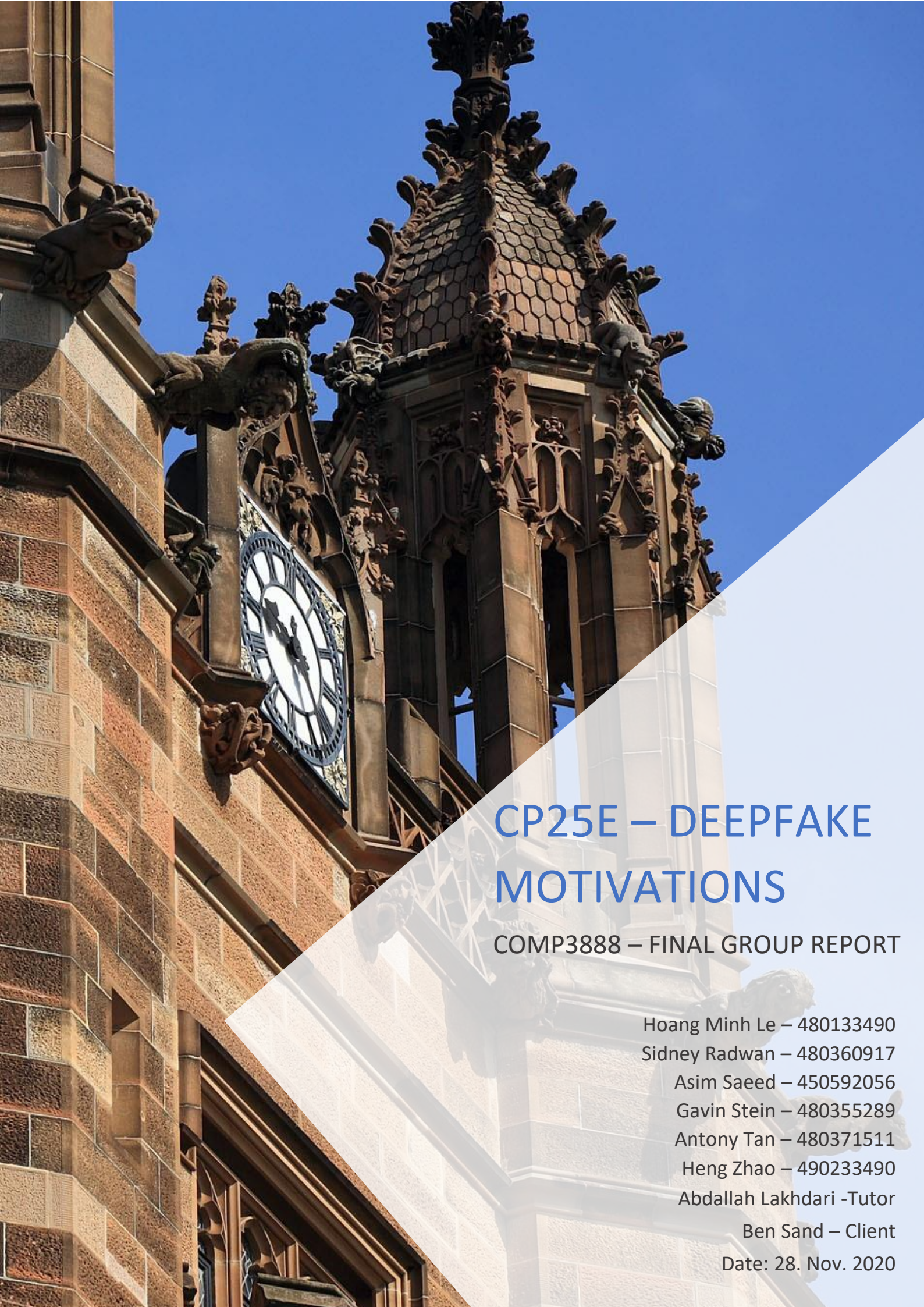


CP25E – DEEPFAKE MOTIVATIONS

COMP3888 – FINAL GROUP REPORT

Hoang Minh Le – 480133490
Sidney Radwan – 480360917
Asim Saeed – 450592056
Gavin Stein – 480355289
Antony Tan – 480371511
Heng Zhao – 490233490
Abdallah Lakhdari -Tutor
Ben Sand – Client

Date: 28. Nov. 2020

THE UNIVERSITY OF SYDNEY

School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT

Unit of Study: COMP3888 Capstone Project

Assignment name: Final Project Report (Group)

Tutorial time: Tuesday 15.00 - 17.00 **Tutor name:** Abdallah Lakhdari

DECLARATION
We the undersigned declare that we have read and understood the *University of Sydney Student Plagiarism: Coursework Policy and Procedure*, and except where specifically acknowledged, the work contained in this assignment/project is our own work, and has not been copied from other sources or been previously submitted for award or assessment.

We understand that understand that failure to comply with the *Student Plagiarism: Coursework Policy and Procedure* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

We realise that we may be asked to identify those portions of the work contributed by each of us and required to demonstrate our individual knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Project team members				
Student name	**Student ID**	**Participated**	**Agree to share**	**Signature**
1. Antony Tan	480371511	Yes / No	Yes / No	
2. Sidney Radwan	480360917	Yes / No	Yes / No	
3. Heng Zhao	490233490	Yes / No	Yes / No	
4. Gavin Arthur Stein	480355289	Yes / No	Yes / No	
5. Asim Saeed	450592056	Yes / No	Yes / No	
6. Hoang Minh Le	480133780	Yes / No	Yes / No	
7.		Yes / No	Yes / No	
8.		Yes / No	Yes / No	
9.		Yes / No	Yes / No	
10.		Yes / No	Yes / No	

Level 2, SIT Building, J12
The University of Sydney
NSW 2006 Australia

T +61 2 9351 3423
F +61 2 9351 3838
E sit.info@sydney.edu.au
sydney.edu.au/it

ABN 15 211 513 464
CRICOS 00026A

Table of Contents

Executive Summary

This Capstone project titled Deepfake Motivations, aims to conduct research on existing tools and libraries, from which we will deliver a system given user provided text can produce a Deepfaked video of chosen celebrity reading said text for the purposes of motivating the viewer. The team from the University of Sydney is working with project sponsor Ben Sands from High Growth Ventures to deliver this project, and consists of team members Antony Tan, Asim Saeed, Gavin Stein, Heng Zhao, Minh Hoang Minh Le and Sidney Radwan. This project will be applying the best of what is currently possible with Deepfake and speech synthesis technologies and produce an accessible, well-documented framework for future work and use.

Wiki

If this link fails, the .md files have been uploaded to this OneDrive folder.

Bitbucket repository

If this link fails, repository contents as of 29/11/20 have been uploaded to this OneDrive folder.

