201 – Loot at the Quote

Team Information

Team Name : HSPACE

Team Member : Jinung Lee, Beomjun Park, DoHyeon Kim, Soyoung Cho

Email Address : hspacedigitalforensicslab@gmail.com

Teams must:

- Provide a detailed, step-by-step description of their problem-solving approach to ensure reproducibility by another examiner.
- List all tools used to arrive at their conclusions.

Tools used:

Name:	ffmpeg	Publisher:	ffmpeg
Version:	7.1.1		
URL:	https://ffmpeg.org/download.html		

Name:	ffprobe	Publisher:	ffmpeg
Version:	7.1.1		
URL:	https://ffmpeg.org/download.html		

Name:	python	Publisher:	python
Version:	3.10.9		
URL:	https://python.org		

Name:	Visual Studio Code	Publisher:	Microsoft
Version:	1.104.1		
URL:	https://code.visualstudio.com/		

Step-by-step methodology:

문제 풀이에 앞서 대회에서 주어진 해시값과 다운로드 받은 해시값이 동일한지 확인한다.

Hash Value (MD5)

- sps-pps.zip : 1bf9efb3ea418567d3c91cb546dfc43d
- nal_unit.zip : 085caacc0da9341736c662bb209dd693

그림 1 대회에서 주어진 해시값 목록

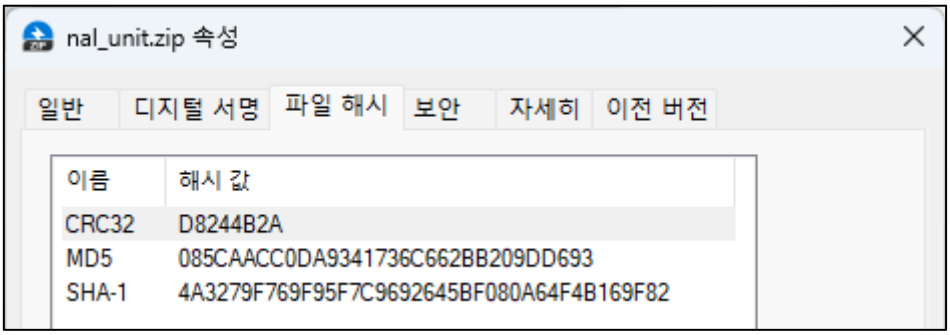

그림 2 hashtab 을 사용한 nal_unit.zip 의 해시 값

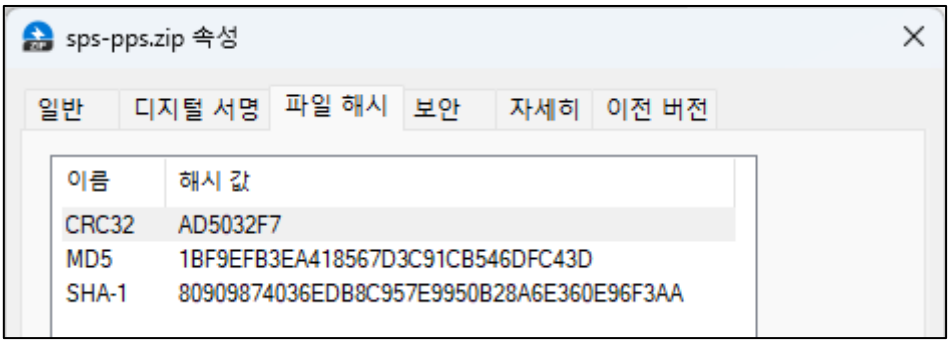

그림 3 hashtab 을 사용한 sps-pps.zip 의 해시 값

대회에서 주어진 해시값과 다운 받은 파일을 HashTab을 통해 해시값이 동일함을 확인하였다.

각각 주어진 압축파일을 해제하면 다음과 같은 파일을 확인 할 수 있다.

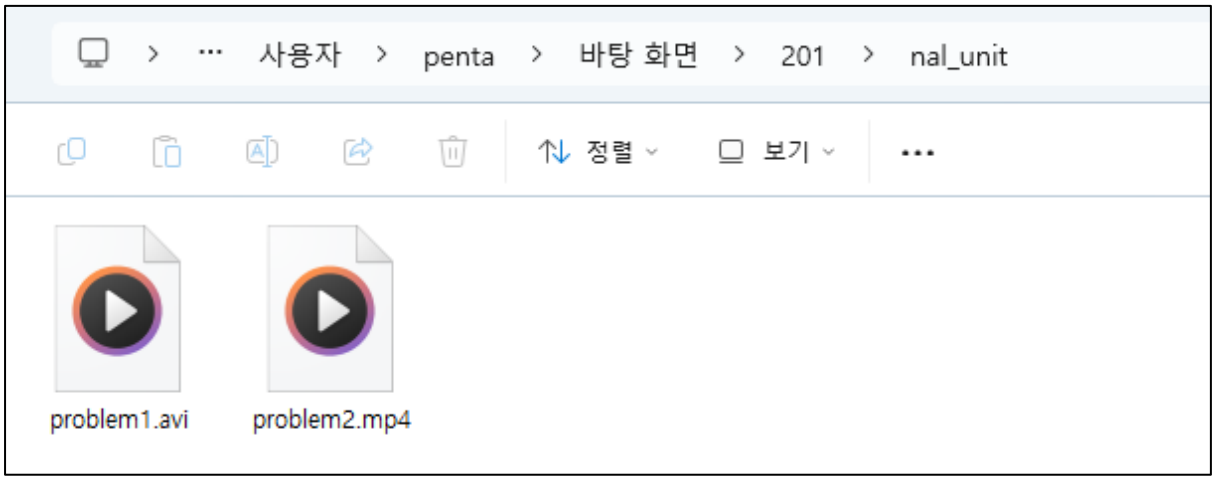

그림 4 nal-unit.zip 압축 해제 파일

nal-unit,zip을 압축 해제하면 2개의 파일을 확인 할 수 있으며, 각각 파일에 대한 정보는 다음과 같다.

FileName	problem1.avi
FileSize	9,717,752 바이트
MD5	00E3BCFC9FA9961F53999B6D836C5070

표 1 problem1.avi 파일 정보

FileName	problem2.mp4
FileSize	6,194,586 바이트
MD5	0596CB493827E7F830B6D3EA3EE70D6D
비디오 길이	00:02:34
데이터 속도	252kbps
총 비트 전송률	253kbps
프레임 속도	15.00 프레임/초

표 2 problem2.mp4 파일 정보

두 파일 모두 재생이 불가능한 상태를 나타내고 있다.

sps-pps.zip 파일을 압축 해제하면 30개의 파일을 확인 할 수 있다.

