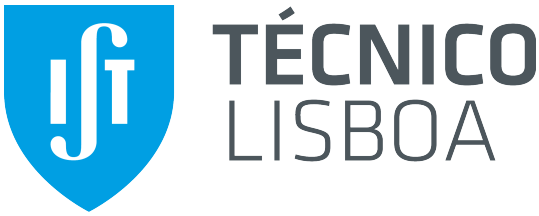

Immersivaudio: audio generation based on video features.

Michele Vitale
ist1111558

Daniele Avolio
ist1111559

Teodor Chakarov
ist1111601

I. Introduction

Generative Artificial Intelligence has brought a completely new approach to the process of creating digital contents: since the public release of services such as ChatGPT [1] or DALL-E [2], the field has been growing exponentially, with new models and applications that can produce text, images, audio and even videos with growing quality levels.
In this project we deepen some of the tasks and possibilities that these models can offer, with the aim of using state-of-art models that are freely available to the public to solve a multimedia task.

II. Problem description

The problem we are trying to solve is the enhancement of a video by adding an audio track that can better fit the content of the video itself. The audio track can be both music or music and sound effects, based on the user preferences.
The problem can fit in various scenarios and contexts, such the creation of audio for a video that has to be published on social medias and requires copyright-free music (it might be the case of online platforms such as YouTube, where the copyright policies are very strict [3]), visually impaired people that can get a more immersive experience from a video combining our solution with already existing audio description services, or simply to enhance audio tracks of videos that have noisy or low-quality audio or that have no audio at all, such as drones footage or submarine recordings.
Thus, the range of use cases of the system is wide and flexibility is a key requirement.

III. Proposed solution

The proposed solution is a pipeline that takes in input a video and returns the input video combined with the enhanced audio track. It is composed of a number of modules that decompose the problem into clear and manageable tasks, with the idea of making the system modular, scalable, easily adaptable to different scenarios and deployable on various workload distribution models, such as computer or container clusters.
The main idea is to use the features of the video to generate a prompt that can be used to condition the audio generation module, so that the generated audio can better fit the video. The features of the video extracted during the process are based on the content of a number of frames extracted by the input. Each step of the process has been discussed in the following sections, in particular in the **Architecture** section of this report.

IV. Architecture

The complete architecture of the system is shown in Figure 1. The figure shows the architecture of the whole pipeline applied to a video, but other pipelines are provided to adapt to different use cases and input types. The different components of the system are communicating using the JSON format and each module is appending its output rather than reshaping the output to be only the required one from the following step, enabling the process to be modular and adaptable to different scenarios.
Each video is internally identified with an unique token, which is computed at input as the MD5 hash of the video itself.

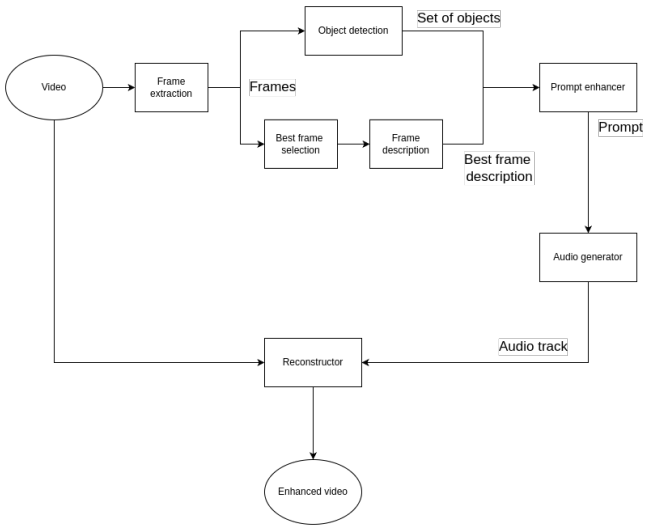

Fig. 1. Architecture of the system.

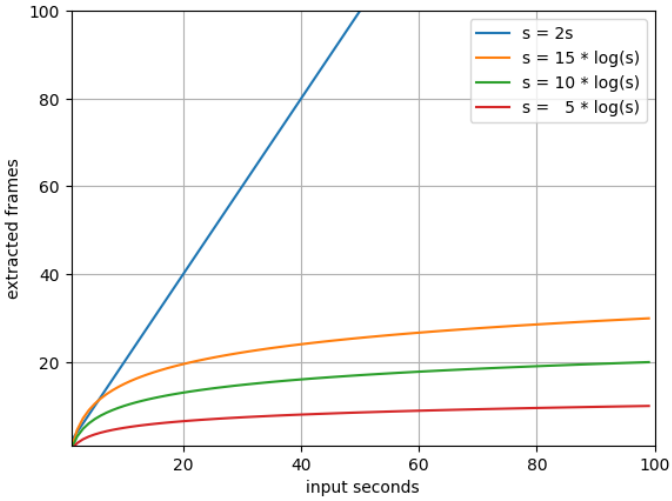

Fig. 2. Comparison of different frame extraction functions.