Jira project

OneDrive

Introduction

What is Machine Learning?

Machine Learning (ML), defined by Arthur Samuel in 1959, is a data analytics technique that teaches computers to act without explicitly being programmed [Esposito; Bheemaiah; Tse 2017]. Data can be anything from numbers to images if it can be stored digitally. With data, machine learning algorithms can learn information without relying on a predetermined equation as a pattern. ML is already part of our life: YouTube recommendation system, email spam filter and financial risk management all apply ML algorithm to remove burden from us and solve complex problems.

Machine learning as a concept goes originated back in the 1950's when Frank Rosenblatt designed the first artificial neural network for pattern and shape recognition. However, it became a frequent topic during the 1980's and 1990's when a chess computer was created and beat the world chess champion at the time. Since then, we have seen large companies putting resources into ML projects, such as Google's GoogleBrain project or Elon Musk's organisation, OpenAI.

What is Deep Faking?

Deep Faking is a method of artificially making it seem that a person is doing or saying something they didn't really do [Westerlund 2019]. This is achieved by overlaying the likeness of a person, typically a celebrity over another person, whether that be face, head or mouth. Deepfakes are produced using Generative adversarial networks (GAN) which from seeing a large set of facial images of a person, can learn and reproduce images of that person's face different to the images it has seen [Westerlund 2019].

What is motivation and why do we need it?

Defined as the driving forces behind our actions led by will [Lai 2011], Motivation is something that everyone struggles with at some point. It is hard to remain motivated when attempting to complete tasks or reach goals that seem overwhelming or are not enjoyable, and lack of motivation significantly affects productivity in all aspects of life. Some conventional methods, such as goal setting or incentivising, help attack this problem, but often fall short of seeing meaningful success. We ask, is there another, unconventional method that can help tackle this problem?

Problem Statement

People often struggle to stay motivated to accomplish their goals due to the lack of personalisation and the insignificance of motivational messages.

Aims

To deliver a system that given a user provided script will create a customised motivational video of our chosen celebrity of Morgan Freeman speaking the provided script.

Aim 1: Research the libraries, tools and processes required to implement the system pipeline

- Investigate Deep Fake libraries and tools to overlay and imitate the chosen celebrities' visual likeness onto another video
- Investigate Text-To-Speech libraries and tools to convert text into realistic synthesised voice imitating the chosen celebrity
- Investigate libraries and tools to combine the audio and visual components of the project into the final deliverable product

Aim 2: Develop, optimise and automate the system pipeline

- Construct and finalise the system pipeline structure by selecting and outlining the libraries and tools to be used
- Optimise the development process in terms of speed and quality through tweaking of the selected libraries, datasets and tools to produce the final product
- Automate the system pipeline using scripting where possible

Project Motivations and Context

This project aims to create a product that can be used to help users remain motivated. It will do this by exploring Deepfake and voice synthesis technologies to create motivational videos from famous individuals.

Client and Stakeholders

The client and sponsor for this project is Ben Sands from High Growth Ventures. Ben Sands is a sponsor to several Capstone Projects at the University of Sydney. Ben Sands is also the primary stakeholder for this project, followed secondly by the hosting organisation of the University of Sydney.

Scope

By the end of this project, we will deliver a framework to the client that will take text input from a user as a script and output a video of a motivational figure delivering that script. The project scope includes designing, documenting and delivering this framework where quality and realism is paramount, without the need to implement any user interface. For this project we selected Morgan Freeman as the motivational figure.

Resources

This project's resource requirements are mostly access to online collaboration tools, libraries and computational processing power. The client has opened availabilities for requesting additional resources either through lending of computer hardware or reimbursement for project related expenditures.

Risk

Machine learning by nature is very computationally slow and demanding. Given the limited resources as students working from home, it may be necessary to request additional computational hardware. The timeliness of receiving and utilising the requested hardware will affect the ability and speed to produce iterative prototypes and demos, affecting overall final product quality.

System Specifications

The path this project took was largely determined by what the client requested, and was further directed by what we could achieve with existing, available technologies. Throughout the project's lifespan, under the direction of the client, emphasis was given to end-product quality. This emphasis drove many of our decisions and often meant the sacrifice of other attributes of the system such as automation and ease of use.

The client envisioned a product that could use the inherent motivational characteristics of famous celebrities, in combination with personalisation, to produce a video that was uniquely motivating for the user. This would be achieved with Deep Fake and voice synthesis technologies. Our group worked with the client to further refine this vision to the following user stories and functional tasks.

User Stories

1. As a customer of this service, I want to provide text and have a chosen celebrity deliver my message to me in a motivating manner, so that I can be motivated to accomplish my goals.
2. As a user of this system, I want my provided video to be deepfaked and contain the visual likeness of a chosen celebrity, so that I can use this later in the pipeline.
3. As a user of this system, I want my provided text to be voice synthesised and contain the audible likeness of a chosen celebrity, so that I can use this later in the pipeline.
4. As a user of this system, I want my synthesised voice and deepfaked video of a chosen celebrity be combined, so that I can deliver this final video with audio to the customer.
5. As a user of this system, I want the system to be automated as much as possible, so that it is more user-friendly and accessible.

Functional Tasks

1. Given user provided text return voice of Morgan Freeman saying text
2. Given user provided video return deepfaked video of Morgan Freeman
3. Given generated voice and deepfaked video of Morgan Freeman combine into final video
4. Given the final system pipeline create automation scripts with Python to automise the process as much as possible

A system that achieves these functional tasks will satisfy the user stories, and thus the client's requirements. The system we have developed satisfies these user stories (refer to Appendix A). The functional tasks and user stories have been designed to make them intuitive and testable as system components. This will be covered more in later sections.

User Requirements and Acceptance Criteria

From a user perspective, we want to keep the interaction with the system as minimal as possible. This proves difficult when trying to produce the best quality result. Ideally, the user would provide text as a script, and receive a deep faked video with audio based on the script from the system. Production of the final video, which involves synthesis of the voice and video, and then combination of the two, would be fully automated. However, limitations of deep fake tools dictated that the complete automation of this system was not possible. With this system, the deep fake of Morgan Freeman is done externally through the deep fake tool and these are accessed to combine with the voice synthesis.

Acceptance criteria defined with the client for assessing completion and success of the project include:

- *Realism*: The video should be natural, realistic and convincing
- *Cost*: Ongoing costs of the system should be minimal
- *Maintenance*: Manual intervention required to administer the framework/system should be minimal
- *Speed*: The system should be able to produce results quickly
- *Reliability*: The system should reliably produce quality results
- *Accessibility*: Access and use of the system should simplified and not be restricted for technical or other reasons

The project saw little changes in user requirements between the initial and final deployments.