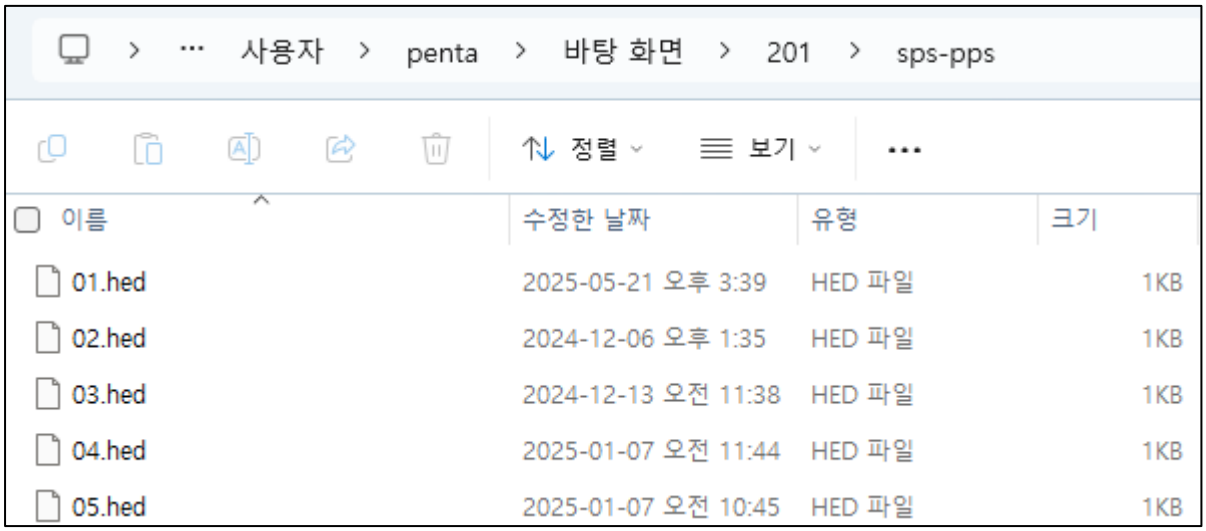

그림 5 sps-pps.zip 압축 해제 파일

문제 풀이에 앞서 H.264 코덱의 설명은 다음과 같다.

H.264/AVC(Advanced Video Coding)는 ITU-T Video Coding Experts Group과 ISO/IEC Moving Picture Experts Group에 의해 공동 개발된 비디오 압축 표준이다. H.264는 이전 표준 대비 약 50%의 비트레이트 절약을 제공하면서도 동일한 화질을 유지할 수 있는 고효율 압축 기술이다.

특징	설명
계층적 구조	Network Abstraction Layer(NAL)와 Video Coding Layer(VCL)로 분리
유연한 참조 프레임	다중 참조 프레임 지원으로 압축 효율성 향상
적응적 인트라 예측	9가지 4×4 예측 모드와 4가지 16×16 예측 모드
가변 블록 크기	16×16부터 4×4까지 다양한 블록 크기 지원
엔트로피 코딩	CAVLC(Context Adaptive Variable Length Coding)와 CABAC(Context Adaptive Binary Arithmetic Coding) 지원

표 3 H.264의 특징 정리

H.264/AVC는 현대 비디오 코덱의 표준 중 하나로, 디코딩을 위해서는 SPS/PPS와 같은 메타데이터가 반드시 필요하다. 전송/저장 과정에서 이들 Parameter Set이 누락될 경우 디코더는 화질 손상이나 전혀 디코딩하지 못하는 상태가 된다.

1. SPS는 영상의 해상도, 프레임 레이트, 인코딩 레벨 등 전체 스트림의 전역적 특성을 정의한다.

2. PPS는 슬라이스 단위의 디코딩 파라미터(엔트로피 코딩 모드, 참조 프레임 설정 등)를 지정한다.

이 두 메타데이터가 누락될 경우 디코더는 올바른 프레임 해석이 불가능해져, 화질 손상 또는 디코딩 불능 상태에 이르게 된다. 특히 네트워크 전송 과정이나 컨테이너 추출 과정에서 SPS/PPS가 손실되는 사례가 발생할 수 있으며, 이 경우 원본 스트림은 존재하더라도 재생이 불가능하다.

본 문제풀이를 위해서 컨테이너 레벨에서 H.264 NAL 스트림은 존재하나 SPS/PPS가 빠진 케이스를 대상으로, 외부에서 수집한 Parameter Set 후보(HED 파일)들을 조합하여 올바른 매칭을 찾아내는 방법이 필요하다.

Question 1. 첫 번째로 주어진 H.264(problem1.avi)의 SPS, PPS가 없는 NAL 유닛을 디코딩하여 영상에 기록된 철학자의 이름과 명언을 적어 제출 하시요.(50 points)

problem1.avi 분석과정은 다음과 같다.

먼저 ffprobe를 통해서 다음과 같은 특징을 확인 할 수 있다.

```
$ ./ffprobe.exe -v quiet -print_format json -show_format -show_streams nal_unit/problem1.avi
```

표 4 ffprobe를 이용한 problem1.avi 특징 분석 명령어


```
{
    "streams": [
        {
            "index": 0,
            "codec_name": "h264",
            "codec_long_name": "H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10",
            "codec_type": "video",
            "codec_tag_string": "x264",
            "codec_tag": "0x34363278",
            "width": 1280,
            "height": 720,
            "coded_width": 1280,
            "coded_height": 720,
            "has_b_frames": 0,
            "level": -99,
            "refs": 1,
            "is_avc": "false",
            "nal_length_size": "0",
```

```
            "r_frame_rate": "25/1",
            "avg_frame_rate": "25/1",
            "time_base": "1/25",
            "start_pts": 0,
            "start_time": "0.000000",
            "duration_ts": 0,
            "duration": "0.000000",
            "disposition": {
                "default": 0,
                "dub": 0,
                "original": 0,
                "comment": 0,
                "lyrics": 0,
                "karaoke": 0,
                "forced": 0,
                "hearing_impaired": 0,
                "visual_impaired": 0,
                "clean_effects": 0,
                "attached_pic": 0,
                "timed_thumbnails": 0,
                "non_diegetic": 0,
                "captions": 0,
                "descriptions": 0,
                "metadata": 0,
                "dependent": 0,
                "still_image": 0,
                "multilayer": 0
            }
        },
        {
            "index": 1,
            "codec_name": "adpcm_ima_wav",
            "codec_long_name": "ADPCM IMA WAV",
            "codec_type": "audio",
            "codec_tag_string": "[17][0][0][0]",
            "codec_tag": "0x0011",
            "sample_fmt": "s16p",
            "sample_rate": "8000",
            "channels": 1,
```

```
            "bits_per_sample": 4,
            "initial_padding": 0,
            "r_frame_rate": "0/0",
            "avg_frame_rate": "0/0",
            "time_base": "8/125",
            "start_pts": 0,
            "start_time": "0.000000",
            "bit_rate": "32000",
            "extradata_size": 2,
            "disposition": {
                "default": 0,
                "dub": 0,
                "original": 0,
                "comment": 0,
                "lyrics": 0,
                "karaoke": 0,
                "forced": 0,
                "hearing_impaired": 0,
                "visual_impaired": 0,
                "clean_effects": 0,
                "attached_pic": 0,
                "timed_thumbnails": 0,
                "non_diegetic": 0,
                "captions": 0,
                "descriptions": 0,
                "metadata": 0,
                "dependent": 0,
                "still_image": 0,
                "multilayer": 0
            }
        }
    ],
    "format": {
        "filename": "nal_unit/problem1.avi",
        "nb_streams": 2,
        "nb_programs": 0,
        "nb_stream_groups": 0,
        "format_name": "avi",
        "format_long_name": "AVI (Audio Video Interleaved)",
```

```
            "start_time": "0.000000",
            "size": "9717752",
            "probe_score": 100
        }
}
```


