트

없음

이미지 생성



03. 프로젝트 결과

3. 이미지 생성 결과 (실패)

입력 오디오

a person talks, followed by a baby laughing, which is then followed by a person laughing, and a baby

입력 텍스트

a city

이미지 생성

03. 프로젝트 결과

3. 이미지 생성 결과 (실패)

입력 오디오

an adult male is laughing and a motor vehicle engine is running and is revved

입력 텍스트

a beach

이미지 생성

03. 프로젝트 결과

4. 한계점 및 개선 방향

한계점

(1) 실제 Source Separation 없음

- hierarchical feature 일 뿐, 실제로 오디오를 분리하지 못함
- 같은 77개 토큰을 다르게 가중치만 적용
- 복잡한 혼합 오디오에서 제한적

(2) 데이터셋 크기 및 품질

- AudioCaps 8,500개만 사용 (매우 작음)
- 복잡한 오디오 혼합, 단일 소리 학습 어려움
- Zero-shot 성능 제한적

개선 방향

(1) 실제 Source Separation 통합

- Demucs나 Spleeter 사용

(2) 데이터 증강

- AudioSet (200만개) 활용
- Mix-up augmentation
- Frequency masking

(3) 품질 향상

- 여러 가중치 값에 따른 실험 필요

KUBIG
DATA SCIENCE & AI

Thank You