VOIMO

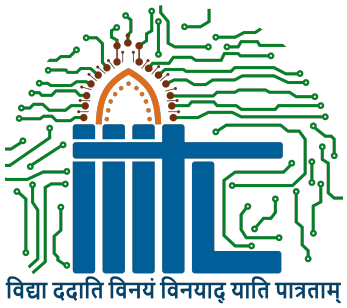
A project report submitted in partial fulfillment of the requirements for the award of the degree of

B.Tech. in Computer Science

by

Aryan Kumar LCI2020004
Harsh Singh LCI2020003
Aman Garg LCI2020031

under the guidance of
Dr. Vishal Krishna Singh



Indian Institute of Information Technology, Lucknow
May 2023

Declaration of Authorship

I/we, **Aryan Kumar LCI2020004, Harsh singh LCI202003, Aman Garg LCI2020031**, declare that the work presented in "**Video Output from Interpretive Music Operation(VOIMO)**" is my/our own. I/we confirm that:

- This work was completed entirely while in candidature for B.Tech. degree at Indian Institute of Information Technology, Lucknow.
- Where I/we have consulted the published work of others, it is always cited.
- Wherever I/we have cited the work of others, the source is always indicated. Except for the aforementioned quotations, this work is solely my/our work.
- I have acknowledged all major sources of information.

Signed:

(Harsh singh) (Aryan Kumar) (Aman Garg)

Date:

CERTIFICATE

This is to certify that the work entitled "**VOIMO**" submitted by **Harsh singh LCI2020003, Aryan Kumar LCI2020004, Aman Garg LCI2020031** who got his/her name registered on **Jul 2020** for the award of B.Tech. degree at Indian Institute of Information Technology, Lucknow is absolutely based upon his/her own work under the supervision of **Dr. Vishal Krishna Singh**, Department of Computer Science , Indian Institute of Information Technology, Lucknow - 226 002, U.P., India and that neither this work nor any part of it has been submitted for any degree/diploma or any other academic award anywhere before.

Dr. Vishal Krishna Singh
Faculty In-Charge *Training* And *Placements*
Indian Institute of Information Technology, Lucknow
Pin - 226002, INDIA

Acknowledgements

We would like to express our gratitude towards Dr. Vishal Krishna Singh Dr. Abhinesh Kaushik Niharka Anand Dr.Naveen Saini for his valuable suggestions and encouragement in completion of our project. We would also extend our thanks to Dr. Deepshika Agarwal and Dr. Mainak Adhikari and our honourable Dean, Dr. Dhananjoy Dey for their support in the accomplishment of our project. We would also be thankful to our director Dr. Arun Sherry for providing all the required facilities in completion of this project. We would like to extend our deep appreciation to all my group members, without their support and coordination we would not have been able to complete this project.

Lucknow
May 2023

ABSTRACT

VOIMO is an innovative project that takes audio as input and generates a relevant video based on the audio lyrics. The main goal of the project is to provide creative visuals for audio content, and enhance user experience.

The implementation of this project faced various challenges like GPU access, prompt generation using LLM, merging of backend with frontend, audio waveform and cropping, model selection, etc. These challenges were resolved using various tools like Collab, Kaggle, fast API, Flask API, and self-experimentation of large numbers of models of CIVIT.

The project uses LLM (Large Language Model) to generate prompts from audio lyrics, which are then used to generate video content. The backend and frontend of the project were integrated using Fast API and Flask API, respectively. The audio waveform and cropping were achieved using Wave Surfer.

Several models of CIVIT were experimented with to select the best one for this specific application.

In conclusion, VOIMO is a promising project that has successfully tackled the challenge of generating creative video content based on audio lyrics. The project provides a new way of consuming audio content and enhances user experience.

Contents

Chapter 1

Introduction

1.1 Overview of VOIMO

VOIMO is an innovative project that aims to revolutionize the way we create and experience multimedia content. By taking audio as input and generating relevant video content based on the audio lyrics, VOIMO has the potential to transform the music industry, advertising, and various other sectors that rely heavily on visual storytelling.

The core idea behind VOIMO is to utilize state-of-the-art artificial intelligence techniques, such as natural language processing and computer vision, to analyze the audio input and generate a video output that best represents the essence of the audio lyrics.

1.2 Challenges and Solutions

This process involves several challenges, including accessing GPU resources, generating prompts using large language models, merging backend and frontend components, handling audio waveforms and cropping, and selecting the most suitable models for the task.