표 5 ffprobe를 통한 problem1.avi 결과

위 결과를 요약하면 다음과 같다.

컨테이너: AVI (format_name: avi), 파일 크기 ≈ 9.7 MB.

비디오: H.264 (codec_tag x264), 1280×720, 25 fps, has_b_frames=0, refs=1.

비디오 메타 특이: is_avc: "false", nal_length_size: "0", level: -99 (FFprobe가 level을 읽지 못함/미상).

오디오: adpcm_ima_wav (ADPCM IMA WAV), 8 kHz, mono, bit_rate 32000 (≈32 kbps), bits_per_sample 4.

문제표시: duration이 0.000000으로 나옴 즉, 컨테이너에 타이밍(index/PTS) 정보가 없거나 FFprobe가 읽지 못함.

AVI 컨테이너에서 순수 H.264 스트림을 추출하여 NAL Unit 구조를 분석했다.

```
$ ./ffmpeg -i nal_unit/problem1.avi -c copy -bsf:v h264_mp4toannexb extracted_h264
.h264
```

표 6 ffmpeg를 통해 H.264 스트림 추출 명령어

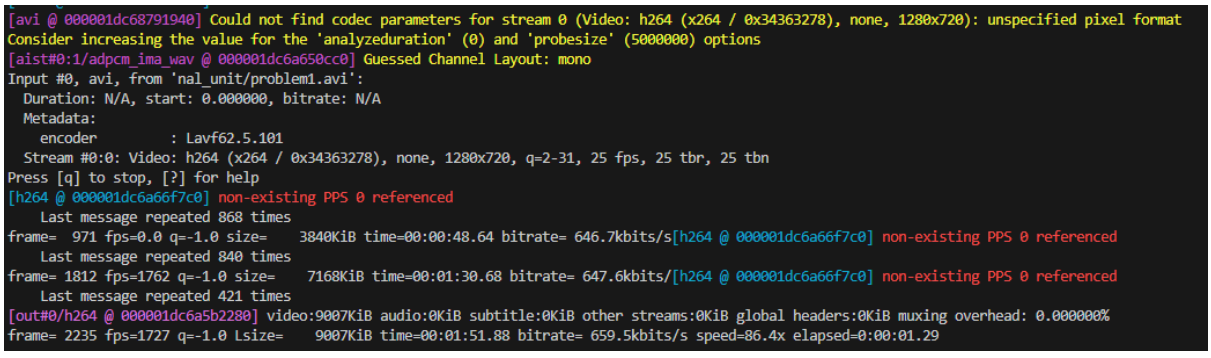

그림 6 ffmpeg를 통한 스트림 추출 과정 사진

ffmpeg의 h264_mp4toannexb 비트스트림 필터로 AVI 컨테이너에서 원시 H.264 스트림을 성공적으로 추출하였다. 추출 과정에서 'non-existing PPS 0 referenced' 경고가 다수 발생했는데, 이는 입

력 스트림에 필수 Parameter Set(SPS/PPS)이 포함되지 않았음을 의미하며, 디코딩 불능의 원인을 확인해 준다. 결과물 extracted_h264.h264는 Annex-B 형식의 원시 스트림으로, 이후 외부 HED 파일의 SPS/PPS를 prepend하여 디코딩 가능성을 브루트포스로 검증하는 단계로 진행하였다.

SPS/PPS 누락으로 디코딩이 불가한 H.264 스트림에 대해, 외부 후보(HED 파일)의 SPS·PPS를 모든 조합으로 대입하여 올바른 파라미터 세트를 탐색하는 절차를 코드로 구현하였다.

1. 후보 집합: sps-pps/ 디렉터리 내 HED 파일 30개(각각 SPS+PPS 포함)
2. 총 조합 수: 30 × 30 = 900 (SPS 30개 × PPS 30개, 크로스 매칭)
3. 접근 방식: 각 조합에 대해 Start Code(Annex-B) 방식으로 SPS, PPS를 원시 H.264 스트림 앞에 prepend 후 디코딩 시도

위 방식을 통해 코드를 작성하면 다음과 같다.

```
import os
import subprocess
from pathlib import Path

def setup_folders():
    folders = ['videos', 'thumbnails']
    for folder in folders:
        Path(folder).mkdir(exist_ok=True)

def test_h264(problem_file, sps_file, pps_file, problem_num, use_length_prefix=False):
    """H.264 조합 테스트"""
    try:
        # 파일 읽기
        with open(sps_file, 'rb') as f:
            sps_data = f.read()
        with open(pps_file, 'rb') as f:
            pps_data = f.read()
        with open(problem_file, 'rb') as f:
            video_data = f.read()

        # 파일명 생성
        sps_num = Path(sps_file).stem
        pps_num = Path(pps_file).stem
        video_file = f"videos/problem{problem_num}_sps{sps_num}_pps{pps_num}.h264"
```

```
        thumb_file = f"thumbnails/problem{problem_num}_sps{sps_num}_pps{pps_num}.png"

        # H.264 파일 생성
        with open(video_file, 'wb') as f:
            if use_length_prefix:
                # Length prefix 방식
                f.write(len(sps_data).to_bytes(4, 'big'))
                f.write(sps_data)
                f.write(len(pps_data).to_bytes(4, 'big'))
                f.write(pps_data)
            else:
                # Start code 방식
                f.write(b'₩x00₩x00₩x00₩x01')
                f.write(sps_data)
                f.write(b'₩x00₩x00₩x00₩x01')
                f.write(pps_data)
            f.write(video_data)

        # ffmpeg로 썸네일 생성
        cmd = ['./ffmpeg.exe', '-y', '-i', video_file, '-vframes', '1', '-loglevel', 'quiet', thumb_file]
        result = subprocess.run(cmd, capture_output=True)

        if result.returncode == 0 and os.path.exists(thumb_file):
            print(f"Problem {problem_num}: SPS={sps_num}, PPS={pps_num}")
            return True
        else:
            # 실패하면 파일 삭제
            if os.path.exists(video_file):
                os.remove(video_file)
            if os.path.exists(thumb_file):
                os.remove(thumb_file)
            return False

    except Exception:
        return False

def main():

    setup_folders()
```

```
    # 파일 목록
    hed_files = sorted([f for f in os.listdir("sps-pps") if f.endswith('.hed')])
    print(f"Found {len(hed_files)} HED files")

    success_count = 0
    total_tests = 0

    # Problem 1 (Start code)
    print("₩nProblem 1 (Start code)")
    for sps_file in hed_files:
        for pps_file in hed_files:
            total_tests += 1
            sps_path = f"sps-pps/{sps_file}"
            pps_path = f"sps-pps/{pps_file}"

            print(f"[{total_tests:3d}] SPS:{sps_file} + PPS:{pps_file}", end=" ")

            if test_h264("nal_unit/problem1.avi", sps_path, pps_path, 1, False):
                success_count += 1

if __name__ == "__main__":
    main()
```

