# Find the clips you want: based on text

## Tving Hackathon Wrap-up Report

**Hyeonwoo Jung [1] Jaehoon Choi [1] Jiwan Park [1]**
**Chanhyeok Leem [1] Changgi Min [1] Danyou Lee [1]**

## 1. 프로젝트 개요

최근 미디어 소비 방식의 변화로 인해 방대한 영상 데이터에서 원하는 장면을 효율적으로 탐색하고 검색하는 기술의 필요성이 증가하고 있다. 특히, OTT(Over-The-Top) 플랫폼과 같은 서비스에서는 사용자 경험을 극대화하기 위해 영상 데이터의 검색 및 관리 기술이 필수적이다.

영상 데이터는 본질적으로 대용량이며, 효율적인 검색 및 저장이 어려운 특성을 가진다. 일반적으로 특정 정보를 찾기 위해 전체 영상을 탐색해야 하는 경우가 많아 메모리 사용 및 검색 속도에서 비효율적이다. 이러한 문제를 해결하기 위해 본 프로젝트에서는 비디오 캡셔닝(Video to Text) 및 비디오 검색(Text to Video) 기술을 활용하고자 한다.

Video to Text 기술은 영상의 주요 장면을 텍스트로 변환함으로써 콘텐츠를 구조화하고 저장하는 것을 목표로 한다. 이를 통해, 대용량 영상 데이터를 더욱 효율적으로 관리할 수 있으며 키워드 기반 검색도 가능하여 메모리 사용을 최적화하고 검색 속도를 향상할 수 있다.

Video to Text 와 Text to Video 기술은 상호 연계되어 영상과 텍스트 간의 효율적인 검색 및 매칭을 가능하게 한다. Video to Text 기술이 영상 내용을 텍스트로 변환하여 데이터베이스화한다면, Text to Video 기술은 사용자가 입력한 특정 텍스트를 기반으로 해당 내용을 포함하는 영상 구간을 검색할 수 있도록 한다. 이 과정에서, 입력된 텍스트와 영상은 임베딩 공간에서 매칭되며, 문장과 의미적으로 유사한 비디오 클립을 검색하는 방식으로 동작한다. 즉, 사용자가 특정 텍스트를 입력하면, 데이터베이스에 저장된 영상 중 해당 텍스트와 가장 높은 유사도를 가지는 영상 구간을 찾아 제공한다.

## 2. 프로젝트 팀 구성 및 역할

| | |
|---|---|
| **정현우** | 전체 파이프라인 및 실험 설계 / Video-to-Text 코드 구현 / 코드 최적화 |
| **최재훈** | 모델 탐색 및 전략 수립 / 검색 성능 평가 실험 / 최종 파이프라인 구축 및 정리 |

| | |
|---|---|
| **박지완** | Text-to-Video : embedding, retrieval 코드 구현 / web demo 구현 |
| **임찬혁** | 데이터 전처리 / captioning 및 query DB 구축 / embedding 모델 fine-tuning 실험 |
| **민창기** | 모델 탐색 / 모델 훈련 및 평가 / 최종 파이프라인 구축 / 오디오 캡셔닝 및 자막 모델 실험 |
| **이단유** | Video-to-Text : 캡셔닝 모델 및 프롬프트 실험 / 입력 query 생성 |

