1. **Download the infamous Big Buck Bunny video Create a new endpoint / feature which will let you to modify the resolution (use FFmpeg in the backend)**

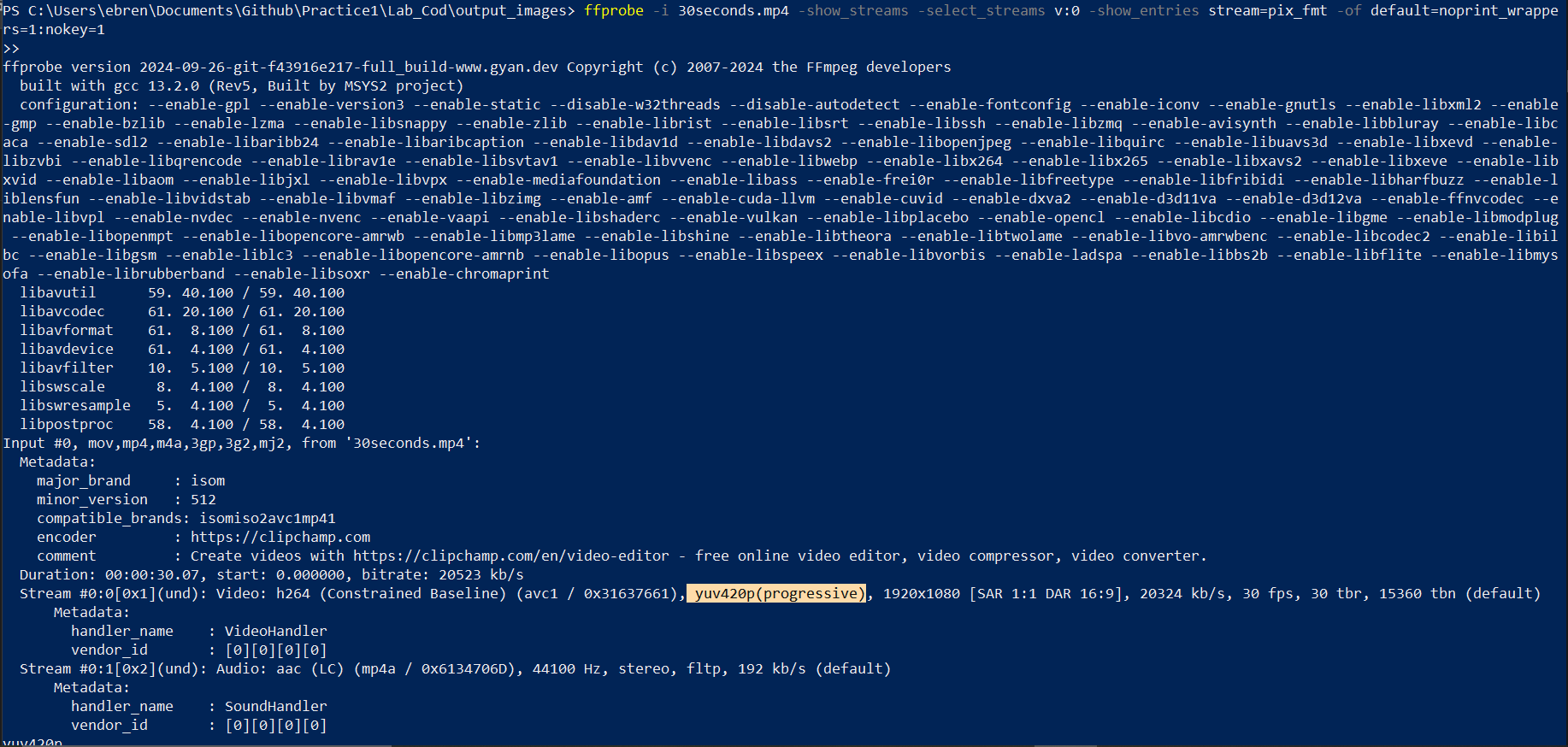
For this exercise we take advantage of our resize method in the *colorconversor* class which allows us to modify the resolution. We have tried with both 40x40 and 200x200 and the results are the following.