표 7 problem1.avi 파일을 복구하기 위해 사용한 코드

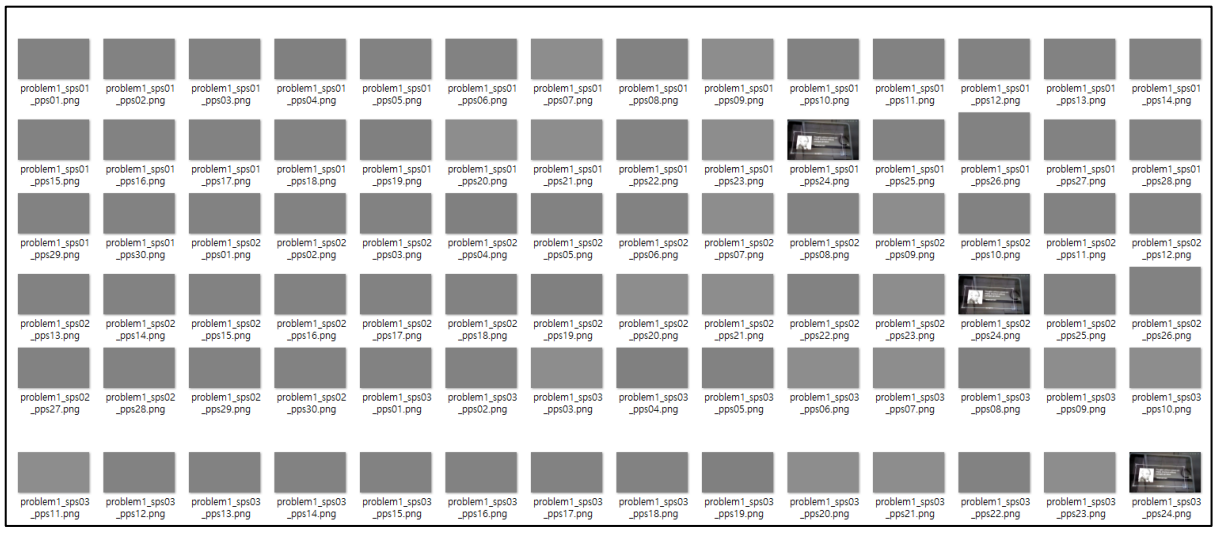

그림 7 표 7을 수행 한 결과 thumbnails에 저장된 사진

코드를 실행하면 thumbnails 폴더에 각각의 사진이 추출된다. 다음은 성공 판별 기준을 통하여 얻은 파일의 조합 목록이다.

파일명	SPS	PPS
problem1_sps01_pps24.png	01	24
problem1_sps02_pps24.png	02	24
problem1_sps03_pps24.png	03	24
problem1_sps04_pps24.png	04	24
problem1_sps05_pps24.png	05	24
problem1_sps06_pps24.png	06	24
problem1_sps07_pps24.png	07	24
problem1_sps08_pps24.png	08	24
problem1_sps09_pps24.png	09	24
problem1_sps10_pps24.png	10	24
problem1_sps11_pps24.png	11	24
problem1_sps12_pps24.png	12	24
problem1_sps13_pps24.png	13	24
problem1_sps14_pps24.png	14	24
problem1_sps15_pps24.png	15	24
problem1_sps16_pps24.png	16	24
problem1_sps17_pps24.png	17	24
problem1_sps18_pps24.png	18	24
problem1_sps19_pps24.png	19	24
problem1_sps20_pps24.png	20	24
problem1_sps21_pps24.png	21	24
problem1_sps22_pps24.png	22	24
problem1_sps23_pps24.png	23	24
problem1_sps24_pps03.png	24	03
problem1_sps24_pps06.png	24	06
problem1_sps24_pps10.png	24	10
problem1_sps24_pps11.png	24	11
problem1_sps24_pps24.png	24	24
problem1_sps24_pps27.png	24	27
problem1_sps24_pps30.png	24	30
problem1_sps25_pps24.png	25	24
problem1_sps26_pps24.png	26	24
problem1_sps27_pps24.png	27	24

problem1_sps28_pps24.png	28	24
problem1_sps29_pps24.png	29	24
problem1_sps30_pps24.png	30	24