# 3. 프로젝트 수행 절차 및 방법

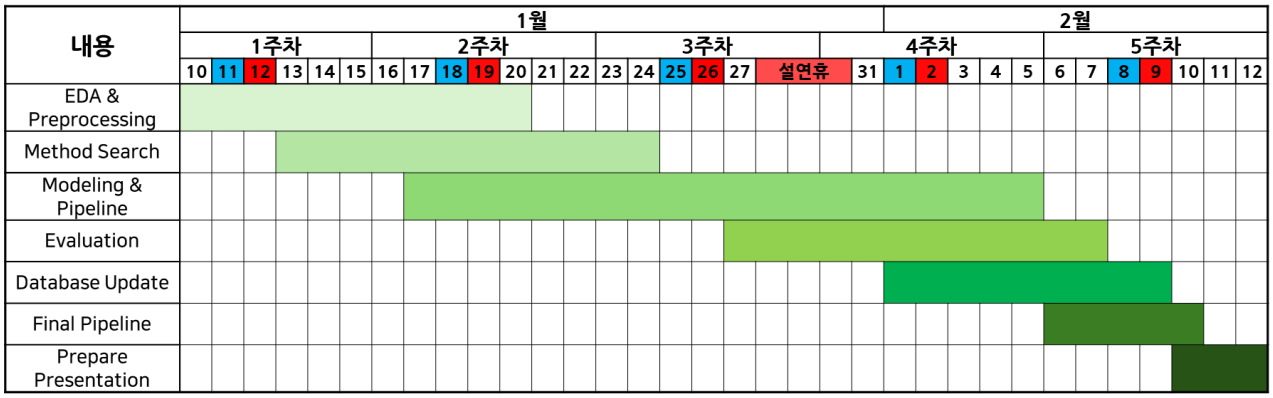


Fig 1. 프로젝트 수행 절차 간트차트

## 3.1. 프로젝트 파이프라인

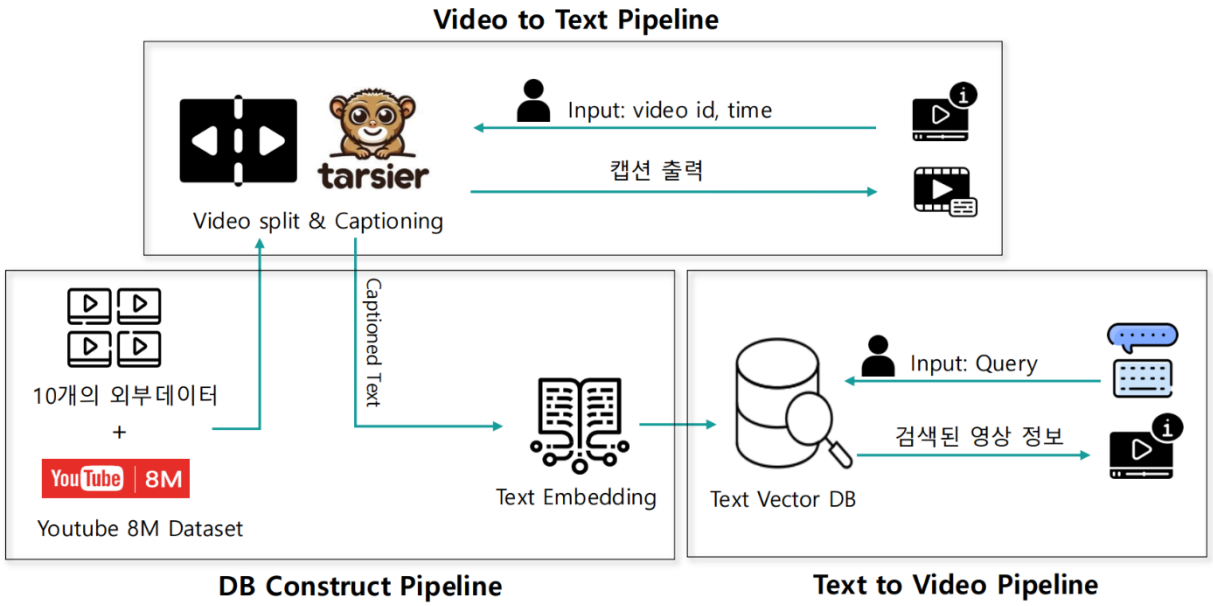


Fig 2. 프로젝트 파이프라인

# 4. 프로젝트 수행 결과

## 4.1. 데이터

### 4.1.1. YouTube-8M 데이터 수집

YouTube-8M [1]은 Google 에서 공개한 대규모 비디오 데이터셋으로, 800 만 개 이상의 YouTube 동영상에 대해서 원본 비디오 없이 메타데이터와 feature embeddings 를 포함하고 있다. 우리는 YouTube-8M 데이터셋 중 Movieclips 카테고리로 분류된 1,683 개의 데이터 중 원본 비디오 다운로드가 가능한 1,218 개 비디오를 사용했다.

Srinivas Rao 의 github*를 참고하여 Movieclips 카테고리에 해당하는 url 을 추출하여 원본 비디오를 다운로드했다. Movieclips 영상의 경우 검색 성능에 영향이 갈 수 있는 후반 30 초 엔딩 크레딧을 잘라서 사용했다. 다운로드된 원본 비디오를 clip 단위로 분할한 후 video_id, title, url, fps, start_time, end_time 등의 정보를 annotation file 로 기록하여 사용했다. (*https://github.com/gsssrao/youtube-8m-videos-frames)

Table 1. YouTube-8M Moviclips 1218 개 원본 비디오 정보

| | Duration (sec) | FPS | Width (pixel) | Height (pixel) |
|---|---|---|---|---|
| Mean | 166.62 | 24.11 | 638.94 | 358.54 |
| Min | 97.62 | 18.00 | 320.00 | 240.00 |
| Max | 487.94 | 30.00 | 640.00 | 360.00 |

Table 1 을 보면 다운로드한 비디오들에 대한 평균 길이는 166.62 초로 3 분 미만이고 해상도는 638.94 x 358.54 로 낮은 해상도를 가짐을 알 수 있다. 비디오의 평균 길이는 데이터 처리 시간 예측에 사용하였고, 평균 FPS 는 일관된 프레임 유지 및 프레임 분할, 평균 해상도는 모델 입력 크기 조정 및 메모리 최적화에 활용되었다.

### 4.1.2. 평가용 쿼리 데이터 생성

자체 평가용 쿼리 생성은 모델의 검색 성능을 평가하기 위해 수행됐다. 기존의 캡션과 임베딩을 기반으로만 평가할 경우, 모델이 실제 사용 환경에서 얼마나 적절하게 검색 결과를 반환하는지 판단하기 어렵다. 따라서 사람이 직접 쿼리를 생성하여 모델이 실제 질의에 대해 적절한 비디오를 검색할 수 있는지를 검증했다.

Movieclips 영상 중에서 다양한 장르를 포함하도록 187 개를 샘플링해 평가용 데이터를 구성했다. 샘플링된 비디오 클립을 대상으로 사람이 짧은 캡션부터 자세한 캡션까지 다양하게 직접 평가용 쿼리를 생성하고, 이를 바탕으로 위와 같은 형태로 Ground Truth(GT)를 구성했다.

Table 2. 평가용 쿼리 예시

| video_id | start_time | end_time | query |
|---|---|---|---|
| 11 | 0 | 6 | 노인이 북을 치는 장면 |
| 27 | 108 | 115 | 검은 선글라스를 낀 남자가 쏘던 총을 버리고 점프 한 뒤 다른 남자의 머리를 발로 차는 장면 |
| 33 | 37 | 46 | 헤드셋을 끼고 있는 복고 옷을 입고 있는 남성이 한 손에는 스케이트 보드를 들고 있다. 이 남성은 철창문을 발로 열고 닫으려다 안 닫혀 스케이트 보드를 던지고 떠나는 장면 |

## 4.2. Video to Text

### 4.2.1. Video to Text Pipeline

비디오 캡셔닝은 비디오의 내용을 빠르게 이해하고 메모리를 효율적으로 저장하기 위해 필수적인 과정이다. 이를 통해 사용자는 특정 비디오의 원하는 구간에 대한 설명을 손쉽게 생성할 수 있으며, 캡션 데이터는 다양한 활용 가능성을 제공한다. Fig 3 은 비디오 QA 모델을 활용한 비디오 캡셔닝 파이프라인을 나타낸다. 사용자가 원하는 비디오를 특정 구간으로 분할한 후, 비디오 QA 모델에 "Describe this video in detail"이라는 프롬프트와 함께 입력하면, 모델이 해당 비디오에 대한 상세한 설명을 생성한다.

