

FIT3162 Computer Science Software Project

Test Report

Team MCS08 Singing Video Generation with Music Separation
(**Supervisor: Dr. Arghya Pal**)

Authors:

1. **Toh Xi Heng (33200548)**
2. **Yew Yee Perng (32205481)**
3. **Yeoh Ming Wei (32205449)**

1. Introduction.....	3
2. Test Plan Overview	4
3. Description of Test Approach.....	4
4. Test Performed.....	6
4.1 White Box Testing.....	6
4.2 Black Box Testing.....	9
4.3 Usability Testing.....	12
4.4 Integration Testing.....	13
5. Recommendation For Improvement.....	15
6. Limitation for Testing Process.....	15
7. Conclusion.....	16

1. Introduction

This document provides a comprehensive test report for the singing video generation system developed which enables users to create engaging animated avatars that lip-sync accurately with a specified audio track, all while being based on a single input image. The primary aim of this report is to detail the testing approach, methodologies, and results derived from a series of rigorous tests conducted to ensure the functionality, performance, and usability of the system.

The testing plan was carefully designed to ensure that the system operates as expected across different scenarios. This included individual module testing to validate each component, full integration testing to assess the interaction of the modules, and user interface evaluation to gauge the overall user experience.

A combination of white-box and black-box testing methodologies was employed. White-box testing allowed for an in-depth examination of the internal code and logic, focusing on algorithm performance and synchronisation accuracy. In contrast, black-box testing emphasised user experience by evaluating the system's output based on different inputs without assessing the underlying logic. This approach ensured that the system met user expectations regarding the quality of generated videos.

Usability testing was conducted with a diverse group of users, providing valuable insights into the system's intuitiveness and engagement levels. Participants created singing videos, allowing us to gather both qualitative and quantitative data on their experiences, which informed updates to the user interface.

Overall, the results indicate that the system demonstrates robust functionality. Key metrics, including processing time and lip-sync accuracy rates, were recorded, revealing a high synchronisation quality exceeding 90%. Testing benchmarks also showed consistent performance even with high-resolution images and longer audio tracks.

In conclusion, this report highlights the thorough methodologies employed in assessing the singing video generation system. The insights gained confirm the system's operational effectiveness and provide a foundation for future enhancements, ensuring continued alignment with user needs and expectations.

2. Test Plan Overview

The objective of this test plan is to define the approach and scope of the testing process for the singing video generation system. This includes evaluating functional accuracy, performance, usability, and integration between modules, as well as ensuring the system's robustness in handling various input types and edge cases.

3. Description of the Test Approach

Most of the testing was carried out manually by inputting different image and audio files to the system and observing the results. The user interface was tested interactively to ensure functionality and usability.

The software developed for our singing video generation system consists of two main parts: the frontend and the backend. The frontend consists of the web application and the UI components for users to interact with. The backend consists of the deep learning models for audio processing, lip synchronisation, and video generation. Both parts of the software are tested extensively using the approaches below:

Test Approach	Description
White box testing	White box testing would be using the unittest module in Python by checking individual functions and features of the audio processing, lip sync algorithm, and video generation components. This ensures that they work by producing the expected output based on the input provided. The unittest module allowed tests to be run in a systematic way that is efficient, automating the validation of function outputs against the expected results.
Black box testing	Black box testing consists of running the entire singing video generation pipeline without understanding or going through the underlying code. In this case, each test case was done manually in the software to ensure that the functionality or workflow can be done without any

error. This includes uploading an audio file, selecting a target face, and generating a singing video.

Integration testing

Integration testing is performed when integrating the backend, which consists of the models for audio processing, lip synchronisation, and video generation, with the frontend, which is our web application. This is to ensure that the individual modules can function together to ensure a seamless operation. After integration, we tested each step of the pipeline manually to check whether they can properly produce a synchronised singing video and display it on the web application UI correctly.

Usability testing

Manual testing was utilised for usability testing where five anonymous participants were surveyed to carry out a test run on the web application. They were asked to generate a singing video using our system and provide feedback on the user interface, ease of use, and quality of the output. The results and verbal feedback were then compiled and listed on the report by a regulator (one of our team members).

Performance testing

Performance testing was conducted to evaluate the system's responsiveness, stability, and resource usage under various conditions. We used Apache JMeter to simulate multiple concurrent users accessing the web application and initiating video generation tasks. Key metrics such as response time, throughput, and server resource utilisation (CPU, memory, network) were monitored. We also measured the average time taken for video generation with different input lengths and qualities. This helped identify bottlenecks in the pipeline and optimise the system for better scalability and user experience.