표 8 복구에 성공한 SPS, PPS 조합 및 파일명

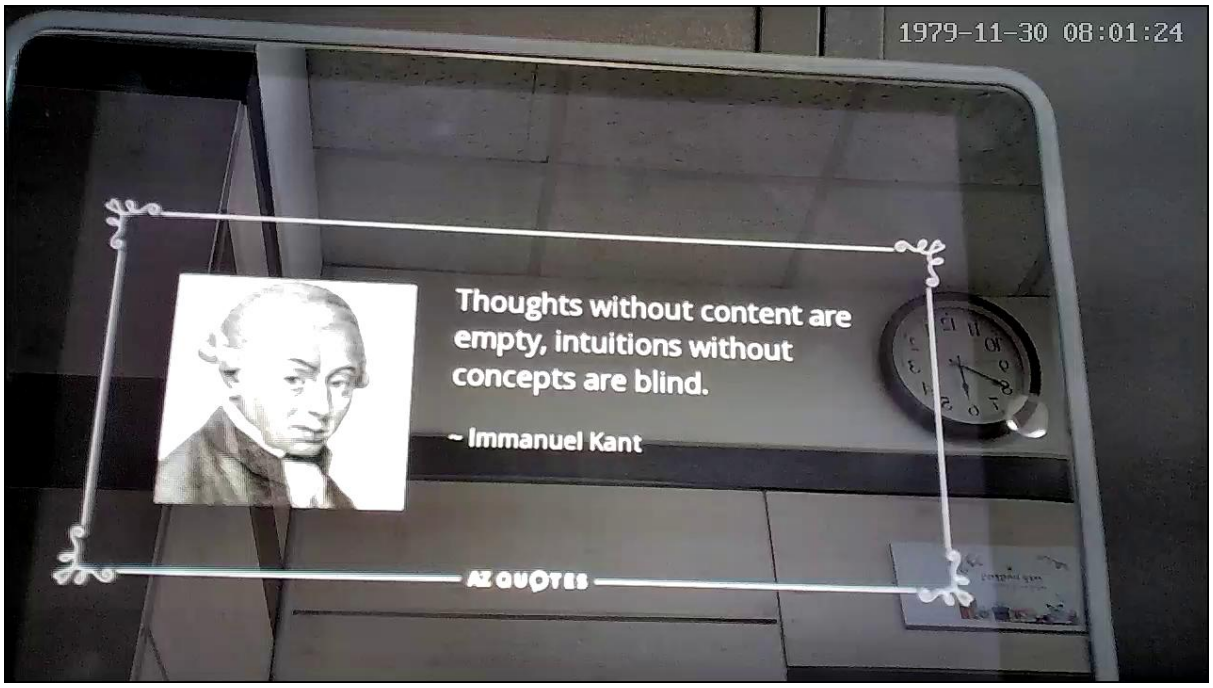

그림 8 복구된 problem1.avi에서 발견된 명언

철학자 이름	Immanuel Kant
명언	Thoughts without content are empty, intuitions without concepts are blind.

표 9 Question 1의 정답

Question 2. 두 번째로 주어진 H.264(problem2.mp4)의 SPS, PPS가 없는 NAL 유닛을 디코딩 하여 영상에 기록된 철학자의 이름과 명언을 적어 제출 하시요.(150 points)

problem2.mp4 분석과정은 다음과 같다.

먼저 ffprobe를 통해서 다음과 같은 특징을 확인 할 수 있다.

```
$ ./ffprobe.exe -v quiet -print_format json -show_format -show_streams nal_unit/problem2.mp4
```

표 10 ffprobe를 통해 problem2.mp4 파일 분석 명령어

```
{
    "streams": [
        {
            "index": 0,
            "codec_name": "pcm_mulaw",
            "codec_long_name": "PCM mu-law / G.711 mu-law",
            "codec_type": "audio",
            "codec_tag_string": "ulaw",
            "codec_tag": "0x77616c75",
            "sample_fmt": "s16",
            "sample_rate": "8000",
            "channels": 1,
            "bits_per_sample": 8,
            "initial_padding": 0,
            "id": "0x1",
            "r_frame_rate": "0/0",
            "avg_frame_rate": "0/0",
            "time_base": "1/8000",
            "start_pts": 3688,
            "start_time": "0.461000",
            "duration_ts": 1235560,
            "duration": "154.445000",
            "bit_rate": "63964",
            "nb_frames": "3859",
            "extradata_size": 4,
            "disposition": {
                "default": 1,
                "dub": 0,
                "original": 0,
                "comment": 0,
                "lyrics": 0,
                "karaoke": 0,
                "forced": 0,
                "hearing_impaired": 0,
                "visual_impaired": 0,
                "clean_effects": 0,
                "attached_pic": 0,
                "timed_thumbnails": 0,
                "non_diegetic": 0,
```

```
                "captions": 0,
                "descriptions": 0,
                "metadata": 0,
                "dependent": 0,
                "still_image": 0,
                "multilayer": 0
            },
            "tags": {
                "language": "eng",
                "handler_name": "ADPCM Audio Handle",
                "vendor_id": "[0][0][0][0]"
            }
        },
        {
            "index": 1,
            "codec_name": "h264",
            "codec_long_name": "H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10",
            "codec_type": "video",
            "codec_tag_string": "avc1",
            "codec_tag": "0x31637661",
            "width": 1920,
            "height": 1080,
            "coded_width": 1920,
            "coded_height": 1080,
            "closed_captions": 0,
            "film_grain": 0,
            "has_b_frames": 0,
            "level": -99,
            "refs": 1,
            "is_avc": "true",
            "nal_length_size": "4",
            "id": "0x2",
            "r_frame_rate": "15/1",
            "avg_frame_rate": "6948000/463157",
            "time_base": "1/90000",
            "start_pts": 0,
            "start_time": "0.000000",
            "duration_ts": 13894710,
            "duration": "154.385667",
```

```
            "bit_rate": "252210",
            "nb_frames": "2316",
            "extradata_size": 53,
            "disposition": {
                "default": 1,
                "dub": 0,
                "original": 0,
                "comment": 0,
                "lyrics": 0,
                "karaoke": 0,
                "forced": 0,
                "hearing_impaired": 0,
                "visual_impaired": 0,
                "clean_effects": 0,
                "attached_pic": 0,
                "timed_thumbnails": 0,
                "non_diegetic": 0,
                "captions": 0,
                "descriptions": 0,
                "metadata": 0,
                "dependent": 0,
                "still_image": 0,
                "multilayer": 0
            },
            "tags": {
                "language": "eng",
                "handler_name": "AVC Video Handler",
                "vendor_id": "[0][0][0][0]",
                "encoder": "AVC Coding"
            }
        }
    ],
    "format": {
        "filename": "/Users/pental/Downloads/201_attachments/nal_unit/problem2.mp4",
        "nb_streams": 2,
        "nb_programs": 0,
        "nb_stream_groups": 0,
        "format_name": "mov,mp4,m4a,3gp,3g2,mj2",
        "format_long_name": "QuickTime / MOV",
```

```
            "start_time": "0.000000",
            "duration": "154.906000",
            "size": "6194586",
            "bit_rate": "319914",
            "probe_score": 100,
            "tags": {
                "major_brand": "mobi",
                "minor_version": "0",
                "compatible_brands": "mobiavc1mp42isom"
            }
        }
}
```


표 11 ffprobe를 통한 problem2.mp4 분석 결과

ffprobe 분석 결과 AVCC(AVC configuration box)에 SPS/PPS 메타데이터는 존재하지만 샘플 단위(NAL)에는 SPS/PPS NAL이 포함되지 않은 전형적인 케이스였다. 또한 비디오의 전체 비트레이트가 1080p 대비 매우 낮고(약 252 kbps), 오디오 트랙은 G.711 μ-law(8kHz)로 시작 시간이 약 0.461초 지연되어 있어 AV 동기 관련 특이점이 관찰되었다. 이런 특성 때문에 raw H.264(Annex-B)로 단순 추출/재생하면 SPS/PPS 누락으로 디코더가 프레임을 제대로 해석하지 못하고 재생 실패 또는 깨진 화면이 발생할 수 있었다.