To overcome these challenges, our team has employed various tools and technologies, such as Google Colab and Kaggle for GPU access, large language model APIs for prompt generation, Flask API and FastAPI for backend-frontend integration, WaveSurfer for audio waveform manipulation, and experimenting with numerous models from the CIVIT library.

1.3 Frontend Design

The frontend of VOIMO consists of several user-friendly pages, including signup and login, home, upload, about us, project details, and logout. These pages have been designed to provide a seamless user experience while interacting with the platform.Technology used React.

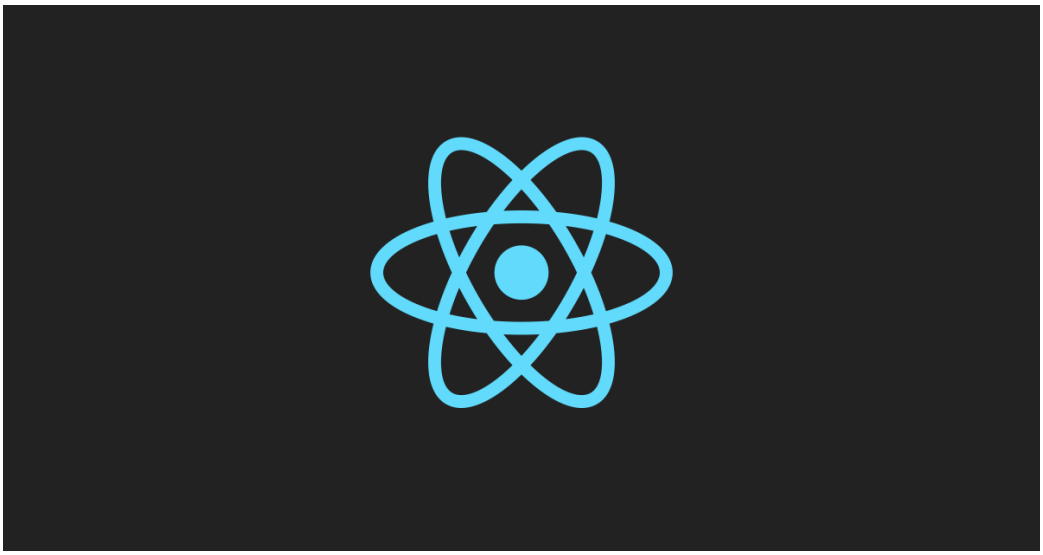


Figure 1.1: React

1.4 Backend Implementation

Figure 1.2: Python

The backend of VOIMO encompasses various functionalities, such as:

Utilizing the Spleeter library for splitting audio into vocals and accompaniment. Employing Whisper[1] to transcribe vocals into text. Using large language model APIs to convert text into prompts. Leveraging Python libraries to generate keyframes. Applying the stable diffusion model to convert text into images, which are generated according to video frames. Integrating FFmpeg to synchronize audio with video. Experimenting with various models and APIs to enhance the overall performance of the project.

Figure 1.3: Open AI whisper

1.5 Report Structure

In this report, we will discuss the details of VOIMO's development process, the challenges faced, and the solutions implemented to create a

groundbreaking multimedia platform. We will also explore potential future improvements and applications of this technology in various industries. The report is organized into the following chapters:

Introduction: Provides an overview of VOIMO, its challenges, and solutions. Literature Review: Presents related work and technologies in the field of AI-generated multimedia content. System Design and Architecture: Describes the overall design and architecture of VOIMO, including frontend and backend components. Implementation Details: Explains the technical aspects of implementing VOIMO, such as tools, technologies, and algorithms used. Results and Evaluation: Showcases the performance of VOIMO and evaluates its effectiveness in generating relevant video content based on audio lyrics. Future Work and Conclusion: Discusses potential improvements and future applications of VOIMO, and concludes the report.

Chapter 2

Literature Review

In this chapter, we will discuss the existing literature and research related to the development of VOIMO, an application that takes audio input and generates a relevant video based on the audio lyrics. The review will cover various areas, including GPU access, prompt generation using LLM, back-end and frontend integration, audio waveform and cropping, and model selection.