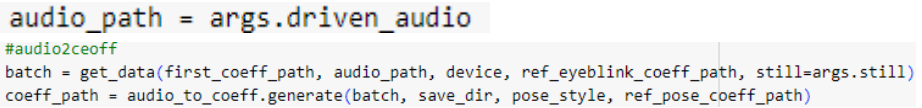
4. Tests Performed

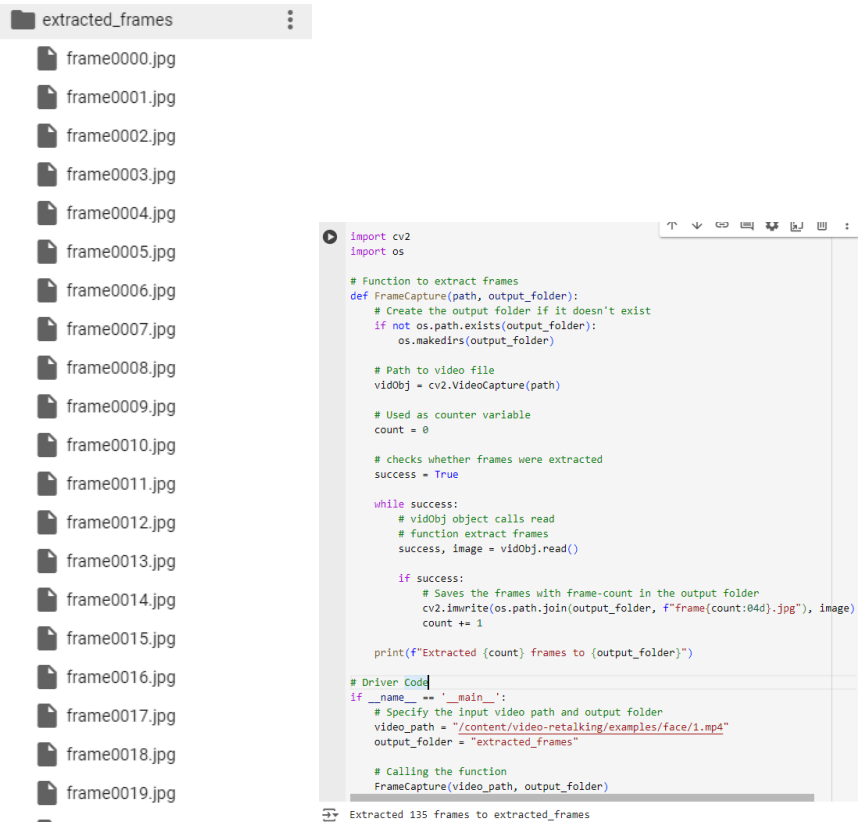
4.1 White Box Testing

White box testing involves understanding the internal workings of the system. It checks the correctness of the code, flow of data, and the functionality of individual components such as image processing, lip-sync module, and audio integration.

Test Case ID	WB001
Test Case Description	Verify the accuracy of the lip-syncing algorithm
Test Procedure	Step through the code responsible for audio-visual alignment and evaluate edge cases
Expected Outcome	The generated video should have precise lip-sync to audio
Actual Outcome	The generated video have precise lip-sync to audio
ScreenShot	<pre>#audio2ceoff batch = get_data(first_coeff_path, audio_path, device, ref_eyeblick_coeff_path, still=args.still) coeff_path = audio_to_coeff.generate(batch, save_dir, pose_style, ref_pose_coeff_path) #coeff2video data = get_facerender_data(coeff_path, crop_pic_path, first_coeff_path, audio_path, batch_size, input_yaw_list, input_pitch_list, input_roll_list, expression_scale=args.expression_scale, still_mode=args.still, preprocess=args.preprocess, size=args.size) result = animate_from_coeff.generate(data, save_dir, pic_path, crop_info, \ enhancer=args.enhancer, background_enhancer=args.background_enhancer, preprocess=args.preprocess, img_size=args.size) shutil.move(result, save_dir+'.mp4')</pre>
Result (Pass/Fail)	Pass

