Ferran Clapés, Pol Brugarolas SCAV

Report - P2

Introduction

The goal of this practice is to upgrade our previous work done in practice 1 and seminar 2 into a more complex video processing system. We integrated FFmpeg more deeply to handle other tasks like transcoding to different codecs and creating encoding ladders. Finally, we developed a Graphical User Interface (GUI) to interact with our Dockerized application that we've been working with since P1.

1. Video Transcoding (VP8, VP9, h265 & AV1)

"Create a new endpoint/feature to convert any input video into VP8, VP9, h265 & AV1"

In the seminar 2 we already did some similar work working with the resolution of a video and doing changes on audio còdecs of the container. In a similar way we can change the video còdec of an input video using FFmpeg.

For this exercici we found some information of the vídeo còdecs in the wiki of the FFmpeg:

https://trac.ffmpeg.org/wiki/Encode/VP9 or https://trac.ffmpeg.org/wiki/Encode/AV1 for example.

Simply using the attribute of the output function "vcodec= name_of_codec" and using the correct codec library we can then realise the transcoding. It also needs to specify the audio codec it must be used.

With that then we create the endpoint of the docker as in the other previous work.

Parenthesis Pause

At this point we were exhausted of having to erase the container and image of the docker desktop application every time we want to try a change that we've done in the codec. So, we wanted to find a solution on that. That's because we added the docker-compose.yml that helps us on having a docker image that keeps updating every time we save one of the files, using the command "docker-compose up" to start the system and Ctrl+C to exit from it.

2. Encoding Ladder

"Create a new endpoint/feature to be able to do an encoding ladder"

For the second exercise, we had to generate an "encoding ladder" , meaning multiple versions of the video at different resolutions.

To do that, we imported the function that we've done in the seminar 2, to not repeat the code as the statement says. With that scaling function, then we only have to use it múltiple times in diferent resolutions and then pack all the generated outputs in some way to output them as a single file, so we opted for a zip file output.

Have to say that to work efficiently and not have a code that lasts many minutes to execute, we only compute it with 30s of the video input.

3. Graphical User Interface

For the final step of the project the statement tells us to develop a Graphical User Interface (GUI) that would help us to work with our functions we made, after searching for different possibilities we decided to implement the solution using Streamlit. This system allows us to build an interactive web application only using Python. (https://docs.streamlit.io/).

The aim is to build a GUI that permits us to upload a video and then select the codec we want and call the last function we made, at least to start, through the docker. And then be able to download the result (what we not achieve).

To start the System we only need to use the command “streamlit run GUI.py” and it would open the browser with the created page. We have to have the docker run at that time, it would not work if not.

4. AI Graphical User Interface

As the statement says, know we can use AI to obtain a good and pretty result. The AI proposed us to upgrade our Streamlit. The result is a much cleaner and cool website than before. It allows users to upload videos, select codecs from a dropdown menu, and download the results with a simple button. To start the System we only need to use the command “streamlit run GUI_AI.py” and it would open the browser with the created page.