2.1 GPU Access

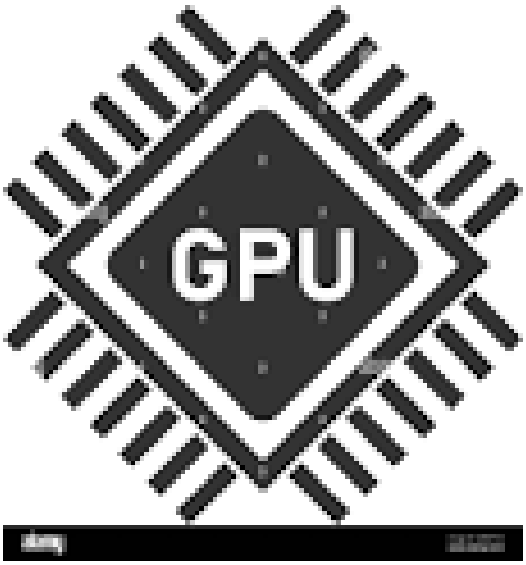


Figure 2.1: GPU

GPU (Graphics Processing Unit) access is critical for training and running deep learning models, as it enables faster computation and parallel processing. Several platforms have been studied for providing GPU access, such as Google Colab and Kaggle. These platforms offer free access to GPUs, making it easier for developers to experiment with different models and techniques without incurring high costs.

2.1.1 Google Colab

Figure 2.2: Colab

Google Colab is a cloud-based platform that allows users to run Jupyter Notebooks with free access to GPUs like NVIDIA Tesla K80, P4, T4, and P100. It has been widely used by researchers and developers for training and deploying machine learning models. The platform supports TensorFlow, PyTorch, and other popular libraries.

2.1.2 Kaggle

Kaggle is a data science and machine learning platform that offers free GPU access through its kernel environment. It supports various machine learning libraries, including TensorFlow, Keras, and PyTorch. Kaggle also provides a vast database of datasets, which can be beneficial for training and testing models.

Figure 2.3: Kaggle

2.2 Prompt Generation using LLM

Large Language Models[2] (LLMs) have demonstrated impressive capabilities in natural language understanding and generation. By fine-tuning these models with specific prompts, they can generate contextually relevant outputs. In the context of VOIMO, LLMs are employed to convert transcribed text into prompts that guide the generation of keyframes and video content.

2.2.1 OpenAI GPT-3

OpenAI's GPT-3 (Generative Pre-trained Transformer 3) is an LLM that has shown exceptional performance in various natural language processing tasks. The model can be fine-tuned to generate prompts for video content creation based on the input audio lyrics.

2.3 Backend and Frontend Integration

Integrating the backend and frontend components of VOIMO is crucial for seamless user experience. Flask API and FastAPI have been explored as potential solutions for connecting the two components.

2.3.1 Flask API

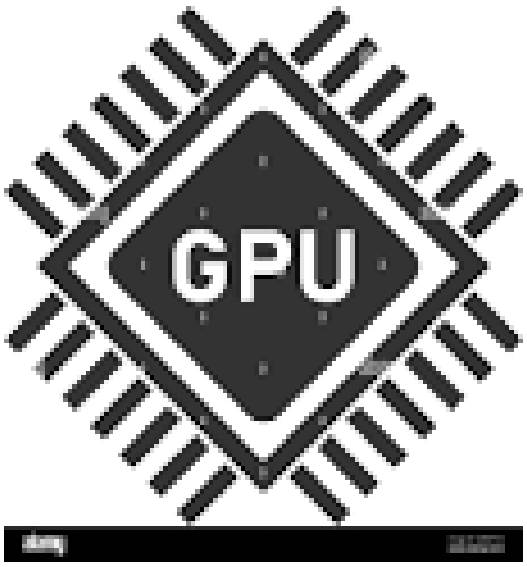


Figure 2.4: GPU

Flask is a lightweight web framework for Python that allows developers to create RESTful APIs easily. It can be used to build the backend of VOIMO, enabling communication between the frontend and the various machine learning models employed in the project.

2.3.2 FastAPI

Figure 2.5: Fast API

FastAPI is another web framework for Python that offers high performance and easy-to-implement APIs. It supports asynchronous programming and type annotations, which can improve code readability and maintainability.

2.4 Audio Waveform and Cropping

Processing the input audio is a crucial step in VOIMO. This involves visualizing the audio waveform and cropping the audio to select the desired portion for video generation. The Wavesurfer library has been identified as a suitable tool for this purpose.