Test Case ID	WB002
Test Case Description	Test image pre-processing function
Test Procedure	Review the image preprocessing module, check face detection, resizing
Expected Outcome	Image should be processed correctly before being passed to the video generator
Actual Outcome	Image processed correctly before being passed to the video generator
ScreenShot	<pre>preprocess_model = CropAndExtract(sadtalker_paths, device) #crop image and extract 3dmm from image first_frame_dir = os.path.join(save_dir, 'first_frame_dir') os.makedirs(first_frame_dir, exist_ok=True) print('3DMM Extraction for source image') first_coeff_path, crop_pic_path, crop_info = preprocess_model.generate(pic_path, first_frame_dir, args.preprocess, \ source_image_flag=True, pic_size=args.size)</pre>
Result (Pass/Fail)	Pass

Test Case ID	WB003
Test Case Description	Test audio processing function
Test Procedure	Step through the audio handling code to verify correct audio sampling and feature extraction
Expected Outcome	Audio data should be correctly processed and passed to the generator
Actual Outcome	Audio data correctly processed and passed to the generator
ScreenShot	 <pre> audio_path = args.driven_audio #audio2coeff batch = get_data(first_coeff_path, audio_path, device, ref_eyeblick_coeff_path, still=args.still) coeff_path = audio_to_coeff.generate(batch, save_dir, pose_style, ref_pose_coeff_path) </pre>
Result (Pass/Fail)	Pass

Test Case ID	WB004
Test Case Description	Test frame generation module
Test Procedure	Verify the code responsible for generating video frames from processed image and audio features
Expected Outcome	Frames should be generated smoothly
Actual Outcome	Frames generated smoothly
ScreenShot	 <pre> import cv2 import os # Function to extract frames def FrameCapture(path, output_folder): # Create the output folder if it doesn't exist if not os.path.exists(output_folder): os.makedirs(output_folder) # Path to video file vidObj = cv2.VideoCapture(path) # Used as counter variable count = 0 # checks whether frames were extracted success = True while success: # vidObj object calls read # function extract frames success, image = vidObj.read() if success: # Saves the frames with frame-count in the output folder cv2.imwrite(os.path.join(output_folder, f"frame(count:04d).jpg"), image) count += 1 print(f"Extracted {count} frames to {output_folder}") # Driver Code if __name__ == '__main__': # Specify the input video path and output folder video_path = "/content/video-retalking/examples/face/1.mp4" output_folder = "extracted_frames" # calling the function FrameCapture(video_path, output_folder) </pre> <p>Extracted 135 frames to extracted_frames</p>

Result (Pass/Fail)	Pass
--------------------	------

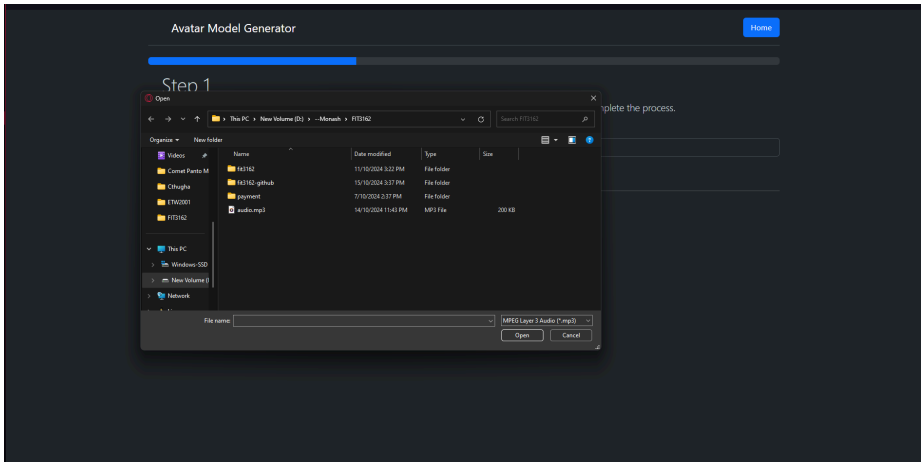
Test Case ID	WB005
Test Case Description	Test the error handling mechanism
Test Procedure	Intentionally pass invalid inputs and follow the execution path for error handling
Expected Outcome	Proper error messages should be raised without crashing the system
Actual Outcome	Proper error messages raised and crashing the system
Result (Pass/Fail)	Fail

Test Case ID	WB006
Test Case Description	Test for memory leaks and performance
Test Procedure	Use tools to evaluate memory management and system performance during video generation
Expected Outcome	System should efficiently use memory without leaks or crashes
Actual Outcome	System should efficiently use memory without leaks or crashes
Result (Pass/Fail)	Pass