Technical Constraints

GANs (Generative Adversarial Networks) are the most crucial technology applied to both audio and video synthesis sections of the project. A large number of high-quality samples for training is the primary requirement for GANs to optimize both the generator and discriminator and produce realistic results. Because of this, project work since the initial deployment involved increasing our dataset and also improving the individual quality of each sample within that dataset. This work has been recorded by Jira issues (refer to Appendix B). This naturally means that rigorous amounts of processing power is required which also results in long training times. This was a significant technical limitation faced in the first half of our project however the processing constraints of GANs were minimized through the use of the server supplied to our team by the client in week 6.

Our project did not demand any exceptional security, performance or compatibility requirements. While the deployment of the project was initially planned across multiple platforms, which would have required platform-specific testing, deployment requirements were refined to be for Linux operating systems only and for hardware of similar specifications to a server machine supplied by the client.

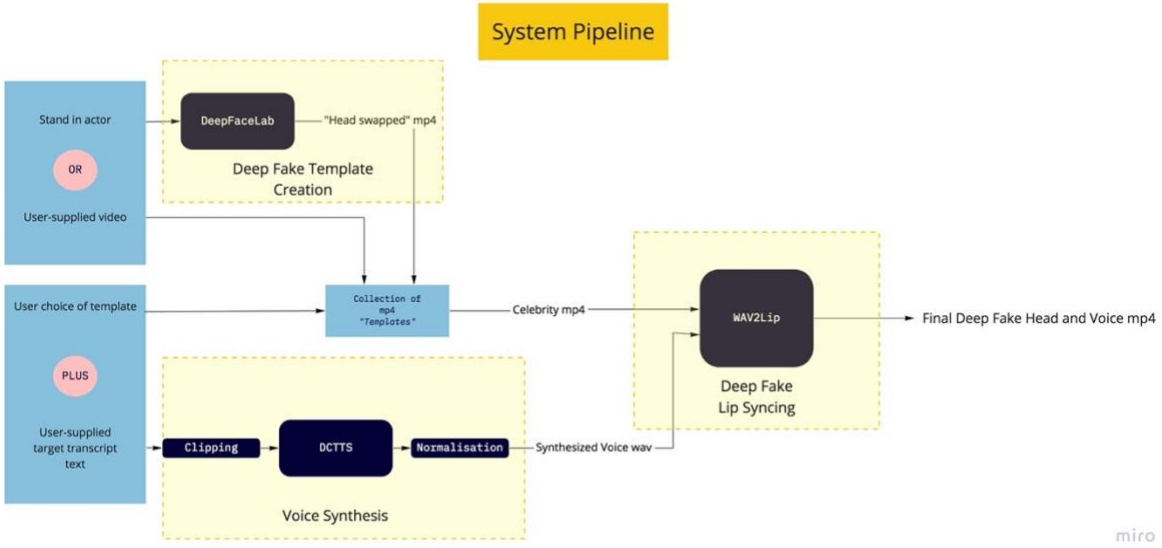
The client had implicitly accepted these technical constraints through acknowledgement of our limited hardware resources and through the setting of expectations for realism to deepfake demos achieved by others in the industry.

Other System Considerations

Project work was often pushing the boundaries of the scope due to the client's constant requests for additional features. Our team strived to satisfy the client to the best of our ability, however at times we needed to refer back to the scope so as not to derail the project. Upon client request, we researched the possibility of using CGI stand-ins for the Deep Fake destination videos. While in theory it was possible, the time, money and effort required to develop a proof of concept, let alone a deployable product with this technology meant that it was not feasible for this project. For that reason, which was also communicated to the client, we kept with the initial system specifications and focused on achieving those. The client made further requests which we handled such as Emotional Tagging and normalising our speech synthesis. We investigated Emotional Tagging but found it wasn't ideal for our project due to the technology's current state (early R&D) and our project's priorities. Further explanation can be found in the research section. For normalising our speech synthesis, we found it feasible and applicable, so we implemented the feature and presented it to the client after the following iteration sprint.

System Design and Implementation

System Architecture



UML Diagram of Final System Pipeline (Refer to Appendix C for full size image)

Deep Fake Template Creation

Our system pipeline is structured to have a library of template videos which contain variation in qualities like facial emotions, backgrounds and body movements. These videos are later combined with voice synthesised audio. Deep Face Lab is a library which is our Deepfake tool of choice. Deep Face Lab takes a provided destination video and outputs a deepfaked mp4 video with the visual likeness of the chosen celebrity replacing the likeness of the provided actor. This satisfies the second user story. These templates are selected by some user of the system, who can either be defined as the customer or the provider of this service. Additionally, there is the option for the user to provide their own video, bypassing the pipeline path of creating a deepfaked template video.

Voice Synthesis

DCTTS is the chosen library for text-to-speech voice synthesis in our system, also known as speech inference. This will take text as input and output a wav audio file of the chosen celebrity's voice reading the text. This satisfies the third user story. Before text is converted to audio, a part of the pipeline is to slice the text at commas and full stops, individually convert slices to audio then recombine with timing adjustments after speech inference. This gives the sense of natural pauses in speech. After inference the pipeline normalises the audio to improve consistency of the audio's volume. Both these pre- and post-processing steps assist in fulfilling the first user story. Later in the pipeline, this audio output will be combined with the video component.

Deep Fake Lip Syncing

WAV2Lip is the chosen library for combining audio and video for our system pipeline. As input WAV2Lip takes the deepfaked mp4 video containing the visual likeness of the chosen celebrity and the voice synthesised wav audio file containing the audible likeness of the chosen celebrity reading the provided script. The output of this component is an mp4 video with the input mp4 video and wav audio lip synchronised and combined. The library makes the mouth of the celebrity in the mp4 video appear to be mouthing the words contained within the wav audio file. The output of this component is the final mp4 video which will be delivered to the customer. This satisfies the fourth user story.

Pipeline Automation

Our pipeline is automated with Python scripting, so the user of our system only has to run a single command:

```
$ python3 dfmotiv.py -t “Hello World, this is Morgan Freeman”
```

This will output an mp4 video of the chosen celebrity saying what the user requested. After running this command, sub-processes are launched to produce voice synthesis and then combine this with a template video using deepfake lip syncing. On main process return, the final combined mp4 video is available in an output directory. This satisfies the fifth user story. Deepfake template video creation was not fully automated and justification for why can be found in the research section.