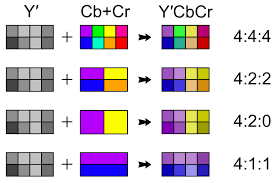
****

40x40 200x200

1. **Create a new endpoint / feature which will let you to modify the chroma subsampling**

After creating the method, we found that there weren’t any changes but a file was being created. We thought that our original file was already in the lowest chroma format possible and by using *ffprobe* (a method of ffmpeg) and we found that our point was exactly proven.

****



We have also tried to upsample the video to the highest possible format (yuv444p) and then subsample it to the lowest possible (yuv420p) to see if some result could be seen for our method. Unfortunately, this wasn’t possible either.

We saw that subsampling a video to a format yuv420p should make the appearance of our video as follows

1. **Create a new endpoint / feature which lets you read the video info and print at least 5 relevant data from the video**

In this case instead of using ffmpeg we have used *ffprobe* (as the previous exercise to check the details) and in this case we have delimited some specifications in our command. In this case we have chosen duration, the codec used, the width, height and frame rate. This is the response of our method which coincides with the input video we have given to it (its a 30 second version of BBB)

****

1. **You’re going to create another endpoint in order to create a new BBB container. It will fulfill this requirements:**

**· Cut BBB into 20 seconds only video.**

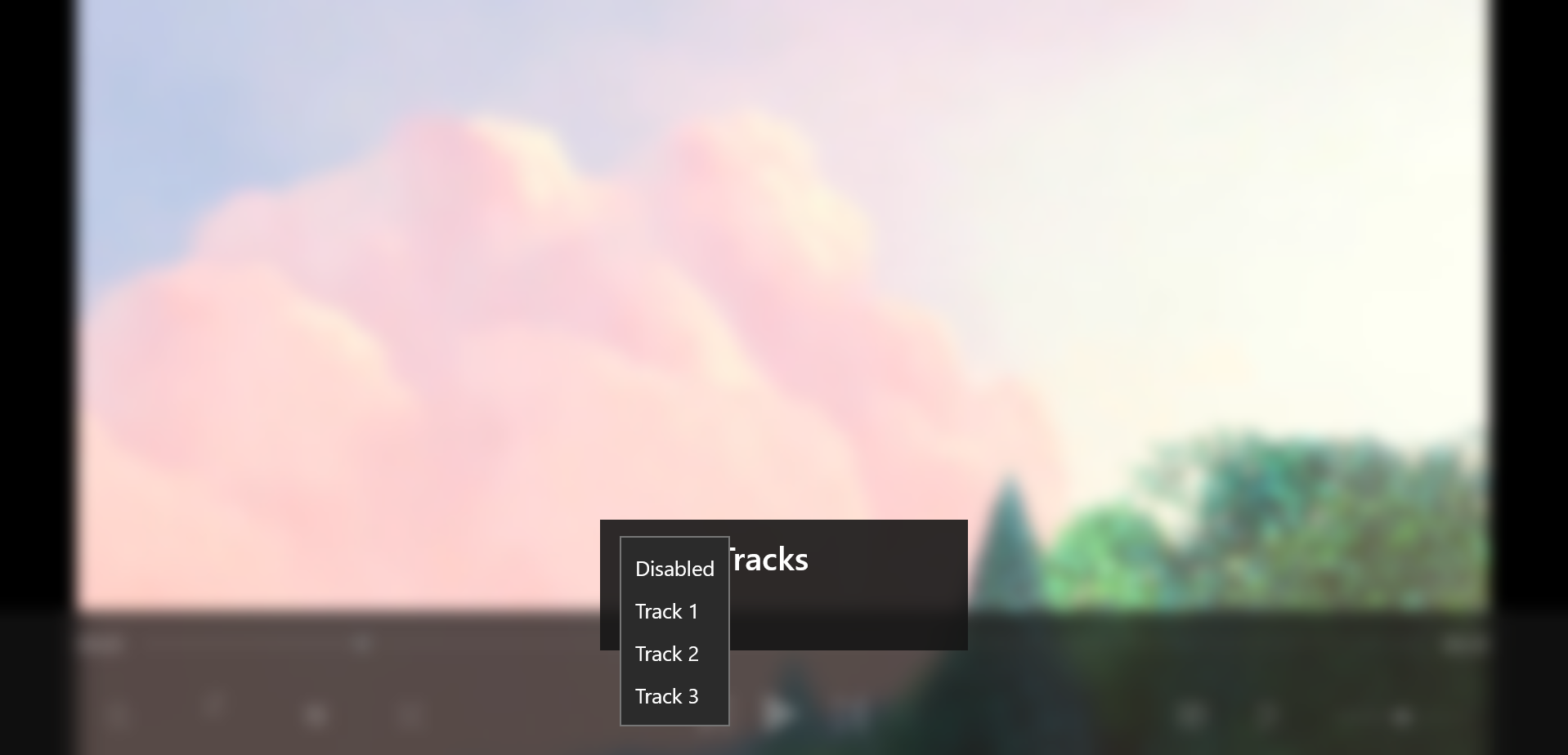
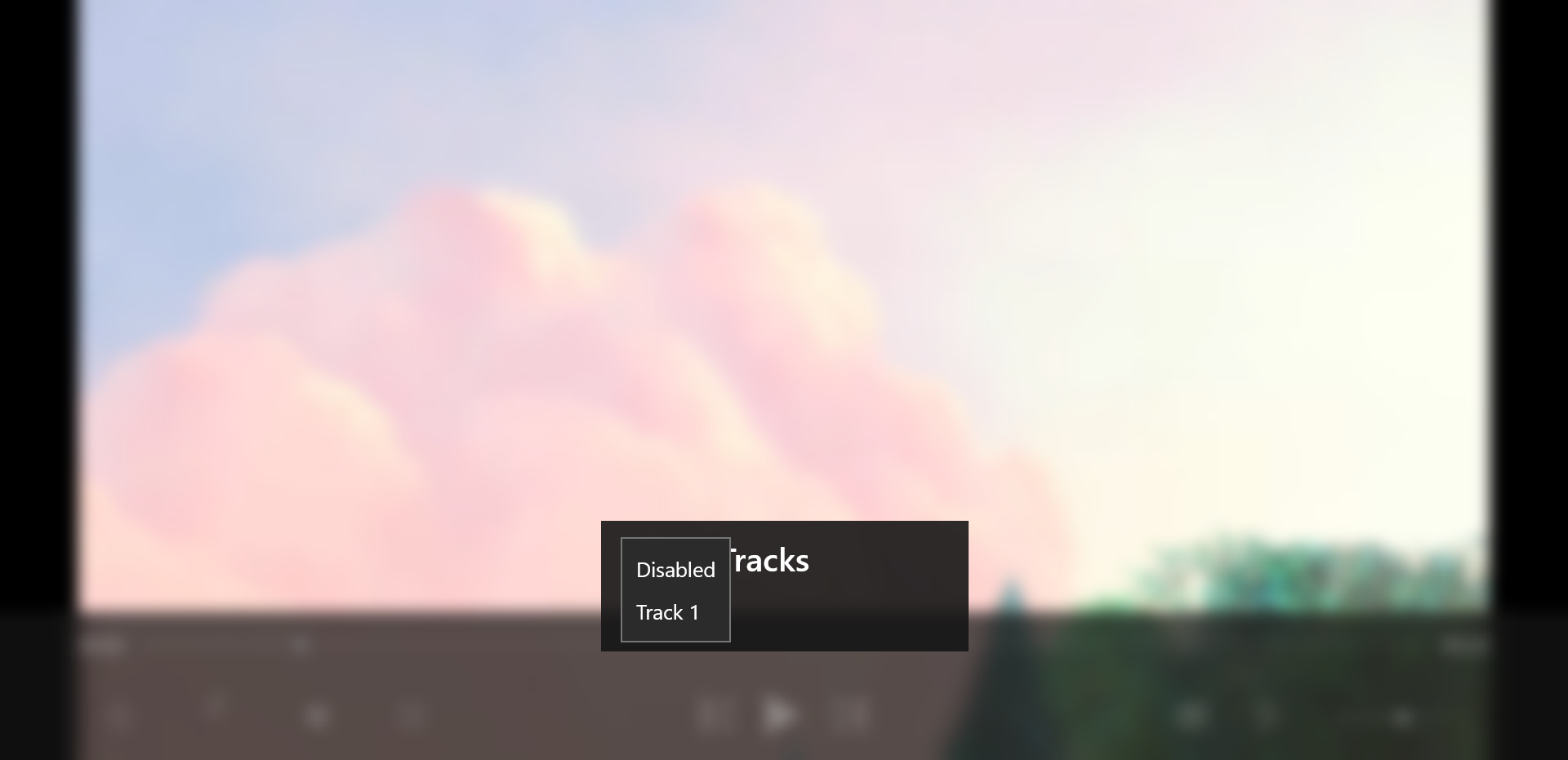
**· Export BBB(20s) audio as AAC mono track.**