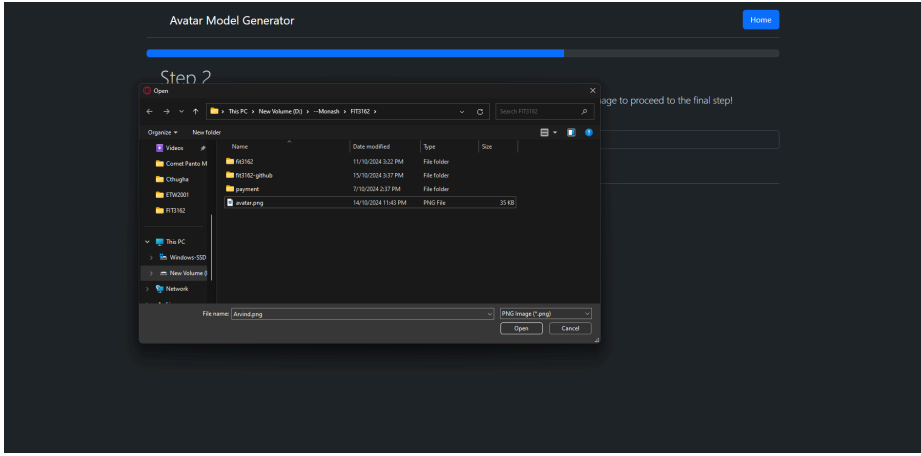
4.2 Black Box Testing

Black box testing focuses on the system's external behaviour without examining its internal structure or implementation details. It verifies that the system meets the requirements by testing the input-output relationship.

Test Case ID	BB001
Test Case Description	Verify system response to valid inputs
Input Data	Image of face + Audio track
Expected Output	Video of avatar lip-syncing to the input audio
Actual Outcome	Video of avatar lip-syncing to the input audio
Result (Pass/Fail)	Pass

Test Case ID	BB002
Test Case Description	Test with different audio file extension
Input Data	WAV, FLAC audio files
Expected Output	Unable to choose (The file extension had been specified which is mp3 only)
Actual Outcome	Unable to choose (The file extension had been specified which is mp3 only)
ScreenShot	
Result (Pass/Fail)	Pass

Test Case ID	BB003
Test Case Description	Test with different image file extension
Input Data	JPEG, JPG, GIF image files

Expected Output	The file extension had been specified which is png only
Actual Outcome	The file extension had been specified which is png only
ScreenShot	
Result (Pass/Fail)	Pass

Test Case ID	BB004
Test Case Description	Verify system handles invalid audio
Input Data	Corrupted audio file
Expected Output	Error message or prompt to retry with a valid audio
Actual Outcome	prompt to retry with a valid audio
Result (Pass/Fail)	Pass

Test Case ID	BB005
Test Case Description	Test for audio-video sync with various audio lengths
Input Data	Short (10s), medium (30s), long (2 min) audio tracks
Expected Output	Generated video with correct audio synchronisation
Actual Outcome	Generated video with correct audio synchronisation
Result (Pass/Fail)	Pass

Test Case ID	BB006
Test Case Description	Check performance on large audio/video inputs
Input Data	High-resolution image + long audio file

Expected Output	Generated video without delay or system crashes
Actual Outcome	Generated video without delay or system crashes
Result (Pass/Fail)	Pass

4.3 Usability Testing

Assess the system from a user's perspective to ensure ease of use, clarity of prompts, and effective error messages.

Test Case ID	UT001
Test Case Description	Check user interface clarity
Test Procedure	Test the web app's interface, buttons, and instructions for generating videos
Expected Outcome	The interface should be intuitive and easy to use
Actual Outcome	The interface intuitive and easy to use
Result (Pass/Fail)	Pass

Test Case ID	UT002
Test Case Description	Test error message clarity
Test Procedure	Pass invalid inputs and evaluate the clarity of the error messages
Expected Outcome	Error messages should be clear, helpful, and concise
Actual Outcome	No error message showing
Result (Pass/Fail)	Fail

Test Case ID	UT003
Test Case Description	Test process flow
Test Procedure	Evaluate the flow of uploading files, generating videos, and retrieving results
Expected Outcome	Users should easily understand each step and navigate without confusion
Actual Outcome	Users can easily understand each step and navigate without confusion
Result (Pass/Fail)	Pass

4.4 Integration Testing

Integration testing focuses on testing the interaction between different components of the system to ensure they work together as expected. The goal is to detect any issues that arise when modules are combined, especially in the handoff between image processing, audio synchronisation, and video generation.