2.4.1 Wavesurfer

Figure 2.6: wave surfer

Wavesurfer.js is a customizable audio waveform visualization library that allows users to interact with the audio by zooming, panning, and selecting specific regions. It can be integrated into the frontend of VOIMO to enable users to crop and select the desired portion of the input audio.

2.5 Model Selection

Several models and APIs have been experimented with to find the best combination for generating relevant video content based on the input audio lyrics. This includes models for keyframe generation, image synthesis, and audio-video synchronization.

2.5.1 Keyframe Generation

Python libraries and algorithms have been used to generate keyframes based on the prompts provided by the LLM. These keyframes serve as a foundation for generating the final video content.

2.5.2 Stable Diffusion Model

Figure 2.7: stable diffusionI

The Stable Diffusion[3] Model is a generative model capable of synthesizing images based on textual input. In VOIMO, it is used to convert the keyframes generated from the prompts into images that match the video frames.

2.5.3 FFmpeg

FFmpeg is a multimedia framework that can be used to synchronize the generated video frames with the input audio. It is employed in VOIMO to

create the final video output, ensuring that the visual content matches the audio lyrics.

In conclusion, the literature review has provided insights into the various technologies and models that can be employed to develop VOIMO, an application that takes audio input and generates relevant video content based on the audio lyrics. By leveraging these tools and techniques, the project aims to create a seamless and intuitive user experience while generating high-quality video outputs.

Chapter 3

Methodology

This chapter discusses the methodology employed in the development of the VOIMO project. The project aims to take audio input and generate a relevant video based on the audio lyrics. The challenges faced during the project development are addressed, and the solutions implemented to resolve these issues are explained. The chapter is organized into the following subsections:

3.1 System Overview

The VOIMO project consists of two main components: the frontend and the backend. The frontend is responsible for user interaction, while the backend processes the audio input and generates the corresponding video output. The system's architecture is designed to ensure seamless integration between the frontend and backend components.

3.2 Frontend Development

The frontend of the VOIMO project comprises several web pages, including signup and login pages, home page, upload page, about us page, project details page, and logout functionality. Each page serves a specific purpose in the user interface:

- Signup and Login pages: These pages allow users to create an account and log in to access the system.
- Home page: This page provides an overview of the project and its features.

Figure 3.1: Login

- Upload page: Users can upload their audio files on this page.
- About us page: This page presents information about the team behind the project.
- Project details page: This page displays the progress and status of the user's projects.
- Logout: Users can log out of their accounts using this functionality.

3.3 Backend Development

The backend of the VOIMO project is responsible for processing the audio input and generating the relevant video output. The following steps outline the backend workflow:

1. Audio splitting: The Spleeter library is used to split the audio into vocal and accompaniment tracks.
2. Vocal transcription: The Whisper library is employed to transcribe the vocal track into text.
3. Prompt generation: The Large Language Model (LLM) API is utilized to convert the transcribed text into a prompt.

4. Keyframe generation: A Python library is used to generate keyframes for the video.

5. Text-to-image conversion: The Stable Diffusion model is applied to convert the generated prompt into images, which are then used as video frames.

6. Audio-video synchronization: FFmpeg is used to synchronize the audio with the generated video.

7. Model selection and optimization: Various models and APIs are tested and compared to ensure the best possible output for the project.

3.4 Challenges and Solutions

During the development of the VOIMO project, several challenges were encountered and subsequently resolved:

- GPU access: Limited access to GPU resources was addressed by using collaborative platforms such as Google Colab and Kaggle.
- Prompt generation using LLM: The issue of generating prompts was resolved by employing the Large Language Model API.
- Merging frontend and backend: Flask and FastAPI were used to integrate the frontend and backend components seamlessly.
- Audio waveform and audio cropping: The Wavesurfer library was used to handle audio waveforms and cropping functionalities.
- Model selection: Extensive experimentation with various models from the CIVIT repository helped identify the most suitable models for the project.

By addressing these challenges, the VOIMO project successfully achieves its goal of converting audio input into relevant video output based on the audio lyrics.

Chapter 4

Simulation and Results

In this chapter, we present the simulation process and the results obtained from our project, VOIMO. The project's main objective is to take audio as input and generate a relevant video based on the audio lyrics. We have successfully addressed various challenges and integrated different technologies to achieve the desired outcome. The following sections detail the simulation process and the results obtained.