The Input/Output process is made using the Gradio [9] library, which allows the fast development of GUIs, especially for machine learning models. The GUIs are served on a simple web interface.

A. Frame extraction

The frame extraction module takes in input a video and extracts a number of frames. Not every frame in the video is used, because it would be computationally expensive and not necessary since close frames usually have similar features and objects.
The initial idea to sample frames was to fix a specific number for each second of the video, but this approach was discarded due to the increasing amount of frames for longer videos. The final approach used in the project is to sample a number of frames based on a function of the input length video, specifically:

$$n = k \cdot log_{10}(s)$$

with n being the number of frames, s the length of the video in seconds and k a constant factor given in input. This approach was chosen to have a logarithmic growth of the number of frames, which is more computationally efficient and still provides a good representation of the video.

Figure 2 shows the growth of the number of frames extracted based on the length of the video in seconds, with different values of k, compared to the function $n = 2 * s$ that takes one frame each half second.
The extraction of frames is done using FFMPEG [6], a versatile and easy CLI tool that is capable of processing audio and video files.
In case of image input, this module is not used.
The output of this module consists in a path to the folder containing the extracted frames and the number of frames extracted.

B. Object detection

The object detection module detects objects in the frames extracted by the previous module. In case of image input, this module performs the object recognition only on the single image. The module uses the YOLOv9 [4] model, which is a state-of-the-art and widely used object detection model that can run in short time and provide good results. The output of this module consists of a list of objects detected in each frame, as well as a set of all objects detected in the batch.

C. Best frame selection

This module is designed to select the best frame from the ones extracted by the first module. The main reason behind this choice is the computational intensity coming from the frame description module, that takes $\sim 5 - 7$ seconds to process a single frame. Thus, passing all the frames would generate a sensitive delay in the process of each video.
The criterion used to select the best frame has been object of many tests, due to the fact that it is a difficult to generalize task that is highly dependent on the input video. Our final proposed solution picks the best frame using a KMedoid algorithm over a latent representation of the images. The latent representation is obtained by passing the image through ResNet18 [7], which is a widely used and efficient image classification model. Since the task is not classification, the ResNet module is used to only get the latent representation of the input, of size $512 \times 16 \times 29$. The said representation is then reshaped and used to cluster the video and calculate the medoid. In particular, the KMedoid algorithm is used with $k = 1$ to get the medoid of the complete.
The output of this module is the path and index of the selected frame.

D. Frame description

For this crucial task different options of image to text models were evaluated and tested. The main source used for models is the Hugginface text-to-image section [10]. In the end, the selected model for this task is MoonDream2 [8], a image-text to text model that is able to produce the answer to a question over the image. In our case, the used question is *"Describe shortly the image and give it a mood that describes it."* that, over different tests, provides good results combined with the following modules. The choice switched to an image-text to text model since it provides a wider inference than image to text models, that can be used to condition the produced description to better fit our goals.
The output of this module is the description of the input image.

E. Prompt enhancer

The prompt enhancer module is only composed of a single string interpolation that combines the outputs of both the frame description and the object detection modules. An initial option was to combine them using a lightweight Large Language Model, but we decided not to include another model to avoid the increase in computational cost. The output of this module is the composed prompt, that will be fed in input to the audio generation module.

F. Audio generation

The audio generation module feeds the input to the AudioLDM2 model [11], which is a state-of-the-art audio generation model that can produce audio from the provided prompt. The model can take as input negative prompts as well, that corresponds to behaviors and keywords we want to avoid. This slightly improves the output overall, generating a more enjoyable audio that can better fit the video itself.
However, AudioLDM2 has a complex structure that impacts the computational cost of the system. With the available systems, we could not run it on local machines. More details on this can be found in the **Deployment** section of this report.
The output of this module is the output path containing the generated audio.

G. Reconstructor

The reconstructor module is the final module of the standard pipeline, that produces the output of the system. It combines the generated audio with the initial input video using FFMPEG [6] to produce the final enhanced version of the video. The output is the path to the final video, that is served to the user via the Gradio interface.

V. Deployment

The deployment of the system is done using Python 3.10.12 on Google Colab [5]. The first intention was to whole process run on a local machine, but it quickly became clear that this is not possible due to the high hardware requirements that audio generation models have. The next idea was to serve the audio generation module over an API, loading it on the machine provided at IST. This hypotesis got aswell discarded for lack of sufficient resources.
The Gradio interface is accessed through a public link, which is generated by the library itself. The link is accessible by anyone and can be used to interact with the system.
The information about the used libraries and technologies can be found in the **Architecture** section of this report.