System Tools and Components

Deep Face Lab (DFL)

Deep Face Lab is a leading Deep Fake library available on GitHub. The entire process of deepfaking with Deep Face Lab is achieved with a sequence of bat files which run pre-written scripts and launch internal user-friendly tools for achieving the steps to complete the deepfake. Deep Face Lab requires both a destination video and a source video for training. The source video is provided by the system user and contains the chosen celebrity portraying a wide range of facial expressions for a wide range of camera angles. This is then to be pre-processed alongside a destination video chosen by the system user. After pre-processing, for the method of head replacements it is necessary to conduct X-SEG mask training to allow the later model training to easier identify the head shapes of the actors. Next training the model to learn the faces of the actors is launched, which typically takes a large amount of time to train. Finally, merging is conducted, and a final mp4 output video is produced. This process is repeated and taken advantage of to create the template mp4 videos of the chosen celebrity which will each have differing destination videos (Refer to Appendix D for UML diagram).

Deep Convolutional Text to Speech (DCTTS)

This Python package was published in 2018 on GitHub and uses *TensorFlow 1.3*, *scipy*, librosa and *tqdm* python modules to build a neural network and conduct voice synthesis from text. These modules contain built-in AI and machine learning algorithms. For training data, users need to create a folder for the dataset, containing audio that is in wav file format, and include a csv file containing the file name of each audio clip with their corresponding text. The path of the dataset and is then defined in the *hyperparam.py* file (find in *dfmotiv* repository *libraries/dctts* folder) with the option of setting up the parameters for model training, such as batch size, learning rate or number of iterations. Running the training, the program takes the script with the audio files and convert them into matrices to then train the neural network. A model is then created and applied to text input to create synthesised speech as an outputted wav file (Refer to Appendix D for UML diagram). More detailed explanation is described in the research section.

Wav2Lip

Wav2Lip is a package and model that morphs lip movements of talking faces in a video to match that of a newly supplied audio. It uses an accurate pre-trained discriminator as a "lip sync expert" forcing the generator in its GAN setup to produce accurate lip shapes. It also features a visual quality discriminator to minimize noise and artifacts produced by the generator. This pretrained lip sync expert works quickly on arbitrary faces, so is a suitable option for fast realistic results. Taking a destination mp4 video and wav audio file as inputs, it outputs a video with the synthesised lip movements synchronised to the wav audio file. Ensuring the audio and video files match in length is automatically handled by the tool (Refer to Appendix D for UML diagram).

Implementation

Overview

Our automated system is a command-line program which takes in various flags and inputs for the user to customize the execution. The options allow specifying input and output data, such as the text to be voiced (inputted via command line string for on-the-fly use or a .txt file as a no-flags default), input template videos (prepped via DFL or other user-supplied video clips) Users can also soutput paths / extensions. With no requirements for a Graphical UI, the client afforded us flexibility in how text could be inputted into the system (initially suggestions included via email or a text file etc). Thus, the python command-line module not only covered this requirement, but left the system open for extension (future projects can build upon our foundation).

There are three main components to the system. The first is our deployment repo named “dfmotiv” containing the entry point python module of the same name, which is to be invoked via python on the command line and passed along the option flag arguments. The second pieces are the two repos of the aforementioned tools, DC_TTS and Wav2Lip. It should also be noted that our deployment repo, dfmotiv, also contains files that were modified in DC_TTS and Wav2Lip. Rather than including the whole repos, we included only these files to try limit the code to that which was written by our team. The third piece is two python virtual environments, which allow us to isolate the required python dependencies of DC_TTS and Wav2lip. Once setup, these virtual environments are a root directory with self-contained installations of python dependencies and a “symlink" to an existing python interpreter executable on the host Linux OS. Together, these three components total five root directories:

- dfmotiv
- dc_tts
- wav2Lip
- dc_tts-env
- wav2lip-env

The dfmotiv module is made aware of the locations of the other 4 via a contained “paths.yml file”, pointing to each directory. Thus, the system doesn't impose strict locations for these directories. Although installation is not automated, the setup steps were meticulously detailed in the dfmotiv repo's network of "README" markdown files (refer to Appendix X). These are rendered for easy readability in the bitbucket repo or a similar previewer.

The deepfake models were pretrained, zipped and hosted in OneDrive with download links and placement instructions in the dfmotiv README. The system execution would simply inference these models and create the desired motivational media, a much shorter turnaround time than the model training process.

Automated Execution Overview

Execution can be done in the host's default environment, invoked with just using python3 to call the dfmotiv.py module, giving other developers (or users) freedom to implement the system for their own use cases. The dfmotiv package then spawns child subprocesses and activates the virtual environments. On execution any modified versions of modules are automatically copied and overwritten into DC_TTS and Wav2Lip, and their respective run.py module is called.

The DC_TTS subprocess is created first, pre-processing the speech text, creating the vocal pieces, and combining them with appropriate pauses (varied based on commas and full stop characters in the speech text) into a normalized final voice wav file. While this is happening the main dfmotiv process is blocked until it receives a return code from the child process; also forwarding the nested standard output and standard error prints. The final output wav from this process is then ready to be used by the next subprocess, or an error is raised if it is not found. The next subprocess, running Wav2Lip, executes very similarly except the output is the final deepfaked mp4.

OS Level Dependencies

Linux Ubuntu was chosen as the most appropriate development and deployment environment, especially given the terminal app, package management tools and wide online community support to aid the setup of all the dependencies we needed. During the mid-semester break, the client provided a server machine featuring multiple CUDA-enabled high-end GPU's meeting the VRAM requirements for the deep learning models. The machine was placed in an air-conditioned server room and the team was given remote access via a remote desktop application TeamViewer.

For managing system level dependencies, especially ones that required conflicting versions between Wav2Lip, DC_TTS, we either used management tools (like "pyenv" to manage multiple python installations, each linked in the appropriate virtual environments) or simply had multiple versions installed like with CUDA. We manipulated what each subprocess environment needed to see via environment variables.

Code Design and Style

The following outline shows the structure of our deployment repo:

Note: This outline does not include the virtual environments and full repos of DC_TTS and Wav2Lip which would be setup outside this repo, but the .md files included heavily detail the final operational setup.

./dfmotiv

- libraries
  - dctts
    - run.py & various modified .py files
    - requirements.txt for pip installs in virtual env
    - dctts.md
    - unit_test.py
  - wav2lip
    - requirements.txt for pip installs in virtual env
    - wav2lip.md
  - dfl
    - dfl.md
- tensorflowtts
  - various inference.py files
- utils
  - google_SST
    - longSST
    - shortSST
  - transcribe
    - generate_transcript.py
- README.md
- venv.md
- **dfmotiv.py**
- paths.yml
- .gitignore

Although it was challenging to keep all code following strict conventions such as PEP-8 rules and Google style Python doc strings consistent across all modules from various authors, the code written by the team followed these conventions excluding that within integrated repos from GitHub. To enforce quality control, uniformity and readability of the README's was our highest priority, due to the sheer amount of information contained within for a reader to follow.