4.1 Simulation Process

The simulation process for our project can be divided into several steps:

Audio input: The user uploads an audio file through the frontend interface. Audio processing: The backend processes the audio by splitting it into vocals and accompaniment using the Spleeter library. Transcription: The Whisper tool is used to transcribe the vocal part into text. Prompt generation: The Large Language Model (LLM) API is employed to convert the transcribed text into a prompt. Keyframe generation: A Python library is utilized to generate keyframes based on the prompts. Image generation: The Stable Diffusion Model is used to generate images according to the video frames. Audio-video synchronization: FFMpeg is used to sync the generated images with the audio, creating the final video output.

4.2 Results

The results of our project can be analyzed in terms of the quality and relevance of the generated videos. We have tested our system with various audio inputs, and the following observations were made:

Audio transcription: The Whisper tool provided accurate transcriptions of the vocal parts in most cases, resulting in relevant prompts. Prompt generation: The LLM API effectively converted the transcribed text into meaningful prompts that guided the video generation process. Keyframe generation: The Python library successfully generated keyframes based on the prompts, ensuring smooth transitions between video scenes. Image generation: The Stable Diffusion Model produced high-quality images that matched the video frames, resulting in visually appealing videos. Audio-video synchronization: FFMpeg effectively synced the audio with the generated images, ensuring a seamless viewing experience.

4.3 Limitations and Future Work

While our project has achieved promising results, there are some limitations that can be addressed in future work:

Real-time processing: The current implementation may not be suitable for real-time video generation due to the time-consuming nature of some processes, such as image generation and audio-video synchronization. Optimizing these processes can enable real-time applications. Customization options: Providing users with more customization options, such as selecting specific styles or themes for the generated video, can enhance the user experience. Integration of additional models and APIs: Exploring and integrating other models and APIs can further improve the quality and relevance of the generated videos. In conclusion, our project VOIMO has successfully demonstrated the ability to generate relevant videos based on audio lyrics by integrating various tools and technologies. The results obtained are promising, and future work can focus on addressing the limitations and enhancing the system's capabilities.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this project, we have successfully developed VOIMO, a system that takes audio input and generates a relevant video based on the audio lyrics. We have overcome several challenges, such as GPU access, prompt generation using LLM, merging of backend and frontend, audio waveform and audio cropping, and model selection.

Our frontend consists of various pages like signup, login, home, upload, about us, project details, and logout, which work according to their names. The backend is designed with multiple libraries and APIs to process the audio input, transcribe vocals into text, generate prompts, create keyframes, convert text to images, sync audio with video, and experiment with different models to enhance the project's performance.

5.2 Future Work

There is always room for improvement and expansion in any project. Some possible future work for VOIMO includes:

5.2.1 Improving Audio Processing

The current implementation can be further enhanced by exploring more advanced audio processing techniques or libraries to improve the quality of the audio transcription and the separation of vocals and accompaniment.

5.2.2 Enhancing Prompt Generation

The prompt generation using LLM can be refined by incorporating more context-aware and semantically relevant information from the audio lyrics. This would result in more accurate and meaningful prompts for the video generation process.

5.2.3 Optimizing Video Generation

The video generation process can be optimized by experimenting with more advanced models or techniques, such as GANs (Generative Adversarial Networks) or VAEs (Variational Autoencoders), to produce higher quality and more visually appealing videos.

5.2.4 User Interface Improvements

The user interface can be made more intuitive and user-friendly by implementing features such as drag-and-drop for file uploads, real-time progress updates during video generation, and more customization options for the generated videos.

5.2.5 Scalability and Performance

To handle a larger number of users and requests, the system can be deployed on cloud-based platforms with auto-scaling capabilities. Additionally, optimizing the backend code and implementing caching mechanisms could further improve the overall performance.

In conclusion, VOIMO has demonstrated its potential as a tool for generating videos based on audio lyrics. With future enhancements and optimizations, it can become a valuable asset for content creators, musicians, and artists in various fields.