VI. Implemented features

At the moment the project is capable of generating music tracks or music and audio tracks with a defined balance between these two audios. It can also produce music or audio and music from an image, with the resulting video lasting for the specified duration. The high modularity of the system allows for easy creation of new pipelines that can be used to adapt the system to different scenarios and requirements. Moreover, each module is independent and can be used separately from the others, allowing for a high level of flexibility in the system. The entire system is customizable and can be easily adapted to different use-cases based on the user requirements, such as the choice of a different model, a different algorithm or a different part of the pipeline.

VII. Evaluation and benchmarks

Evaluation of our task has no precise metrcis, since the output is highly subjective that depends on how a music fits on a video or not in a personal opinion.
We could have done some checks on the effective quality of the objects detection given by the YOLOv9 model or the description of the image given by the MoonDream2 model, but we decided not to do it since it's not the main goal of the project. A survey could have been done to evaluate the quality of the system, but the time constraints of the project did not allow us to do it.
In our opinion, the resulting audios in our test videos were fitting enough to be considered as a good result.
Speaking of benchmarks, we conducted a few tests to assess the time performance of the system. Videos of different time durations have been processed by the system under the same hardware conditions. The required time for the complete process has been recorded, and results are shown in Figure 3.

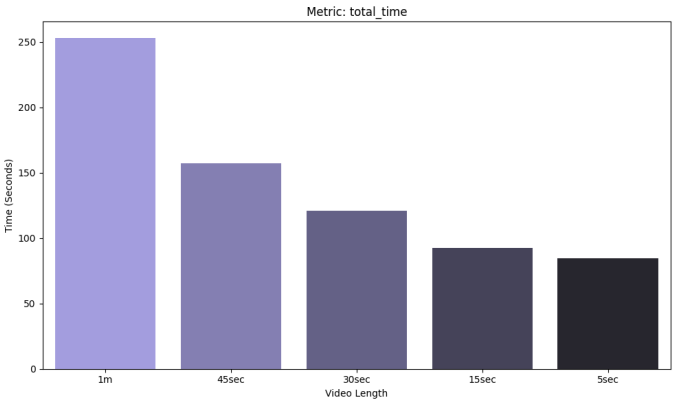

Fig. 3. Total time performance of the system.

VIII. Limitations and challenges

The main limitation of the project was the computational cost of the audio generation model, both in terms of time and hardware requirements. Moreover, having to use Google Colab as the main platform for the deployment of the system, we had to face the limitations of the platform itself, that is not very suitable for running *gradio* interfaces and long-running processes. Speaking of the audio generation model itself, it is very dependent on the prompt given in input and so the challenge was to find the best prompt that could be used to condition the audio generation. This involved a lot of prompting and testing with *moonDream2* model, that is continuously being updated and improved over time with updates on a weekly or monthly basis. This can be both a *good* and a *bad* thing, since the model can improve over time but can also change its behavior and output, making the system less reliable. Of course, a solution to this problem could be to stick with a specific version of the model, but this would limit the system.

This lead to another limitation of the project, that is the correct choice of the *best frame* of the video. This choice is relying on the KMedoid algorithm, that is not always able to select the best frame of the video. However, the problem is not easy and doesn't have a naive solution, so we decided to keep this choice in the project, being a good compromise between time and quality of the output.

IX. Conclusions and future work

The project was a real challenge that was not easy at all. The main goal was to create a system that could generate audio from a video, and we achieved this goal. The system is modular, scalable and easily adaptable to different scenarios, and it can be used over video or image inputs. The main challenge was finding a good approach that could balance efficiency and effectiveness of the system, and we believe that we achieved this goal. Of course, the system is not perfect and can be improved in many ways, but the fact that we had such small amount of time to develop it, we are satisfied with the results.

Future improvements are mainly focused on the audio generation model, that is the most critical part of the system. The correct creation of the prompt is the key to make the system work properly, and since text is involved, a proper text generation model trained on the specific task could be used to improve the quality of the output. Moreover, the system could be improved by using different models for both music and sound effects, to have a wider range of outputs. The main reason we didn't use different models is the computational cost of the model loading and memory usage, that would have been too high for the available resources.

References

[1] OpenAI ChatGPT introduction.
[2] OpenAI DALL-E introduction.
[3] YouTube copyright policies.
[4] Yolov9 documentation and Yolov9 paper reference.
[5] Google Colab main page.
[6] FFMPEG website.
[7] ResNet18 implementation on PyTorch.
[8] MoonDream2 model card on HugginFace.
[9] Gradio library main page.
[10] HuggingFace image-to-text section.
[11] AudioLDM2 documentation and paper.