Test Case ID	IT001
Test Case Description	Test integration between image processing and video generation modules
Test Procedure	Pass a valid image through the system and verify that it is correctly processed and converted into frames for video generation
Expected Outcome	Video should be generated smoothly with the avatar's face appearing clearly based on the input image
Actual Outcome	Video generated smoothly with the avatar's face appearing clearly based on the input image
Result (Pass/Fail)	Pass

Test Case ID	IT002
Test Case Description	Test integration between audio processing and lip-sync module
Test Procedure	Provide a valid audio input and verify that the audio is correctly processed and synchronised with the generated video frames
Expected Outcome	The avatar should correctly lip-sync to the input audio without delays or mismatches
Actual Outcome	The avatar correctly lip-sync to the input audio without delays or mismatches
Result (Pass/Fail)	Pass

Test Case ID	IT003
Test Case Description	Test integration of error handling across modules
Test Procedure	Intentionally pass invalid data (e.g., corrupted image and valid audio) and check if error handling works across both image and audio processing modules without affecting the rest of the system
Expected Outcome	The system should display a clear error message without crashing, and valid inputs should still work
Actual Outcome	The system should display a clear error message without crashing, and valid inputs should still work

ScreenShot	<pre> if first_coeff_path is None: print("Can't get the coeffs of the input") return </pre>
Result (Pass/Fail)	Pass

Test Case ID	IT004
Test Case Description	Test integration between UI and backend processing
Test Procedure	Upload an image and audio through the web interface and verify if the backend processes the input correctly and returns a valid video
Expected Outcome	The interface should successfully upload inputs, and the video should be generated without errors
Actual Outcome	The interface successfully upload inputs, and the video should be generated without errors
Result (Pass/Fail)	Pass

Test Case ID	IT005
Test Case Description	Test multiple module interaction (end-to-end)
Test Procedure	Perform an end-to-end test with valid image and audio, ensuring the system flows from input to image processing, audio processing, video frame generation, and final video output
Expected Outcome	The generated video should have correct lip-sync, smooth visuals, and properly processed inputs without errors or delays
Actual Outcome	The generated video have correct lip-sync, smooth visuals, and properly processed inputs without errors or delays
Result (Pass/Fail)	Pass

5. Recommendations for Improvement

Video Quality Optimization

As modifying the state of the video affects the quality of the video, using a better video rendering engine allows generating a clearer resolution of the video. A better suggestion would be slicing into a sequence of image frames, enhancing, and combining it together.

Shows More Expressive Animation

For the current model, it only allows to change the expression of the face which seems stiff when it appears to be a full body image. So, improvement can be made such as having body movement when a full body image is given so that the animation looks livelier.

Allow multiple image and audio upload and able to choose one for each type of file before processing

When testing the application, it is found that the user is quite frustrated as users were required to access the first step again to upload a new image and audio. A database that stores all the image and audio that is uploaded by the user (and also able to remove as well) and allows the user to pick one of the video and image for processing shows a more user-friendly interface that saves users a lot of time.

Allow other image and video file extensions to be upload

Our application currently allows 'mp3' extension for audio and 'png' for image to reduce error. However in some circumstances, users might have audio and images that have different but commonly used file extensions. Most users might not know how to convert to the right file extension. Hence, it is better that we need to cater it as well.

6. Limitation for Testing Process

Certain tests were not conducted due to resource constraints:

Load Testing: The system was not tested under heavy traffic to measure performance at scale. This could impact the system's responsiveness under multiple simultaneous requests.

Cross-Browser Compatibility Testing: Testing was limited to Chrome, leaving out other browsers like Safari and Edge. Future tests should ensure broader compatibility. These tests could have been conducted by simulating a high number of users or by setting up automated browser testing across various environments.

Local Runtime: Since we have a poor performance local machine, it is not recommended to test as it will damage the hardware while extensively running heavy processes such as rendering the video. Running locally might be a better choice for users that do not know how to deploy in the cloud, but it needs to meet the hardware requirement to avoid hardware issues.

7. Conclusion

The testing conducted for the singing video generation system has validated its functionality, integration, and usability. The system works as intended, producing videos with synchronised lip movements based on input images and audio files. Usability testing confirmed that users can interact with the system effectively, though minor interface improvements are recommended. Further optimization of video quality and additional testing in real-world scenarios, such as load and compatibility testing, would enhance the robustness and user experience of the system.