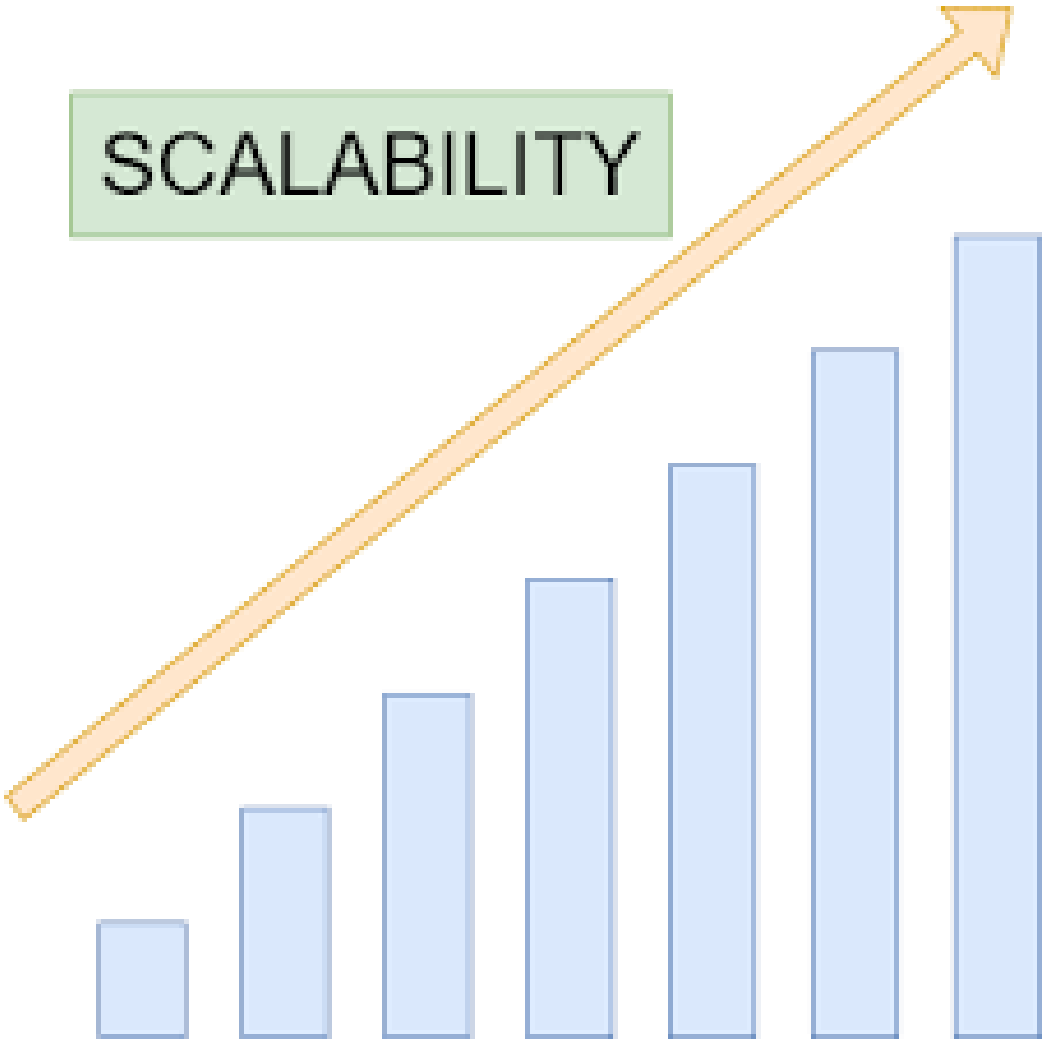


Figure 5.1: scalability

Appendix A

Contributions

Aman Garg's LCI2020031

- Developed and designed the frontend of the application using React
- Implemented various functionalities in the front-end including audio player and video rendering
- Conducted thorough debugging and testing of the front-end code
- Collaborated with other members of the team to achieve smooth integration of front-end with back-end
- Assisted in creating user documentation

Harsh Singh LCI2020003

- Automate Keyframe Generation
- Implement model
- Audio sinking of video frame with Audio
- Background Noise clearing using Spliter
- Collaborated with other members of the team to achieve smooth integration of front-end with back-end
- Assisted in creating user documentation

Aryan Kumar LCI2020004

- Created Fast API to link frontent and backend
- Implement model
- Add feature of crop Audio
- Add Theme selection feature
- Collaborated with other members of the team to achieve smooth integration of front-end with back-end

- Assisted in creating user documentation

All members of the team contributed via code reviews, discussion, and feedback during weekly meetings.