비디오 QA 모델은 단순히 정해진 방식으로 캡션을 생성하는 것이 아니라, 사용자의 프롬프트 입력을 통해 유연하게 응답을 조정할 수 있도록 설계되었다. 예를 들어:

- **영상의 전체적인 맥락이 필요할 경우** → "Describe this video in detail"
- **특정 객체나 장면에 초점을 맞추고 싶을 경우** → "What is happening with the main character in this scene?"
- **비디오의 감정적 분위기를 분석하고 싶을 경우** → "Describe the mood and emotions in this video."

사용자는 위와 같이 프롬프트를 조정하며, 원하는 비디오의 구간을 캡셔닝할 수 있고, 이를 통해 비디오를 요약할 수 있다.

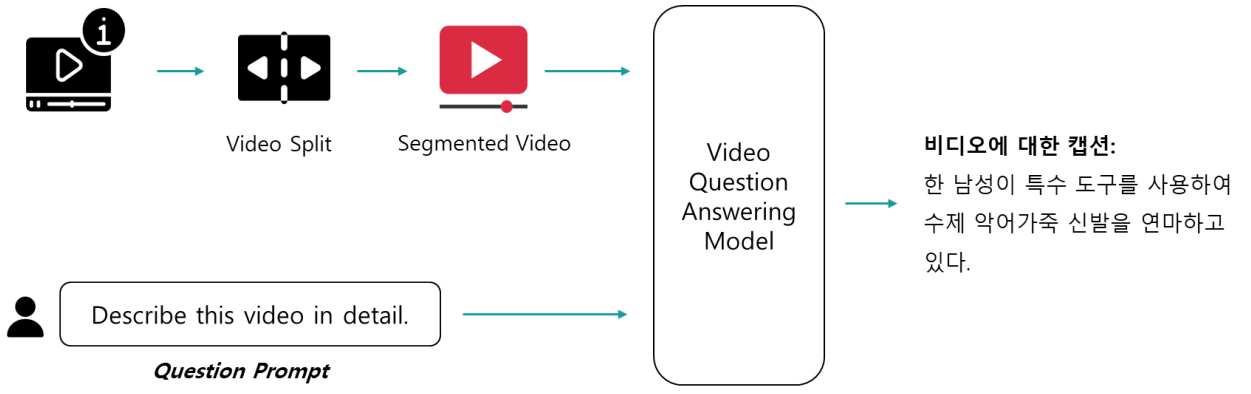


Fig 3. 비디오 캡셔닝 파이프라인

### 4.2.2. 모델별 Captioning 성능 비교

비디오 캡셔닝 성능을 비교하기 위해 자체적인 정성 평가 기준을 설정하여 평가를 진행하였다. 평가에는 YouTube-8M 데이터셋에서 무작위로 선택된 5 초 이내의 비디오 클립이 사용되었으며 mPLUG-Owl3 [2], InternLM-XComposer [3], Tarsier-7B [4] 세 가지 모델에 대해서 캡션 결과를 비교했다.

총 5 인의 평가자가 참여하여 개별적으로 평가한 후, 평균 점수를 산출하여 최종 평가 결과를 도출했다.

모델별 캡셔닝 성능을 비교하기 위해 다음과 같은 세 가지 정성 평가 지표를 설정했다.

- **정확성(5 점):** Caption 이 영상을 얼마나 정확하게 설명하는지 평가
- **포괄성(5 점):** [장소], [배경], [인물 외형], [인물 동작], [인물 간 상호 동작] 등의 정보 포함 여부
- **간결성(5 점):** Caption 길이에 대한 필요한 정보의 비율, 얼마나 간결하게 영상을 설명하는지 평가

이러한 정성 평가 점수와 함께 캡션 생성에 소요된 시간(초 단위)도 측정하여 모델의 실용성을 고려했다.
아래 Table 3 은 세 모델의 정성 평가 및 캡셔닝 속도를 비교한 결과를 나타낸다.

Table 3. 비디오 캡셔닝 성능 비교

| 모델 | 정성 평가 | | | | 정량 평가 |
|---|---|---|---|---|---|
| | 정확성 ↑ | 포괄성 ↑ | 간결성 ↑ | 평균 ↑ | 소요시간(초) ↓ |
| mPLUG-Owl3 | 1.4 | 1.0 | 2.0 | 1.3 | **5.43** |
| InternLM-XComposer | 3.4 | **4.8** | 4.0 | 4.13 | 29.37 |
| Tarsier-7B | **4.2** | 4.2 | **4.8** | **4.3** | 6.76 |

평가 결과, Tarsier-7B 모델이 정확성(4.2), 포괄성(4.2), 간결성(4.8)에서 가장 높은 평균 점수(4.3)를 기록하면서도 캡션 생성 속도(6.76 초)가 상대적으로 빠른 것으로 나타난다. 따라서, 정성 평가와 캡션 생성 속도를 종합적으로 고려하여 최종적으로 Tarsier-7B 모델을 비디오 캡셔닝 모델로 선정했다.

### 4.2.3. 최적화

앞서 선택된 Tarsier-7b 모델의 효율성을 극대화하기 위해 배치 처리, 샘플링 프레임 수 조절, 서버 분산 처리의 최적화 기법을 적용했다. 실험은 총 3,637 초의 비디오를 대상으로 진행되었으며, 각 5 초 단위로 분할된 총 728 개의 클립을 캡셔닝하는 소요시간을 측정하여 실험을 진행했다. 결과적으로, 기존 대비 캡셔닝 처리 속도를 82.27% 향상시킬 수 있었다.

#### 서버분산처리

비디오 캡셔닝 과정에서 병목 현상을 최소화하고 처리 속도를 향상하기 위해 서버 분산 처리 기법을 적용했다.