현재의 문제점을 요약하면 다음과 같다.

1. SPS/PPS가 스트림 내부에 없고 AVCC에만 존재: MP4 내부 AVCC에 SPS/PPS 정보만 들어 있고 각 프레임 앞에 SPS/PPS가 붙어 있지 않으면 일부 플레이어나 파싱 도구가 초기화 정보를 찾지 못하고 있다.

2. NAL 포맷 차이(AVCC vs Annex-B): MP4는 길이정보(4바이트 길이 프리픽스) 방식(AVCC)을 사용하므로 raw Annex-B(0x00000001 start code) 포맷으로 변환/전환이 필요하다.

3. 메타 불일치 표시: ffprobe에서 level=-99 등 레벨 정보가 비정상으로 표기되어 SPS 내부의 level_idc를 정상적으로 읽지 못한 흔적이 있다(비표준/손상 가능성 또는 파서 표시 문제).

4. 오디오 시작 지연(AV 오프셋): 오디오 start_time=0.461으로 영상과의 초기 동기 차이가 존재할 수 있다(재생/동기화 고려 필요).

5. 저비트레이트(화질 저하 우려): 1080p 해상도에 대해 비트레이트가 낮아 화질이 거칠 수 있다.

따라서 해결 방법을 다음과 같이 진행하였다.

핵심 아이디어는 올바른 SPS/PPS를 비디오 비트스트림 앞에 넣어 디코더가 초기화하도록 만드는 것이다. SPS/PPS 후보를 다수(브루트포스)로 조합해 넣어보고, 각 조합으로 생성한 raw H.264 파일을 ffmpeg로 디코딩해 썸네일(프레임 이미지)을 얻어 '의미 있는 프레임'을 판별함으로써 올바른 SPS/PPS 조합을 찾는 방법을 택했다. 이는 SPS/PPS가 손상되었거나 원본 AVCC 내 값이 정확치 않을 때 유용하다.

먼저 mp4 파일을 Annex-B로 변환하는 과정을 진행하였다. 다음은 변환에 사용한 명령어이다.

```
ffmpeg -i nal_unit/problem2.mp4 -c copy -bsf:v h264_mp4toannexb nal_unit/extracted_h264.h264
```

표 12 ffmpeg를 통한 mp4파일을 Annex-B로 변환하는 명령어

이 명령은 MP4 내부의 AVCC(길이 프리픽스) 기반 NAL들을 start-code(0x00000001) 방식(Annex-B)로 변환해 extracted_h264.h264에 저장한다. 다만 AVCC 내 SPS/PPS가 별도 박스에만 존재하는 경우(또는 샘플에 포함되어있지 않은 경우) 변환 후에도 유효한 SPS/PPS가 앞에 붙어있지 않을 수 있다.

추출된 Annex-B 파일(extracted_h264.h264)은 이후 브루트포스 조합의 대상이 된다. 이 단계에서는 여러 개의 HED 파일에서 분리해낸 SPS와 PPS 후보들을 서로 조합하여, 각각을 Annex-B 스트림의 맨 앞에 삽입한 새로운 H.264 파일을 생성한다.

각 조합된 H.264 파일은 FFmpeg을 이용해 첫 번째 프레임을 이미지로 덤프하고, 이어서 Pillow 라이브러리로 분석된다. 분석 과정에서는 고유 색상 수, 이미지 해상도, 파일 해시와 같은 메타데이터가 계산된다. 이렇게 얻어진 지표를 기준으로 실제 영상 프레임인지 여부를 판별한다. 예를 들어, 색상 수가 일정 기준을 넘으면 의미 있는 결과로 간주하며, 500 이상이면 매우 신뢰할 수 있는 성공 사례로 판단한다.

성공으로 판정된 조합은 bruteforce_results/successful 디렉터리에 .h264와 .png 형태로 보존되며, 각 결과는 조합명(spsXX_ppsYY), 생성 시각, 분석 메타정보와 함께 기록된다. 이후 사용자는 수동 재생을 통해 복구된 스트림의 품질과 연속성을 직접 확인할 수 있다.