Appendix B

Frontend - For Submitting Details

The following code snippet shows the frontend code for handling the submission of details in a form.

```
const handleSubmit = async (event) => {
event.preventDefault(); // prevent the default form submission behavior
const formData = new FormData(event.target); // get the form data
const url = "https://ff64-35-247-163-73.ngrok-free.app/submit_form"; // repla
formData.delete("audio");
urlToBuffer(audioURL)
    .then((buffer) => {
        const croppedBuffer = cropBuffer(buffer, time[0], time[1]);
        const wavBytes = bufferToWav(croppedBuffer);
        const wav = new Blob([wavBytes], { type: "audio/wav" });
        const audioTemp = new File([wav], "my-audio-file.wav", { type: "audio
        formData.append("audio", audioTemp, "temp.wav");
        setLoading(true);
        fetch(url, {
            method: "POST",
            body: formData,
        })
        .then((response) => {
            console.log(121);
            // The response object contains metadata about the file
            const filename = "temp.mp4";
            // Use the blob() method to extract the file content as a Blob ob
            return response.blob().then((blob) => ({ filename, blob }));
            })
        .then((file) => {
            // Use URL.createObjectURL() to generate a URL for the file conte
            const url = URL.createObjectURL(file.blob);
            // Create an <a> element and programmatically click it to downloa
            const a = document.createElement("a");
            a.href = url;
            a.download = file.filename;
            document.body.appendChild(a);
            a.click();
            document.body.removeChild(a);
```

```
            setLoading(false)
        })
        .catch((error) => {
            setLoading(false)
            console.log(error);
        })
    })
    .catch((error) => {
        console.log(error);
    });
};
```

Frontend - For Recording Audio

The following code snippet shows the frontend code for recording audio.

```
const startRecording = () => {
    setAudioURL(null);
    setRecordingButtonClass("recording-btn recording");
    navigator.mediaDevices
        .getUserMedia({ audio: true })
        .then((stream) => {
            const recorder = new MediaRecorder(stream);
            recorder.start();
            setRecording(true);
            setMediaRecorder(recorder);
            recorder.addEventListener("dataavailable", (event) => {
                chunks.push(event.data);
            });

            recorder.addEventListener("stop", () => {
                const blob = new Blob(chunks);
                const file = new File([blob], "recording.wav", { type: "audio
                // audioInputRef.current.files = [file];
                const dataTransfer = new DataTransfer();
                dataTransfer.items.add(file);

                const fileList = dataTransfer.files;
                audioInputRef.current.files = fileList;
                const url = URL.createObjectURL(blob);
                setAudioURL(url);
                });
        })
        .catch((error) => {
            console.log(error);
        });
};
```

Backend

The backend of the code is implemented using FastAPI, a modern, fast (high-performance) web framework for building APIs with Python. The backend server receives the form data submitted from the frontend and processes it to generate a video file as per the provided inputs. The following code snippet provides an overview of the backend implementation:

```
from fastapi import FastAPI, Request, File, Form
from fastapi.responses import FileResponse
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
import sys
import os
sys.path.extend([
    'src/taming-transformers',
    'src/clip',
    'stable-diffusion/',
    'k-diffusion',
    'pytorch3d-lite',
    'AdaBins',
    'MiDaS',
])
import load_model
import torch
from mainOld import musictoVid

app = FastAPI()
origins = ["*"]
app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

custom_checkpoint_path = '/content/drive/MyDrive/AI/models/dreamlikeDiffusion10_10.c
models_path = "/content/models"
os.makedirs(models_path, exist_ok=True)
model = load_model.model(models_path,custom_checkpoint_path=custom_checkpoint_path)
```

```
device = torch.device("cuda") if torch.cuda.is_available() else torch.device(
model = model.to(device)
class Audio(BaseModel):
    option: int
    upscale: bool = False
    artist: str = ""
    theme: str = ""

@app.get('/')
async def home():
  return {'mes':'as'}

@app.post("/submit_form")
async def submit_form(audio: bytes = File(...), artist: str = Form(''), theme
    # Save audio to current directory
    with open("audio.wav", "wb") as f:
        f.write(audio)

    # Print other form inputs
    print(f"Text input 1: {theme}")
    print(f"Text input 2: {artist}")
    print(f"Radio button: {option}")
    print(f"Checkbox: {upscale}")
    video_path = musictoVid(device,model,'/content/audio.wav',theme,artist,in
    return FileResponse(video_path)
    return {"message": "Form submitted successfully!"}
```

Bibliography

[1] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.04356*, 2022.

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.