실험에서는 파이프라인을 관리하는 1 개의 메인서버와 캡셔닝을 진행하는 4 개의 서브서버로 구성하여 병렬처리를 수행했다. 메인서버에서 입력된 비디오를 5 초 단위 클립으로 분할한다. 4 개의 서브서버는 병렬적으로 분할된 클립에 대한 캡셔닝을 진행한 후 메인서버에 결과를 전송한다. 이와 같은 분산 처리 기법을 적용한 결과, 서버를 4 개로 확장했을 때 단일 서버 대비 처리 속도가 70.45% 향상되었음을 확인할 수 있었다.

**배치처리**

비디오 캡셔닝을 수행할 때, 배치 처리를 활용하여 여러 개의 비디오 클립을 동시에 처리함으로써 전체 소요 시간을 단축하는 최적화 기법을 적용했다. 실험 결과, 배치 크기를 증가시킬수록 캡셔닝 속도가 개선됨을 확인할 수 있었다. 특히, 배치 크기 4 를 적용했을 때 단일 처리 방식 대비 처리 속도 향상이 이루어졌다.

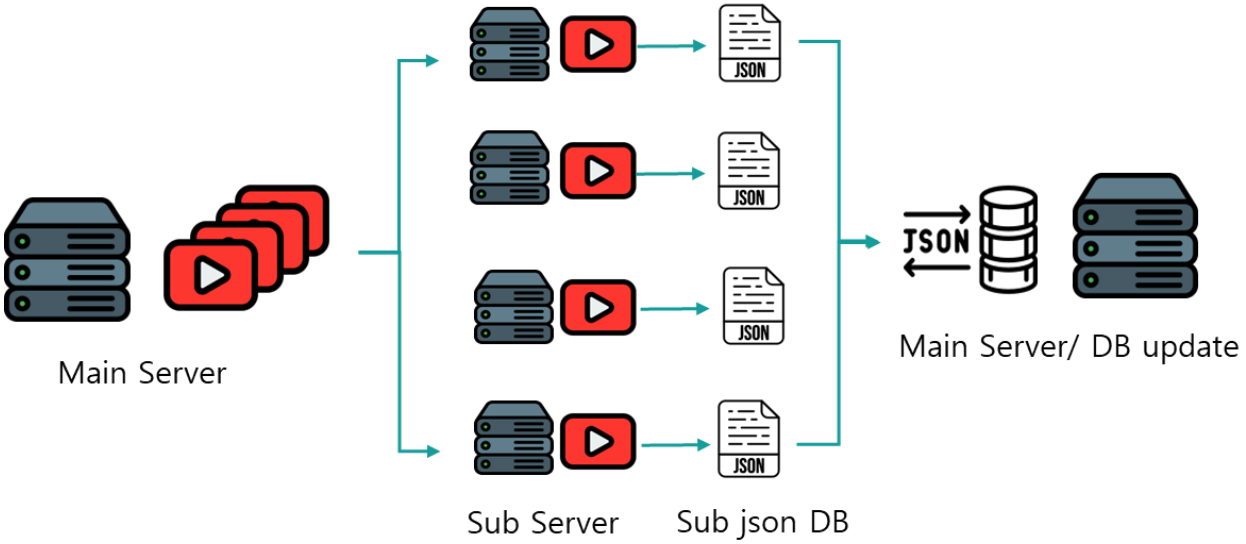


Fig 4. 분산 병렬 처리 파이프라인

**샘플링프레임 수 조절**

Tarsier-7B 모델은 입력 비디오에서 샘플링한 8 개의 프레임을 기반으로 캡셔닝을 수행한다. 그러나, Tarsier-7B 의 훈련 데이터인 WebVid-10M 데이터셋[5]의 평균 비디오 길이는 18 초이며, 이는 본 연구에서 사용한 5 초 단위 비디오 클립과 차이가 있다. 이러한 차이를 고려하여, 5 초 비디오 클립에 최적화된 샘플링 프레임 수를 조정하는 실험을 진행했다. 샘플링 프레임 수가 줄어들수록 처리 속도 향상을 확인할 수 있었다.

Table 4. 최적화 기법에 따른 소요시간 및 속도향상 정도

| 서버 개수 | 배치 크기 | 샘플링 프레임 수 | 소요시간 (초) | 속도향상(%) |
|---|---|---|---|---|
| 1 | 1 | 8 | 2245.55 | 0 |
| 4 | 2 | 8 | 663.53 | 70.45 |
| 4 | 2 | 8 | 548.37 | 75.58 |
| 4 | 2 | 6 | 454.1 | 79.78 |
| **4** | **4** | **6** | **398.12** | **82.27** |

## 4.3. Text to Video

### 4.3.1. Text to Video pipeline

비디오 검색은 사용자가 원하는 정보를 포함하는 비디오 클립을 신속하게 찾아주는 과정이다. 오늘날 비디오 데이터는 방대한 양으로 축적되고 있으며, 이를 효율적이고 정확하게 검색하는 것이 필수적이다. 단순히 제목이나 태그를 기반으로 검색하는 방식에서 벗어나, 비디오 내용 자체를 분석하여 사용자의 질의와 가장 유사한 클립을 찾아내는 것이 핵심이다.