사용한 코드는 다음과 같다.

```
import subprocess
from pathlib import Path
```

```
from PIL import Image
import hashlib
from datetime import datetime

class H264BruteForce:
    def __init__(self):
        self.sps_pps_dir = Path("sps-pps")
        self.h264_file = Path("nal_unit/extracted_h264.h264")

        self.base_output_dir = Path("bruteforce_results")
        self.videos_dir = self.base_output_dir / "videos"
        self.thumbnails_dir = self.base_output_dir / "thumbnails"
        self.successful_dir = self.base_output_dir / "successful"

        for directory in [self.videos_dir, self.thumbnails_dir, self.successful_dir]:
            directory.mkdir(parents=True, exist_ok=True)

        self.results = []
        self.successful_combinations = []

    def read_hed_file(self, hed_path):
        with open(hed_path, 'rb') as f:
            data = f.read()

        nal_units = []
        i = 0
        while i < len(data) - 4:
            if data[i:i+4] == b'₩x00₩x00₩x00₩x01':
                nal_start = i + 4
                nal_type = data[nal_start] & 0x1F

                next_start = len(data)
                for j in range(i + 4, len(data) - 3):
                    if data[j:j+4] == b'₩x00₩x00₩x00₩x01':
                        next_start = j
                        break

                nal_data = data[i:next_start]
                nal_units.append({
```

```
                    'type': nal_type,
                    'data': nal_data,
                    'size': len(nal_data)
                })

                i = next_start
            else:
                i += 1

        return nal_units

    def extract_sps_pps(self, nal_units):
        sps_data = None
        pps_data = None

        for nal in nal_units:
            if nal['type'] == 7:
                sps_data = nal['data']
            elif nal['type'] == 8:
                pps_data = nal['data']

        return sps_data, pps_data

    def create_combined_h264(self, sps_data, pps_data, sps_num, pps_num):
        with open(self.h264_file, 'rb') as f:
            h264_data = f.read()

        combined_data = bytearray()
        combined_data.extend(sps_data)
        combined_data.extend(pps_data)
        combined_data.extend(h264_data)

        output_path = self.videos_dir / f"sps{sps_num:02d}_pps{pps_num:02d}.h264"
        with open(output_path, 'wb') as f:
            f.write(combined_data)

        return output_path, len(combined_data)

    def generate_thumbnail(self, h264_path, sps_num, pps_num):
```

```
        thumbnail_path = self.thumbnails_dir / f"sps{sps_num:02d}_pps{pps_num:02d}.png"

        cmd = [
            'ffmpeg', '-y', '-v', 'quiet',
            '-i', str(h264_path),
            '-vframes', '1',
            '-f', 'image2',
            str(thumbnail_path)
        ]

        try:
            result = subprocess.run(cmd, capture_output=True, timeout=30)
            return thumbnail_path, result.returncode == 0 and thumbnail_path.exists(),
result.stderr.decode() if result.stderr else "Unknown error"
        except:
            return thumbnail_path, False, "Timeout or error"

    def analyze_thumbnail(self, thumbnail_path):
        img = Image.open(thumbnail_path)

        if img.mode == 'RGB':
            colors = img.getcolors(maxcolors=256*256*256)
        else:
            img_rgb = img.convert('RGB')
            colors = img_rgb.getcolors(maxcolors=256*256*256)

        unique_colors = len(colors) if colors else 0

        with open(thumbnail_path, 'rb') as f:
            file_hash = hashlib.md5(f.read()).hexdigest()

        return {
            'resolution': img.size,
            'unique_colors': unique_colors,
            'file_hash': file_hash,
            'file_size': thumbnail_path.stat().st_size,
            'is_meaningful': unique_colors > 100,
            'is_highly_meaningful': unique_colors > 500,
            'mode': img.mode
```

```
        }

    def copy_successful_files(self, sps_num, pps_num, analysis):
        if not analysis.get('is_meaningful', False):
            return

        combo_name = f"sps{sps_num:02d}_pps{pps_num:02d}"

        video_src = self.videos_dir / f"{combo_name}.h264"
        thumbnail_src = self.thumbnails_dir / f"{combo_name}.png"

        video_dst = self.successful_dir / f"{combo_name}_{analysis['unique_colors']}colors.h264"
        thumbnail_dst = self.successful_dir / f"{combo_name}_{analysis['unique_colors']}colors.png"

        import shutil
        if video_src.exists():
            shutil.copy2(video_src, video_dst)
        if thumbnail_src.exists():
            shutil.copy2(thumbnail_src, thumbnail_dst)

    def test_single_combination(self, sps_num, pps_num, sps_data, pps_data):
        print(f"테스트 중: SPS{sps_num:02d} + PPS{pps_num:02d}", end=" ... ")

        h264_path, h264_size = self.create_combined_h264(sps_data, pps_data, sps_num, pps_num)
        thumbnail_path, thumbnail_success, error = self.generate_thumbnail(h264_path, sps_num, pps_num)

        result = {
            'sps_num': sps_num,
            'pps_num': pps_num,
            'combination': f"sps{sps_num:02d}_pps{pps_num:02d}",
            'h264_path': str(h264_path),
            'h264_size': h264_size,
            'thumbnail_path': str(thumbnail_path),
            'thumbnail_success': thumbnail_success,
            'timestamp': datetime.now().isoformat()
        }
```

```
        if thumbnail_success:
            analysis = self.analyze_thumbnail(thumbnail_path)
            result.update(analysis)

            self.copy_successful_files(sps_num, pps_num, analysis)

            colors = analysis.get('unique_colors', 0)
            if analysis.get('is_highly_meaningful', False):
                print(f"□ {colors}개 색상")
                self.successful_combinations.append(result)
            elif analysis.get('is_meaningful', False):
                print(f"□ {colors}개 색상")
                self.successful_combinations.append(result)
            else:
                print(f"□ {colors}개 색상")
        else:
            print(f"□ 실패")
            result['error'] = error

        self.results.append(result)
        return result

    def run_bruteforce_analysis(self):
        print("H.264 브루트포스 분석 시작")
        print("=" * 60)

        hed_files = {}
        for hed_file in sorted(self.sps_pps_dir.glob("*.hed")):
            file_num = int(hed_file.stem)
            nal_units = self.read_hed_file(hed_file)
            hed_files[file_num] = nal_units

        print(f"HED 파일: {len(hed_files)}개")

        file_numbers = sorted(hed_files.keys())
        total_combinations = len(file_numbers) ** 2

        print(f"총 조합: {total_combinations}개")
```

```
            print("=" * 60)

            tested = 0
            for sps_num in file_numbers:
                for pps_num in file_numbers:
                    tested += 1
                    progress = (tested / total_combinations) * 100

                    print(f"[{tested:3d}/{total_combinations}] ({progress:5.1f}%) ", end="")

                    sps_data, _ = self.extract_sps_pps(hed_files[sps_num])
                    _, pps_data = self.extract_sps_pps(hed_files[pps_num])

                    if sps_data and pps_data:
                        self.test_single_combination(sps_num, pps_num, sps_data, pps_data)
                    else:
                        print(f"SPS{sps_num:02d} + PPS{pps_num:02d} ... □ 데이터 추출 실패")

            print("=" * 60)
            print("분석 완료!")

def main():
    bf = H264BruteForce()

    try:
        bf.run_bruteforce_analysis()
    except Exception as e:
        print(f"오류: {e}")

if __name__ == "__main__":
    main()
```

표 13 problem2.mp4 파일을 복구하기 위한 코드

```
pental@pentalui-MacBookAir 201_attachments % /opt/homebrew/bin/python3 /Users/pental/Downloads/201_attachments/test.py
H.264 브루트포스 분석 시작
============================================================
HED 파일: 30개
총 조합: 900개
============================================================
[  1/900] (  0.1%) 테스트 중: SPS01 + PPS01 ... ❌ 실패
[  2/900] (  0.2%) 테스트 중: SPS01 + PPS02 ... ❌ 실패
[  3/900] (  0.3%) 테스트 중: SPS01 + PPS03 ... ❌ 실패
[  4/900] (  0.4%) 테스트 중: SPS01 + PPS04 ... ⚪ 1개 색상
[  5/900] (  0.6%) 테스트 중: SPS01 + PPS05 ... ❌ 실패
[  6/900] (  0.7%) 테스트 중: SPS01 + PPS06 ... ❌ 실패
[  7/900] (  0.8%) 테스트 중: SPS01 + PPS07 ... ❌ 실패
[  8/900] (  0.9%) 테스트 중: SPS01 + PPS08 ... ❌ 실패
[  9/900] (  1.0%) 테스트 중: SPS01 + PPS09 ... ❌ 실패
[ 10/900] (  1.1%) 테스트 중: SPS01 + PPS10 ... ❌ 실패
[ 11/900] (  1.2%) 테스트 중: SPS01 + PPS11 ... ❌ 실패
[ 12/900] (  1.3%) 테스트 중: SPS01 + PPS12 ... ❌ 실패
[ 13/900] (  1.4%) 테스트 중: SPS01 + PPS13 ... ❌ 실패
```