Moreover, with our initial repo created during the start of semester oversized (crossing the limits of the allowed 2GB) due to unintentional git commits of various data files (generated

or otherwise), the team needed to start afresh with a new repo by mid-semester with strict rules in place enforced by the lead programmer:

- *Ensure no data, model, video, image, audio files are committed to this repo (use .gitignore)*
- *Don't commit files that can/will be auto generated*
- *If you need to include a whole library/package, fine, but please use gitignore diligently*

These rules were also featured on the main README of the repo, along with other best practices and tips from the team's git expert.

Other Specifications

Pre-Processing Tools

While it wasn't a component in the final system design, data pre-processing was an essential foundation to this whole project, especially for speech synthesis models. For this project to succeed, it was necessary to create a large dataset of speech samples of Morgan Freeman. Ranging from three to ten seconds, they had to not only be clear, but also free of background noise, music and other speakers. This proved more difficult to create than originally thought. Many of our samples were extracted from interviews with the celebrity, a small few were taken from movie scenes, and some were sourced from an audiobook. By the end of this project, we had collected roughly 50 minutes of speech. To trim and edit these samples we used Audacity, a free audio manipulation tool.

The training of speech synthesis models also required the samples to be transcribed. Doing this for the hundreds of samples was too tedious to do manually. Instead, we used the speech-to-text tool GoogleSTT, which is a transcription tool, and then manually quality controlled its output. For properly using GoogleSTT, we made shortSTT.py script for transcribing the audio files less than 5 mins, and longSST.py script for transcribing the audio files longer than 5 mins. Based on manually testing with different categories of audio files, the accuracy of GoogleSTT was found to be great except that some of the names could not be correctly recognised. Manually correcting lines after transcribing is only a tiny portion of the total workload. Overall GoogleSTT saves much time and helped our team collect a large amount of audio samples feasibly.

TensorflowTTS

Throughout the project, two speech synthesis techniques – DC_TTS and TensorflowTTS – were being developed. It was the intention from the beginning that the technique that produced the best result would be the one we used in the final deployment. DC_TTS is what was included in the final deployment, however that is not to say TensorflowTTS is a lost cause. This technique has produced promising results in other implementations, specifically with larger datasets, however within the timeframe of our project we were unable to replicate such results. With further research and development, we believe that this technique can supersede DC_TTS to be the best option for future use of this project. Thus we have included TensorflowTTS into our project with a detailed wiki page (refer to Appendix E).

System Evaluation

Quality of Work

To raise and uphold our quality of work, we aimed to meet certain quality constraints for software design. With our specific project and client requirements, security was not a primary concern. Both the libraries and tools we used are reputable, and the service has minimal customer involvement. Performance and reliability were most important. The speed of producing results is closely related to the specifications of hardware, input, settings for training, and can vary even days of waiting time. The design of the pipeline also is intended to reliably deliver consistent results as training models can crash and produce undesired results, which can waste lots of valuable time. Our pipeline is supposed to simplify the process of production to be accessible to general users, by extensive documentation, packaging and automation of the process.

Moreover, for the assessment of our success, we employed a test-driven developmental approach. Any piece of work; qualitative such as deepfaking or quantitative such as coding was subject to a pre-defined success criterion to ensure that each sprint of work had a clear measure of if it was successfully completed or needed to continue into another sprint.

Acceptance Tests

The acceptance tests we have defined below measure the degree of success of our product in meeting the acceptance criteria we defined with the client. Defining these early in the project process aligned with our test-driven developmental approach:

Testing Cost

The cost to maintain and operate the system should not exceed the client's budget or expectations. The success of achieving this criterion is deterministically found by consulting regularly with the client about proposed costs to running the system.

Testing Speed

The client is able to continue the use of this system with sufficient hardware resources. However, it has been mentioned that a system that can deliver a result within two days will satisfy the client, given the quality also being satisfactory.

Testing Reliability

The system should be tested if it can run reliably on Linux. It must run without failing at least five times without failing on the client's intended operating system. This acceptance criteria was refined due to changing client requirements as the system no longer needed to be operational on Windows 10 and MacOS.

Testing Accessibility

For users of the system pipeline we will conduct usability testing with a survey response (Refer to Appendix F). This will allow us to iterate the system framework's delivery in terms of documentation and automation to better improve the system's ease of use.

Testing Realism

We will provide a mean opinion score survey to give to our testers regarding the quality of our video to grant insights on the improvement of our product. We will also measure the speech accuracy of our voice synthesis with benchmark testing.

Testing Plan

Testing is an important part of our project to ensure we achieve a satisfying result and satisfy our acceptance criteria. Five types of testing have been done: unit testing, regression testing, user-experience testing, performance testing and benchmark testing. Other types of testing were not conducted due to the qualitative nature of this project.

Unit Testing

The purpose of unit testing is to assess the functionality of each stage of automation in our system and uphold a minimum standard for input and output. Using the Python *unittest* module, unit testing was written into the *test_unit.py* file and its aim was to cover different scenarios. These were defined before development began to keep with test-driven development:

Abnormal Cases

- User provides unusual characters into the program (e.g. +, £, #)
- User tries to run the program without giving motivational text (< 1 word)
- User tries to give a too long motivational text (> 50 words)
- Template video for lip syncing is not provided

Boundary Cases

We allowed a maximum of 50 and minimum of 1 words of motivational text. These were tested. Setting this limit ensured that the motivation is not too long to be overexaggerating or too short to be less effective.

Normal Cases

- Checking on a normal text input
- Testing whether correct files are created
- Testing whether files are created in the right path
- Testing whether files are created in the right format

Boundary Cases Exceptions

Single word text input is allowed. However, as this may not satisfy the realism criteria it is provided an exception to be a successful delivery.

User provided destination videos as input with minimal destination face visible or low-quality video is allowed as input. This can lead to the final product not satisfying the realism criteria and it is provided an exception to be a successful delivery.

Abnormal Cases Exceptions

Undiscernible or poorly formatted text input can lead to un-realistic voice synthesis results not meeting the realism criteria, however due to the system's allowance of this type of input an exception is provided for this to be a successful delivery.

The logs of testing are then tracked in an Excel spreadsheet, describing the scenario, type of testing, result and reason of failure. Overall, we covered enough scenarios to be satisfied with unit testing and have generated a full unit testing suite (refer to Appendix G).

Regression Testing

The aim of this type of testing is to verify the functionality of each step in the pipeline. We have tested whether our program creates expected results from given inputs at each pipeline stage (Refer to Appendix G). That means we check to see if the program creates the correct audio synthesis and deepfake video, as well as see if the end result is combined correctly into a single video file. This testing was necessary as during sprints, modification was needed and we had to have a working pipeline regardless of any changes that occurred. The following steps were tested each time a change was introduced:

- Check if we can input text and the correct wav file is created
- Check if we can input a video file and the correct deepfaked video is created
- Input a wav file with a video file and see if the output is a correctly combined video file

This was conducted through unit testing although it is differentiated due to its different testing purpose.