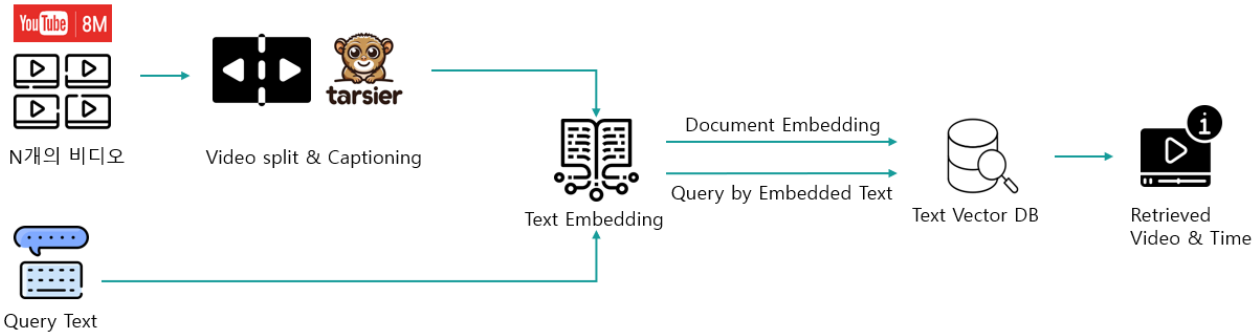


Fig 5. 비디오 검색 파이프라인

Fig 5 는 우리가 사용하는 비디오 검색 방식을 시각적으로 보여주는 파이프라인이다. 이 과정에서는 대규모 비디오 데이터셋인 YouTube-8M 과 새롭게 수집된 N 개의 비디오 데이터를 검색 데이터베이스로 활용한다. 먼저, 비디오를 일정한 길이의 클립으로 분할하는데, 이를 위해 FFmpeg 과 같은 도구를 사용한다. 이후, 각 클립의 내용을 설명하는 캡션을 생성하는 캡셔닝 파이프라인이 적용된다.

생성된 캡션은 트랜스포머 기반의 문장 임베딩 모델을 통해 벡터로 변환되며, 사용자의 검색 쿼리 역시 동일한 방식으로 임베딩된다. 이때, FAISS(Facebook AI Similarity Search) [6] 벡터 검색 엔진을 활용하여 가장 유사한 캡션을 찾아내고, 해당 클립과 그 시간 정보를 반환한다. 이 과정을 통해 사용자는 원하는 비디오 클립을 빠르고 정확하게 검색할 수 있다.

### 4.3.2. Retrieval 성능 평가 지표

비디오 검색 시스템이 효과적으로 동작하는지 확인하려면, 검색된 결과가 실제 사용자 요구를 얼마나 잘 충족하는지 평가하는 과정이 필수적이다. 검색 모델을 평가함으로써, 개선이 필요한 부분을 명확하게 파악하고 성능을 향상할 수 있다.

검색 성능을 평가하는 대표적인 지표 중 하나는 Recall@N 이다. 이는 검색된 상위 N 개 결과 중 정답을 포함하는 쿼리의 비율을 측정하는 지표로, 사용자의 질의와 가장 유사한 결과들이 얼마나 정확하게 검색되었는지를 평가하는 기준이 된다. Recall@N 의 수식은 다음과 같다.

$$Recall@N = \frac{정답을\ 포함한\ 쿼리의\ 개수(상위 N 개\ 결과\ 내)}{전체\ 쿼리\ 개수}$$

평균 유사도(TOP-1)는 검색된 최상위 결과(Top-1 캡션)와 입력된 쿼리 간의 유사도를 측정하여 평균값을 계산하는 지표이다. 이는 검색 모델이 반환하는 최상위 결과가 사용자 의도와 얼마나 정확하게 일치하는지를 정량적으로 평가하는 데 활용된다. 평균 유사도 수식은 다음과 같다.

$$평균\ 유사도 = \frac{\sum(Top-1\ 검색\ 결과와\ 입력\ 쿼리의\ 유사도)}{전체\ 쿼리\ 개수}$$

평균 순위(Mean Rank)는 검색 결과에서 정답이 나타나는 평균적인 위치를 측정하는 지표이다. 즉, 사용자의 질의에 대해 검색된 결과 목록에서 정답이 몇 번째 위치에 등장하는지를 평가한다. 평균 순위 값이 낮을수록 정답이 검색 결과의 상위에 위치한다는 의미이며, 이는 검색 시스템이 더욱 정확하고 효율적으로 작동함을 나타낸다. 평균 순위 수식은 다음과 같다.

$$평균\ 순위 = \frac{\sum(입력\ 쿼리에\ 대한\ 정답의\ 순위)}{전체\ 쿼리\ 개수}$$

이러한 평가 지표들은 검색 시스템의 성능을 다각도로 분석하는 데 중요한 역할을 한다. Recall@N 은 검색된 결과가 사용자 의도와 얼마나 일치하는지를 측정하며, 이는 검색 시스템의 품질을 판단하는 핵심 요소이다. 반면, 평균 유사도(Top-1)는 검색된 최상위 결과의 직관적인 정확도를 평가하며, 검색 모델이 신뢰할 만한 결과를 반환하는지 확인하는 데 도움을 준다. 평균 정답 순위는 검색된 결과에서 정답이 평균적으로 몇 번째 위치에 등장하는지를 평가하는 정확도 지표로 활용된다. 이러한 평가 기준을 통해 검색 시스템의 성능을 지속적으로 개선하고 최적화할 수 있으며, 보다 정교하고 정확한 검색 기술을 개발하는 데 기여한다.

### 4.3.3. Text embedding 모델 비교

Retrieval 성능에서 중요한 요소 중 하나는 Text embedding 모델의 성능이다. Table 5 에 있는 Text embedding 모델들은 자연어를 벡터로 변환하는 모델로 문장의 의미를 보존하면서 자연어를 벡터 공간에서 표현할 수 있다. Retrieval task 에서 Text embedding 모델을 사용하면 단순히 텍스트를 사용할 때보다 벡터 연산을 통해 검색 속도가 빨라지며 단순 키워드 매칭이 아니라 의미상으로 유사한 문장을 검색할 수 있다. 따라서 성능 좋은 Text embedding 모델을 사용하는 건 매우 중요하다.

stella-en-1.5B-v5 는 평균 유사도에서 가장 높은 성능을 보였지만, 다른 지표들의 성능이 상대적으로 낮아서 최종 모델로 선택하지 않았다. 반면, all-mpnet-base-v2 는 평균 유사도는 stella-en-1.5B-v5 보다 조금 낮지만, Recall@20 이 월등히 높았고, 전체적인 성능이 균형을 이루고 있어 최종적으로 선택했다. 또한, all-MiniLM-L6-v2 는 all-mpnet-base-v2 보다 모든 지표에서 성능이 낮았기 때문에 선택에서 제외됐다. 최종으로는 Recall@20 과 평균 순위 성능이 가장 뛰어난 all-mpnet-base-v2 를 baseline 모델로 선택했다.