**· Export BBB(20s) audio in MP3 stereo w/ lower bitrate**

**· Export BBB(20s) audio in AC3 codec Now package everything in a .mp4 with FFMPEG**

In this case we have created four new methods, one to cut the video down to 20 seconds (using -t 20 in our ffmpeg command) and then three different for each one of the audio codecs. We have to check ffmpeg -codecs to verify how to call each of these codecs. Also by checking the official website and the examples of ffmpeg we were guided to the final commands used.

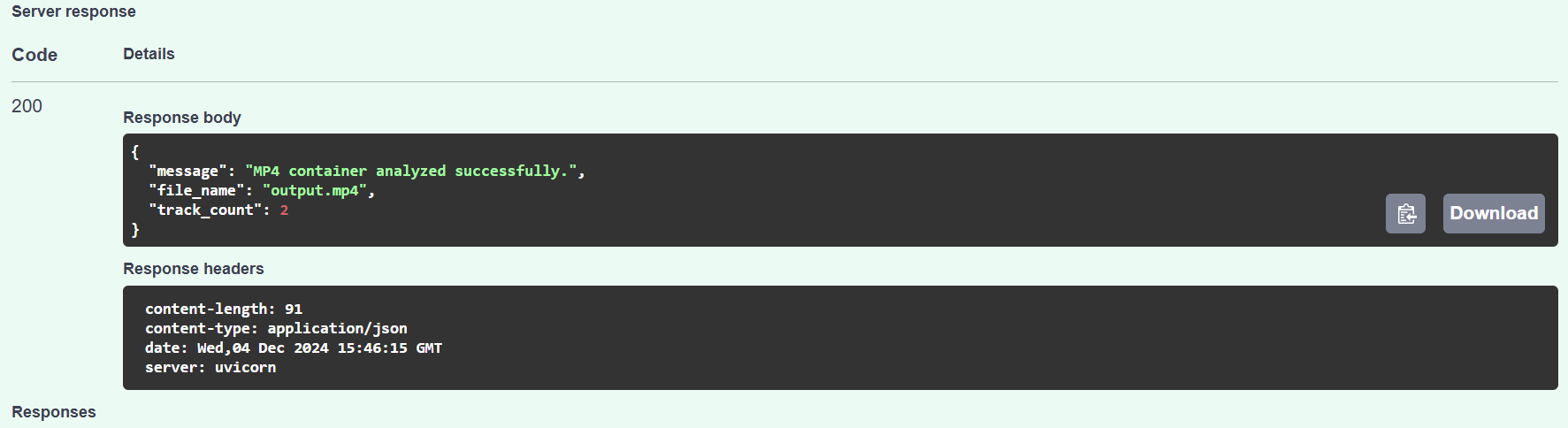
Finally we need to package everything in an mp4 with ffmpeg, but we saw that there are many methods to compact different audios in a video, some methods merge them, others overlap each other. We found a way to map each of the audios in different tracks of our video such that we have 1 video and 3 different track options to choose from:

****

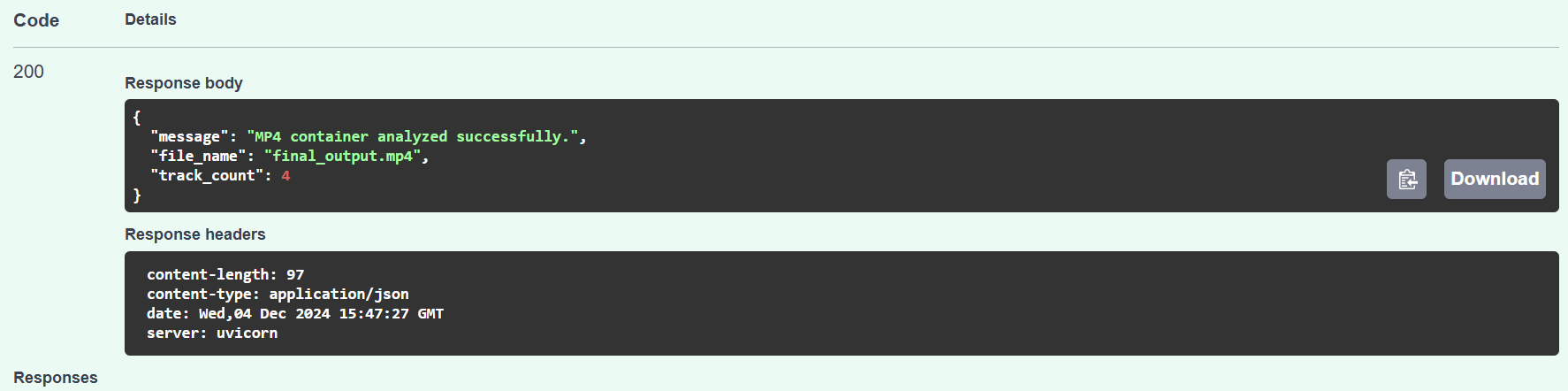
Original video has just one track The final one has 3 tracks

1. **Create a new endpoint / feature which reads the tracks from an MP4 container, and it’s able to say (deliver an output) of how many tracks does the container contains**

In this case we can see that the method used with the input file gives us correctly the track count correctly

****

For original video: 2 tracks (track one and disable one)

****

For the new one with the new tracks, we have 4

(Disable, Track one, track 2, track 3)

1. **Create a new endpoint / feature which will output a video that will show the macroblocks and the motion vectors**

For this endpoint to be possible, we need to make our command have enabled the motion vector export, so we can make the output video have the motion vectors and the macroblocks. Since we want to visualize these vectors and macroblocks, we use the command -vf codecview=mv = pf+bf+bb where pf represent the forward predicted motion of P pictures, bf is for B pictures and bb represents the backward predicted motion of B pictures

****

1. **Create a new endpoint / feature which will output a video that will show the YUV histogram**

For this case we have searched through ffmpeg web to create a histogram since we wanted to make it overlay on top of the video, then by following the example command.

At this point we saw that it is important to use a quality control for some of the codecs used (as VP8, VP9, H265) and we saw that smaller values of CRF can result in higher quality but larger file sizes while higher CRF values produce the other possibility, lower quality but smaller files. We decided to put a crf of 23 which is an intermediate value.