User-Experience Testing

Due to the nature of the project it is not possible to judge a deepfaked video with test cases. Since assessing the quality, motivational nature, and realism of our deepfaked videos was critical to ensure we satisfied our first user story and our acceptance criteria of realism, user-experience testing was elected to be conducted. To measure improvements in quality and see if our end result satisfied our expectations we derived a survey with questions having scores ranging from 1 to 10 and targeted toward assessing the quality of voice synthesis, video, lip sync and motivation (Refer to Appendix H). These questions would be given to a third-party group. By measuring mean opinion scores of different aspects of our videos we can gage their success as motivational agents. By observing the scores received by our products at two stages of the project, it is possible to quantify improvements in quality.

Benchmark Testing

This type of testing was conducted to measure the degree of performance in our product meeting set benchmarks. The result is compared with a quantitative benchmark that and this determines success. This includes the speed of the system, the accuracy of pronunciation and system reliability.

Testing Results

Unit Testing and Regression Testing

The unit and regression testing logs including scenario details, function names and error messages can be found in the Excel spreadsheet (refer to Appendix I). Overall, 6 errors were found and fixed during the testing phase.

User-Experience Testing - Product Quality

We created five deepfaked videos of Morgan Freeman reading five motivational quotes with our week 6 product and used this as our control (refer to Appendix J). Then we compared scores with collected responses on our final product, having the celebrity reading the same five quotes as the control videos (refer to Appendix J). Furthermore, we surveyed voice synthesis on the same quotes combined with an original Morgan Freeman video extracted from The Shawshank Redemption (refer to Appendix J). This was added to see the results achieved when bypassing deepfake template video creation and removing one more element of artificiality.

Through a third-party group of university students, we totalled 69 survey responses and we derived a summary of results (Refer to Appendix K). For all wider aspects, being voice synthesis, video quality, lip sync and motivational quality, our new videos received a higher mean score. Our final product and final product without deepfaked template video achieved significantly higher median scores than our week 6 product. For our final product, lip syncing had the highest maximum score as people rated it 9 out of 10. Although, many people did not feel motivated by our videos as that aspect received the lowest maximum, minimum and median scores out of 10 compared to all other aspects, we did see a 2-point increase in mean and median scores for motivational quality since our week 6 product. The reason for low motivational scores could be due voice synthesis being too artificial, with some survey responders giving feedback that the robotic voice was too distracting to be motivational.

Despite that, both the voice synthesis and video quality improved substantially, with the deepfaked head replacement of Morgan Freeman having the highest satisfactory rate. Our final product's deepfake of Morgan Freeman even received higher scores than the video from The Shawshank Redemption, potentially due to our less stationary stand-in actor.

Benchmark Testing - Speech Accuracy

This benchmark testing was undertaken to check the output accuracy of our speech synthesis, specifically the pronunciation of our text-to-speech (TTS) model. The purpose of this test was to measure realism, by checking how well our system can pronounce the 44 English phenomes. The result of which would indicate whether we were lacking in data for specific phenomes, as the model can only learn those present in its training dataset. This test was done by generating random sentences and feeding it into the model, where the results would be manually reviewed (refer to Appendix L).

Our speech model by the end of the week 7 pronounced 51.35% of words correctly. The sentences suffered from extreme distortions; words were either an incomprehensible jumble of sounds rather than speech, or the words that were pronounced correctly sounded as if it was stitched together from multiple sound bites. From these tests, we were able to identify insufficient training time and a weak database as the main causes. To rectify this, our database was expanded from 15 minutes to 40 minutes, and several weeks were spent experimenting with hyperparameters and training our speech model. Furthermore, the addition of audiobook clips of Morgan Freeman provided consistent and clean audio reflected in improved speech synthesis accuracy.

By the end, our speech model could correctly pronounce 94.59% of all words in our test sentences. The mispronounced words had uncommon letters or were homographs. For example, our model struggles to pronounce “qu” and pronounces the “ch” in “charisma” as “tʃ”, which is the same ‘ch’ in cheese. Possible ways to improve our speech model further is addressed in “Testing Conclusions”.

Benchmark Testing - Speed Testing

The performance and speed of our speech model was tested to evaluate whether it can consistently inference speech at a consistently fast rate and satisfy the acceptance criteria of speed.

As seen in the results (refer to Appendix M), our model can produce results at a consistently fast speed, with negligible variance, and far succeeds the acceptance criteria of producing results faster than two days. The number of characters in the user’s input has a minimal effect in how long the inference process will take. Rather, the length of the inference process is bound by the number of punctuations in the user's input which never grows large enough to significantly become of concern. The performance results achieved far succeed the requirements of speed defined in our acceptance criteria, with the slowest result on the largest possible input taking less than one minute to produce, much less than the benchmark of two days.

Benchmark Testing – Reliability Testing

Through the course of our product and demo development. We ran our intermediate and final systems on two separate Linux machines, far more than five times on each, without any error. Thus, we were able to claim our system runs reliably and passes our benchmark set to measure success in meeting the acceptance criteria of reliability.

Testing Conclusions

The quality of synthesised voice depends heavily on the quality of voice dataset, as a noisy dataset will result in a voice affected by static, and a dataset lacking in phenomes leads to inaccurate pronunciation. We have collected high quality audiobook samples and cleaned voice lines of the chosen celebrity noting tangible improvements, but in general noting high-quality voice samples are scarce. An alternative we prepared and can recommend, but never used for this project due to its cost, is to use voice actors to impersonate the chosen celebrity and read out a script to cover all 44 of the English phenomes.

To also help our DC_TTS model to learn to pronounce the same set of letters differently. A potential solution is to convert our database's transcription and our user's input into ARPABET notation and have the model train on that data. However, the latter step cannot be automated unless our model can access the ARPABET notation of every word in the English dictionary, and if it could, it would likely be a large processing sinkhole. The alternative is to use a speech model that can learn to pronounce homographs correctly such as Tacotron2.

In summary of our findings, the most important points of our project have been tested, fixed and succeed in meeting our defined acceptance criteria and benchmarks. All operational scenarios were tested, and overall performance, such as speed and accuracy, was observed and recorded. Automation of the system works reliably, is relatively fast, and quality of our product is satisfiable.

Although the team did a lot of testing, one limitation of our testing was of the utilised built-in packages. Packages like DC_TTS and Deep Face Lab have complex methods built upon Python modules like Librosa or Tensorflow and so output of the neural network models from voice synthesis or deepfake inferencing can change drastically according to input data. Since these built-in packages are widely used and tested, and we did not make any major changes, the team decided not to directly test the operation of the package models.