Table 5. Text embedding 모델 기본 성능 비교

| 모델 | Recall@20 ↑ | 평균 순위↓ | 평균 유사도↑ |
|---|---|---|---|
| stella-en-1.5B-v5 [7] | 24.06% | 4.72 | **0.6397** |
| all-MiniLM-L6-v2 [8] | 48.66% | 5.64 | 0.5983 |
| all-mpnet-base-v2 [9] | **50.80%** | **4.71** | 0.6284 |

### 4.3.4. Text embedding model fine-tuning

선정된 모델을 이용하여 특정 장면에 대한 캡션을 추출하는 과정에서, 더 유사한 장면이 존재함에도 불구하고 적절하지 않은 장면이 탐색 되는 문제를 발견했다. 이를 해결하기 위해 쿼리를 생성하여 학습 데이터를 구축했다. 먼저, YouTube-8M 영상 데이터에서 5 초 단위로 분할된 클립 중에서 총 429 개의 클립을 샘플링했다. 이후, 분할된 클립에 대해 모델을 활용하여 캡션을 생성하고, 쿼리 생성 규칙을 설정하여 3 인의 작업자가 직접 쿼리 데이터를 작성했다. 쿼리 생성 규칙은 "[날씨] [시간대], [장소], [배경], [배경의 상태/특성], [특징]의 [사람/동물/사물]이(가) [상호작용/동작] 장면"의 형태를 따르도록 설계했다.

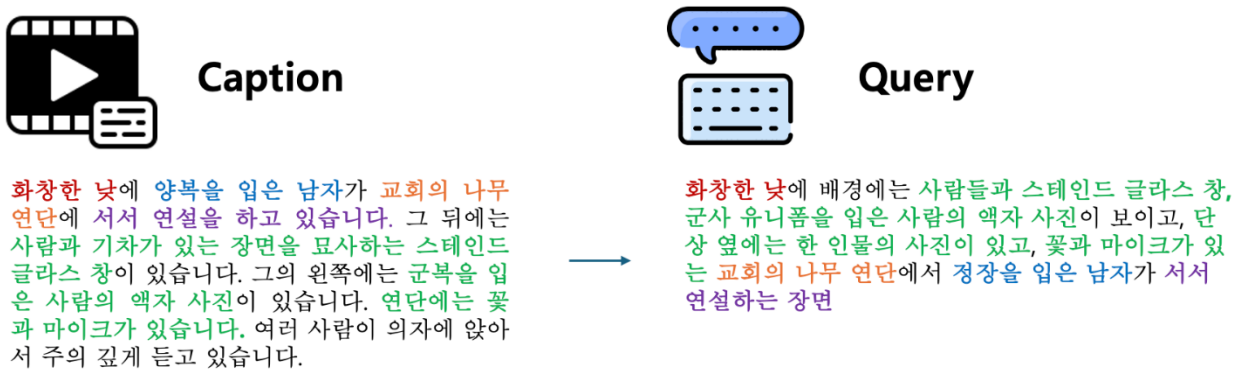


Fig 6. 캡션 기반 쿼리 생성 예시

Fig 6 을 보면 쿼리 생성 규칙에 따라 캡션을 보고 쿼리를 생성한 훈련 데이터 예시를 볼 수 있다. 쿼리 생성 규칙에서 정한 카테고리를 다음과 같은 색으로 표현했다.

- [날씨] [시간대] - 빨간색
- [장소] - 주황색
- [배경] [배경의 상태/특성] - 초록색
- [특징]의 [사람/동물/사물] - 파란색
- [상호작용/동작] - 보라색

또한 caption 에서 "양복"이라는 키워드를 query 에선 "정장"으로 바꾸는 등 동일한 키워드가 아니라도 유사한 의미가 있는 단어를 사용하여 훈련 데이터를 생성했다. 이는 자연어를 단순히 키워드 비교가 아닌 의미론적으로 유사한 문장은 같은 벡터 공간으로 매핑하는 Text embedding 모델을 만드는 데 사용되었다.

Table 6. Trained Text embedding 모델 성능 비교

| 모델(all-mpnet-base-v2) | Recall@3 ↑ | Recall@1 ↑ | 평균 유사도 ↑ |
|---|---|---|---|
| Base | 27.81% | 19.25% | 0.6612 |
| Trained | **34.22%** | **20.86%** | **0.7467** |

상술한 규칙을 기반으로 구축한 429 개의 쿼리-캡션 쌍을 Positive Sample 로 활용하여 임베딩 모델을 fine-tuning 했다. 학습 과정에서 의미론적으로 유사한 쿼리-캡션 간의 거리를 가깝게 하도록 학습시켰다. 모델의 평가에는 기존의 187 개 캡션-쿼리 쌍을 사용했으며 이를 통해 학습된 모델이 실제 데이터에서 유사도를 얼마나 효과적으로 측정하는지 검증했다. 그 결과 Recall@3 약 7% 증가, Recall@1 약 1% 증가, 평균 유사도 약 15%가 증가하였다. Fine-tuning 을 통해 쿼리와 캡션 간의 유사도를 보 정확하게 반영할 수 있도록 했으며 그 결과 원하는 장면과 더욱 정확하게 매칭되는 쿼리를 탐색할 수 있었다.

### 4.3.4. 비디오 분할 방식 실험

Text to Video 를 위한 데이터베이스를 구축하기 위해, 비디오 분할 방법에 따른 검색 성능 변화를 실험적으로 분석했다. 비디오 분할은 검색의 기본 단위를 결정하는 요소로, 장면을 효과적으로 분할하지 못하면 검색 결과의 정확도가 저하될 수 있다. 따라서 위에서 서술한 검색 성능 평가 지표인 Recall@N 을 활용해 실험적으로 평가하고 최적의 분할 방식을 도출했다.