그림 9 표 13 코드를 수행하는 과정 사진 1

```
[575/900] ( 63.9%) 테스트 중: SPS20 + PPS05 ... 🔥 82013개 색상
[576/900] ( 64.0%) 테스트 중: SPS20 + PPS06 ... 🔥 86556개 색상
[577/900] ( 64.1%) 테스트 중: SPS20 + PPS07 ... ⚪ 1개 색상
[578/900] ( 64.2%) 테스트 중: SPS20 + PPS08 ... ⚪ 1개 색상
[579/900] ( 64.3%) 테스트 중: SPS20 + PPS09 ... ⚪ 1개 색상
[580/900] ( 64.4%) 테스트 중: SPS20 + PPS10 ... ⚪ 1개 색상
[581/900] ( 64.6%) 테스트 중: SPS20 + PPS11 ... ⚪ 1개 색상
[582/900] ( 64.7%) 테스트 중: SPS20 + PPS12 ... 🔥 103271개 색상
[583/900] ( 64.8%) 테스트 중: SPS20 + PPS13 ... ⚪ 1개 색상
[584/900] ( 64.9%) 테스트 중: SPS20 + PPS14 ... ⚪ 1개 색상
[585/900] ( 65.0%) 테스트 중: SPS20 + PPS15 ... ⚪ 1개 색상
[586/900] ( 65.1%) 테스트 중: SPS20 + PPS16 ... ⚪ 1개 색상
[587/900] ( 65.2%) 테스트 중: SPS20 + PPS17 ... ⚪ 1개 색상
[588/900] ( 65.3%) 테스트 중: SPS20 + PPS18 ... ⚪ 1개 색상
[589/900] ( 65.4%) 테스트 중: SPS20 + PPS19 ... ❌ 실패
[590/900] ( 65.6%) 테스트 중: SPS20 + PPS20 ... ⚪ 1개 색상
[591/900] ( 65.7%) 테스트 중: SPS20 + PPS21 ... ⚪ 1개 색상
[592/900] ( 65.8%) 테스트 중: SPS20 + PPS22 ... ⚪ 1개 색상
[593/900] ( 65.9%) 테스트 중: SPS20 + PPS23 ... ⚪ 1개 색상
[594/900] ( 66.0%) 테스트 중: SPS20 + PPS24 ... ⚪ 1개 색상
[595/900] ( 66.1%) 테스트 중: SPS20 + PPS25 ... ⚪ 1개 색상
[596/900] ( 66.2%) 테스트 중: SPS20 + PPS26 ... ⚪ 1개 색상
[597/900] ( 66.3%) 테스트 중: SPS20 + PPS27 ... 🔥 86556개 색상
[598/900] ( 66.4%) 테스트 중: SPS20 + PPS28 ... ⚪ 1개 색상
[599/900] ( 66.6%) 테스트 중: SPS20 + PPS29 ... 🔥 103271개 색상
```

그림 10 표 13 코드를 수행하는 과정 사진 2

그림 11 bruteforce_results/thumbnails 폴더에 저장된 사진

코드를 실행하면 bruteforce_results/thumbnails 폴더에 각각의 사진이 추출된다. 다음은 성공 판별 기준을 통하여 얻은 파일의 조합 목록이다.

파일명	SPS	PPS
sps20_pps05.png	20	05
sps20_pps06.png	20	06
sps20_pps12.png	20	12
sps20_pps27.png	20	27
sps20_pps29.png	20	29

표 14 명언을 확인 할 수 있는 SPS, PPS 조합 및 파일명

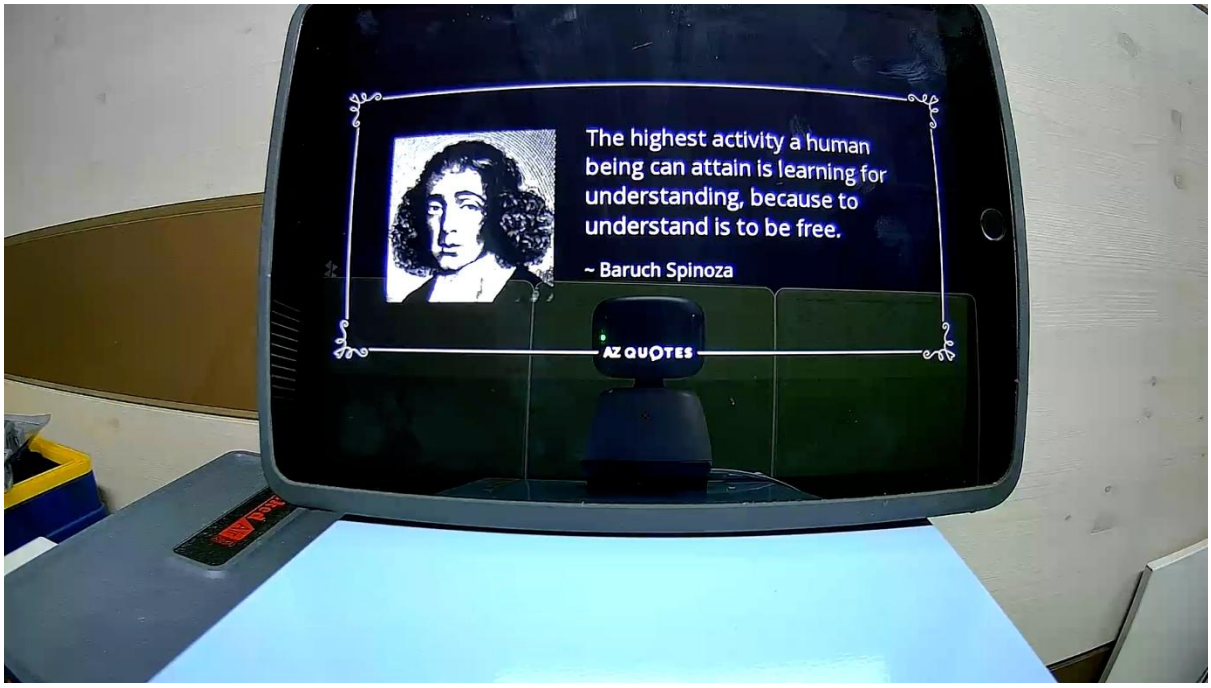

그림 12 복구된 problem2.mp4에서 발견된 명언

철학자 이름	Baruch Spinoza
명언	The highest activity a human being can attain is learning for understanding, because to understand is to be free.

표 15 Question2의 정답