Importantly, we found that we satisfied all our acceptance criteria with our final system and product. For cost, our system was made entirely with free components, data and tools so we succeeded in that test. Speed also succeeded being much lower than our defined benchmark with the client. Reliability also was no issue with our system working without fail at least 15 times producing demos for testing. Accessibility unfortunately was our least tested criteria. This was due to a client provided team, which was supposed to deploy our system and provide feedback, quoting that they would deploy our system but delaying the action and eventually ceasing contact with us (refer to Appendix W). This resulted in our team relying instead on client feedback on our first deployment to improve accessibility as well as having multiple team members test installing the system from scratch using our provided instructions. Realism, due to our product's qualitative nature, was tested in two ways; mean opinion scores and speech accuracy. Both tests indicated large improvements in realism across our project duration and satisfied expectations for this criterion.

Project Research

Deepfaking

This whole project revolves around deepfaking. Therefore, it was essential that we research and learn what this technology is capable of so we can push the boundaries ourselves.

From a client suggestion early on in the project, we researched into **Deep Face Lab**. Deep Face Lab is a leading deepfake tool hosted on GitHub (Refer to Appendix N) that provides accessible and easily producible results. It has been utilised to produce our deepfake demos of Morgan Freeman which were presented as milestones to the client (Refer to Appendix O). The entire process of producing deepfakes with Deep Face Lab is simplified into a sequence of bat files which makes the process extremely accessible to users, with very little domain knowledge about AI and machine learning being needed. The results of utilising Deep Face Lab have been satisfactory, and we have achieved our final desired fidelity using the tool.

However, there is a large amount of tweaking of settings required to find ideal results for each user and the large amount of training time makes iterative prototyping very time consuming. Furthermore, substantial hardware requirements to run training, lead to either an inability to conduct training for not meeting the minimum requirements, or a need to lower the quality of result. There was initially thought to be a possibility that training a model well on one high-variation destination video could allow the model to be re-merged with similar appearing destination videos without the need for re-training. This was found to be untrue as overfitting occurs during training. Thus, no global, general model is achievable which can output the quality we require. Lastly there is a need for manual intervention to assist the AI in detecting head shapes and cleaning the dataset after pre-processing, as the AI occasionally makes mistakes which if undetected will harm the final result. The user must trace the head shape of the celebrity and ensure no images that contain corrupted face-sets are in the dataset to be used for training. Also, initiating training launches a process that runs for typically 8-10hrs minimum without need for intervention. It was deemed appropriate that at the end of training, the user of the system use this point as a checkpoint to see if the process should continue. This means the process of deepfaking with Deep Face Lab cannot and should not be completely automated.

To counter the detriment of manual intervention in using this tool, extensive documentation and instructions for pre-processing and training, written by the team based on our research, eliminates guesswork for a user of the system. The concern of substantial computer processing requirements was mitigated when the client provided the solution of additional computational hardware. Considering the pros and cons of the tool, Deep Face Lab was found to be ideal for the deepfaking component of our project.

During our deepfake research, we also discovered **Wav2Lip,** which is another deepfake tool hosted on GitHub (refer to Appendix N), that morphs the lips of a face in a destination video to match new audio. The authors and contributors have provided pre trained models for face detection as well as a “lip sync expert” checkpoint. It is quick to create realistic morphs and has allowed us to create a variety of tests quickly, more so than any other component (Refer to Appendix O). The visual quality can be a hit or miss. The lip shapes sometimes can look too

exaggerated. Also morphing a talking face to become silent, to match pauses in audio, doesn't perfectly shut/restrict the lip movements. Used in tandem with Deep Face Lab it is possible to generate a custom video of a chosen celebrity with lip movements synchronised to a provided voice .wav file. The realistic quality of WAV2Lip is greatest when actor is facing nearly front-on and level-headed to the camera. Inferencing lip movements based on a provided audio clip takes under a couple minutes. Furthermore, the tool will automatically trim or loop the video component if the audio is respectively shorter or longer than the video. Thus, we have selected Wav2Lip to be our deepfake lip syncing tool for this system based on its quick, high quality results and ease of implementation into our pipeline.

Voice Synthesis

Voice To Text

Voice to text was initially considered in the early stages of the project before requirements changed to what they are now. Intentions were to convert a user's voice to text, then into a celebrity's voice. We found two available options; **Google's WaveNet** and **Python's Speech_Recognition** (refer to Appendix N), then conducted research on them. We found that the speed and accuracy was satisfactory, even for noisy samples, and they could be easily automated into our project pipeline. However, it was determined that it would not be a viable solution for this project due to client requirements. Nonetheless, Python's Speech_Recognition was still used when building our speech dataset to transcribe samples.

Text To Voice

To create a low-resource speech synthesis, Hideyuki Tachibana suggested a deep convolutional text-to-speech synthesis technique (**DC_TTS**), found on GitHub (refer to Appendix N) that can result in an acceptable voice synthesis without a powerful computer [Tachibana, 2017]. The idea behind it is to use two training methods: Text2Mel where a text input is converted into a mel spectrogram; and a Super-resolution Network (SSRN) that reduced the robotic sound of the voice synthesis. The basic idea behind Text2Mel is to use a text encoder, an audio encoder and an audio decoder. The encoders convert text into encoded matrices and audio data into mel spectrogram. A mel spectrogram is a matrix that is generated from an audio data using Fourier Transform and keeps the frequencies of the audio. From these two matrices an attention matrix or plot is created that maps the characters in the matrix to a sound from the mel spectrogram. This attention matrix then decoded back to a mel spectrogram to compare it with the training data and allow the neural network to learn from the error.

After that, SSRN is used for upsampling the frequencies from the mel spectrogram to get a final spectrogram. This is necessary since a low frequency sound resembles robotic sound.

The actual implementation of this paper was found on GitHub (refer to Appendix N) and used in our final project. An extra modification, however, was made where neural network training progress was saved. That meant that every 1000 iteration of the training progress was saved enabling us to pause and resume the long training process. The reason of doing this was not only necessary because of long training time but due to the fact that training time is not

correlated to quality of voice synthesis. A long training time does not lead to significant voice improvement and the attention plot was crucial to follow (refer to Appendix P). Tracking the attention plot, the aim was to see a diagonal line meaning every character is matched at the right sound at the right time where y-axis represent the characters and x-axis represent the time. During the project, we experienced that long training time sometimes resulted in worse attention plot (refer to Appendix P).

This library also had a noticeable problem where the created audio ended with a long noise. The source of this problem was researched and we found that the input text is fed into the model in matrix format where columns stored characters in encoded format and rows stored each sentences. Therefore, the dimension of the matrix is dependent on the number of sentences and on the number of character the longest sentence has. Obviously there are sentences that are shorter and so the DCTTS library fills out the missing character with 0 representing silence values to fit the matrix. However, this is not perfect solution and so noise appears in the audio file. To solve that we decided to cut the end of the audio file based on the number of character that was inputted into the model multiplied by 100 milliseconds (refer to Appendix V). This resulted a better audio quality.