우선 PysceneDetect 라이브러리를 활용하여 Content Detector 와 Adaptive Detector 기반의 장면 전환 감지 방법을 실험했다. Content Detector 는 HSV 색 공간에서 픽셀 변화량을 기반으로 장면 전환을 감지하며, 명확한 컷 편집이 존재하는 영상에서 유용하다. Adaptive Detector 는 이동 평균을 활용하여 빠른 움직임이 포함된 영상에서도 더욱 안정적으로 장면을 분할하는 방식이다. 그러나 실험 결과, Recall@1 및 Recall@5 성능이 상대적으로 낮았으며, 검색 성능이 전반적으로 일정하지 않은 경향을 보였다.

다음으로, OpenCV [10] 기반의 프레임 차이 감지 기법을 적용했다. 영상 프레임을 Grayscale 로 변환한 후, 픽셀 차이를 계산하여 특정 임곗값 이상 변화가 감지되면 새로운 장면으로 인식하는 방식이다. 이 방법은 색상 정보가 아닌 구조적 변화(객체 이동, 카메라 이동 등)에 반응하며, 영상의 명암 대비가 높을 효과적으로 동작할 것으로 예상되었다. 실험 결과, Recall@1 및 Recall@5 에서 일부 개선된 성능을 보였으나, 전체적으로 PysceneDetect 와 유사한 수준에 머물렀다.

마지막으로, FFmpeg [11] 를 사용하여 일정한 초 단위로 비디오를 분할하는 방법을 실험했다. 이 방법은 장면 전환을 고려하지 않고 일정한 간격(3 초, 5 초, 7 초)으로 영상을 분할하는 방식으로, 장면 탐지 알고리즘을 적용하는 방식보다 단순하지만, 일정한 시간 간격으로 데이터를 확보할 수 있다는 장점이 있다. 실험 결과, FFmpeg 기반의 초 단위 분할이 모든 기준(Recall@1~Recall@20)에서 가장 우수한 성능을 기록하였으며, 특히 3 초 간격으로 분할한 경우 Recall@1 이 29.95 로 가장 높게 나타났다.

이러한 결과는, 장면 기반 탐지 방식(PysceneDetect, OpenCV)보다 일정 간격으로 분할하는 방식이 검색 성능을 더 안정적으로 유지할 수 있음을 시사한다. 이는 장면 변화 감지 방식이 특정 영상 특성(컷 편집, 조명 변화 등)에 따라 성능이 달라질 수 있는 있지만, 고정된 시간 간격으로 분할하는 방식은 다양한 검색 조건에서도 일관된 성능을 유지할 수 있기 때문으로 해석된다.

Table 7. 비디오 분할 방식 성능 비교

| Split Method | | Recall@1 ↑ | Recall@5 ↑ | Recall@10 ↑ | Recall@15 ↑ | Recall@20 ↑ |
|---|---|---|---|---|---|---|
| PysceneDetect | Content | 22.99 | 48.66 | 56.68 | 62.57 | 64.17 |
| | Adaptive | 25.13 | 47.59 | 58.29 | 63.64 | 66.31 |
| OpenCV | | 25.67 | 48.13 | 57.22 | 63.10 | 65.24 |
| FFmpeg | 3s | **29.95** | **54.55** | 64.71 | 70.05 | 74.33 |
| | 5s | 26.74 | 52.94 | **65.24** | **73.80** | **75.40** |
| | 7s | 28.34 | 52.41 | 64.17 | 67.38 | 71.12 |

### 4.3.5. 최종 데이터베이스 성능 평가

일정한 간격으로 분할된 데이터베이스를 구축하는 방식이 검색 성능에 미치는 영향을 분석하기 위해, 데이터베이스를 확장할 검색 성능이 향상되는지 실험적으로 평가했다. 초기 실험에서는 전체 권장 데이터셋을 활용할 때 실험 시간이 과도하게 소요되는 문제가 발생하였기 때문에, 총 108 개의 비디오 샘플을 선정하여 실험을 진행했다.

우선 3 초, 5 초, 7 초 간격으로 각각 분할하여 구축한 데이터베이스와, 이들을 결합하여 구축한 데이터베이스의 검색 성능을 비교했다. 실험 결과, 단일 분할 방식으로 구축한 데이터베이스보다 3 초, 5 초, 7 초로 분할된 데이터를 함께 포함한 데이터베이스가 가장 우수한 검색 성능을 기록했다. 특히 3 초·5 초·7 초를 결합한 데이터베이스의 경우 Recall@1 성능이 35.29, 평균 유사도가 0.6577 로 가장 높은 성능을 보였다.

이러한 결과를 바탕으로 전체 1,218 개 비디오 데이터에 대해서도 동일한 방식으로 데이터베이스를 구축하고 실험을 확장했다. 실험 과정에서, 사용한 임베딩 모델에 따라 검색 성능이 달라지는 이전 실험 결과를 통해 최종적으로 학습시킨 임베딩 모델을 적용하고 3 초·5 초·7 초로 분할하여 통합 구축한 데이터베이스가 가장 높은 검색 성능을 기록했다.

전체 비디오를 활용한 최종 실험에서, 3 초·5 초·7 초를 결합한 데이터베이스의 Recall@1 성능은 21.39, 평균 유사도는 0.7696 으로 가장 우수한 결과를 보였다. 이는 단일 분할 방식보다 다양한 시간 간격을 결합하는 방식이 다양한 검색 조건에서도 일관된 성능을 유지하는 데 기여했음을 시사한다.

결과적으로, 본 프로젝트에서는 3 초·5 초·7 초로 분할된 데이터를 모두 포함한 데이터베이스 구축 방식을 최적의 방법으로 채택하였으며, 이를 기반으로 최종적인 Text to Video 검색 시스템을 설계했다.

Table 8. 최종 데이터베이스 검색 성능 비교

| DB 비디오 종류 | Embedding Model | Split Time | Recall@1 ↑ | 평균 유사도 |
|---|---|---|---|---|
| 108 | all-MiniLM-L6-v2 | 3s + 5s | 33.69 | 0.6509 |
| | | 3s + 7s | 32.09 | 0.6548 |
| | | 5s + 7s | 34.22 | 0.6320 |
| | | 3s + 5s + 7s | 35.29 | 0.6577 |
| 1218 (all) | all-MiniLM-L6-v2 | 3s + 5s | 14.44 | 0.6578 |
| | all-mpnet-base-v2 | 3s + 5s | 19.25 | 0.6612 |
| | **Trained model** | 3s + 5s | 20.86 | 0.7683 |
| | | **3s + 5s + 7s** | **21.39** | **0.7696** |

이 프로젝트에서는 Video to Text 모델을 통해 비디오를 텍스트 DB 로 저장했다. 또한, 비디오 검색 시스템의 성능을 향상하기 위해 임베딩 모델을 Fine-tuning 하고, 최적의 비디오 분할 방식을 탐색했다. 최종적으로, 이 프로젝트에서는 최적의 임베딩 모델과 비디오 분할 방식을 적용하여 Text to Video 검색 시스템을 구축했다. 이를 통해 비디오 데이터의 방대한 양을 효율적으로 탐색할 수 있는 기반을 마련하였으며, 향후 실제 검색 서비스에 적용할 가능성을 확인했다.

# 5. 자체 평가 의견

## 5.1 잘했던 점

**파이프라인 완성 및 데이터 이해**

초반에 설계한 파이프라인을 모두 완성하였기 때문에 달성도 측면에서는 훌륭하게 완수했다고 생각된다. 직접 데이터를 생성하는 작업을 통해 데이터의 이해도를 높이고, 라벨의 정확도를 높였다.

**정확도 성능 개선**

파이프라인을 완성한 후에 개선사항을 계속해서 생각하여 완성도를 높였다. 단순한 5 초 캡셔닝 DB 에 3 초, 7 초 DB 를 추가 및 임베딩 모델을 fine-tuning 하여 검색 성능인 Recall@1 을 9.09% 높였다.

**추론 속도 개선**

4 개의 서버에서 분산하여 처리하고, 캡셔닝 모델에 들어가는 이미지 수를 효과적으로 줄이는 등의 처리를 하였다. 또한 배치 수를 늘려 효율적인 처리가 가능하게 하였다. 이를 통해 최종적으로 추론 속도를 82.27**%** 상승시켰다.

**Web demo 구현**

본 연구에서는 파이프라인의 시연을 직관적으로 확인할 수 있도록 Web demo 를 구현하였다. 이를 위해 Flask 를 사용하였으며, Text-to-Video (T2V) 및 Video-to-Text (V2T) 기능을 제공했다. 본 데모는 연구의 주요 기능을 직관적으로 확인할 수 있도록 설계되었으며, 사용자 입력에 따른 영상 검색 및 캡셔닝 과정을 효율적으로 시연할 수 있다.

## 5.2 아쉬운 점 및 개선 사항

**오디오 캡셔닝, 자막 추가**

현재 평가 기준은 “시각 정보”만에 대한 캡셔닝 및 검색 성능이다. 따라서 오디오 정보가 주어진다면, 오히려 노이즈가 될 것으로 판단하여 오디오 정보를 제거했다. 하지만, 실제 서비스 상황에서는 오디오 정보(캡셔닝, 자막)를 추가한다면 더 나은 사용자 경험을 제공할 수 있을 것이다.

**등장인물 검색**

실제 서비스에서 사용자가 검색하는 경우, 영상 내 등장인물의 이름을 통해 검색하는 경우가 존재한다. 이 경우 연예인 이미지를 학습시킨 Multi Label Classification 모델 사용하면 비디오의 등장인물을 파악하는 것이 가능하다.

**Fine-tuning**

모델을 학습시키기 위해 대용량 데이터를 labeling 하는 작업은 큰 비용을 요구한다. 정확한 모델을 통해 대용량 데이터를 labeling 하거나 self-supervised learning 기법을 사용하여 모델을 fine-tuning 한다면 큰 labeling 비용 없이 모델의 성능을 개선할 수 있다.

# Reference

[1] Abu-El-Haija, Sami, et al. "Youtube-8m: A large-scale video classification benchmark." *arXiv preprint arXiv:1609.08675* (2016).

[2] Ye, Jiabo, et al. "mplug-owl3: Towards long image-sequence understanding in multi-modal large language models." *arXiv preprint arXiv:2408.04840* (2024).

[3] Zhang, Pan, et al. "Internlm-xcomposer: A vision-language large model for advanced text-image comprehension and composition." *arXiv preprint arXiv:2309.15112* (2023).

[4] Wang, Jiawei, et al. "Tarsier: Recipes for training and evaluating large video description models." *arXiv preprint arXiv:2407.00634* (2024).

[5] BAIN, Max, et al. Frozen in time: A joint video and image encoder for end-to-end retrieval. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021. p. 1728-1738.

[6] Jégou, H., et al. "FAISS: A library for efficient similarity search and clustering of dense vectors." (2017).

[7] Zhang, Dun, et al. "Jasper and Stella: distillation of SOTA embedding models." *arXiv preprint arXiv:2412.19048* (2024).

[8] Wang, Wenhui, et al. "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers." *Advances in Neural Information Processing Systems* 33 (2020): 5776-5788.

[9] Song, Kaitao, et al. "Mpnet: Masked and permuted pre-training for language understanding." *Advances in neural information processing systems* 33 (2020): 16857-16867.

[10] Gushchin, Alexander, Anastasia Antsiferova, and Dmitriy Vatolin. "Shot boundary detection method based on a new extensive dataset and mixed features." *arXiv preprint arXiv:2109.01057* (2021).

[11] Wu, Xintian, et al. "Extend the FFmpeg framework to analyze media content." *arXiv preprint arXiv:2103.03539* (2021).