Tacotron2 was another TTS model that was explored for our project (refer to Appendix N). Tacotron2 is a neural network architecture for speech synthesis from text. It functions based on a combination of a convolution neural network (CNN) and recurrent neural network (RNN). The RNN is a feature prediction network that predicts a sequence of mel spectrogram frames from a character sequence, and the CNN generates waveform samples conditioned on these mel spectrogram frames. That is; Tacotron2 uses one neural network to convert text into a spectrogram, and the other to convert it into sound. Like DCTTS, Tacotron2 requires a large dataset and its transcription. However, it also requires a WaveNet model as a vocoder, which is trained purely based on the sounds of the dataset; no transcription is needed.

The main advantage that Tacotron2 has over DCTTS and the reason its development was pursued because of its ability to distinguish between meanings of heteronyms and pronounce them based on usage [Shen et al., 2017]. For example, it can learn to pronounce words like “read” correctly given its context, which is a weakness of our current DCTTS system as explained in “Qualitative Product Testing - Speech Accuracy” found in the “System Evaluation” section. However, the main reason Tacotron2 was not implemented in our final product was due to the time and processing constraints. Training the RNN takes the same amount of time as DCTTS, but training the CNN takes approximately 7 times longer than the RNN, which would require over a week to train. As we were sharing the server machine between other groups and group members, we decided to focus development on DCTTS and TensorflowTTS. Though a model was never finished, evidence of its development can be seen (refer to Appendix Q).

TensorflowTTS is an implementation of multiple different existing text-to-speech architectures based on cutting edge research, including Tacotron2, hosted on GitHub (refer to Appendix N). It combines a text-to mel spectrogram model with a vocoder model to do end-to-end speech synthesis. Initial research and development of this tool used the Tacotron2 architecture for the text-2-mel spectrogram model, however we found that an alternative architecture, **Fastspeech2** (refer to Appendix N), suited our project better. Fastspeech2

provided quicker inference and faster training at a lower computational cost. This architecture paired with the multiband-melgan vocoder, appeared to produce the best results according to our research into other implementations.

The challenge we faced with this approach was the requirement for these models to have a large dataset of speech samples to train on. To tackle this problem, instead of building our dataset to a size of around 20 hours, we attempted to use **transfer learning** and fine-tune a pretrained model. To develop this idea, we relied on an active thread on the repository's GitHub, which focused on fine-tuning models using a small dataset. The link to this thread can be found on the TensorflowTTS wiki (refer to Appendix E). This proposed the use of a pretrained model based on the LJSpeech dataset (roughly 23 hours) and then training on specific layers for another 100k+ iterations. The results we achieved with this technique however were not satisfactory enough to be delivered to the client.

We also attempted to fine-tune with a multi speaker dataset, and use "speaker embeddings" to encode speaker characteristics that could be extracted at the time of inference. While we had moderate success with other speakers in the multi-speaker dataset, inference using Morgan Freeman's speaker encoding proved equally unsuccessful as the previous technique. While we did not achieve the results we wanted using TensorflowTTS, with more time and further research and development we believe this tool can provide fast, accurate speech synthesis better than that of DC_TTS.

On client request, we looked into what available libraries there are for tagging emotion (**emotional tagging**) to our voice synthesis. There were several but one that caught our eyes from GitHub was **dl-for-emo-tts** (refer to Appendix N). The repository explains different trials of emotional speech synthesis and points out the benefit and the disadvantage of each. Overall, it seemed like all these emotional speech syntheses are based on Tacotron2 or DC_TTS models, but implementing it would cost us in quality of our voice synthesis. The produced audio would be less clear, and since our main priority was to create an articulate and clear voice, the team decided to redirect effort into accomplishing a voice synthesis similar to a human voice and eliminate the robotic sound.

CGI Stand In Actors

By mid-semester, the client had really pushed for inputs into the system be text-only, and for procedurally generated CGI animation to drive the deepfake faces. However, not only was most of the team unfamiliar with 3D animation, the timeframe was too short for the scope expansion that would occur from this additional hard constraint. Thus a thorough formal assessment was made by the lead programmer (refer to Appendix Y), concluding that this was out of scope.

Cloud Computing

Given the nature of deepfaking and machine learning, it is necessary for this project to employ more powerful computing resources. For this reason, we researched services that we could have used to train our models. One such solution would be Google Colab. It is a cloud computing service which provides timed cloud machine instance that can be interfaced via an executable notebook. We've explored this with a community provided notebook for the DFL package. Free accounts can provide up to 12 hours of using a high-end GPU Nvidia Tesla k80. Also as mentioned the DFL community has shared a notebook for setting up a DFL workspace and get up and running with training.

We've found this service gives priority to interactive notebook use over longer computations like training. Thus, our trainings end in frequent disconnects at around the 30 minute mark, and usage restrictions are applied in the form of having a CPU only cloud machine instance, removing our much required GPU access. Also, their premium service being U.S only means we can't even pay to mitigate these restrictions. Therefore, we determined that it was not appropriate for our project.

An alternative was Paperspace gradient, which has free accounts running Jupyter notebooks in a timed cloud machine instance. It includes powerful free GPU use; a P5000 during off-peak hours, and unlimited GPU usage. However, it will auto disconnect every 6 hours and it only runs in a pure Jupyter Notebook environment. This means for some changes such as the usage of Google packages and modifying the directory path it is difficult to include in dependencies required in setting up the workspace. Setting up all the packages we have been using for our project takes up much time in the form of experimenting and testing. We paused our research on this as we were granted access to a Client's powerful server machine.

Data Requirements

Visual Data

Visual samples of Morgan Freeman were collected to train our deep fake models. Video footage taken from movies, interviews, and documentaries were taken and transformed through Deep Face Lab. Images of his head were manually traced and extracted from frames from these videos and used as a mask to be deep faked onto our own team members. Furthermore, the video data need to include a variation of head angles and expressions and must be of consistent lighting and resolution as the video Morgan Freeman was being deep faked onto.

Audio Data

Similarly, speech data of Morgan Freeman was extracted from movies, interviews, documentaries, and later audio books. In early project stages we identified the risk of the quality of our sample library hindering our synthesis quality and addressed this by securing audio samples of Morgan Freeman from audio books.

The audio samples need to be clear and concise, with little to no background noise. These audio samples had to be pre-processed such as trimmed to 10 seconds, have background music removed and have the length of silence between words and clauses made consistent across all files. A transcription of all files also had to be provided. Transcription was done manually initially, though later our group used Google_STT to provide a rough transcription of the files, which were later manually cleaned up. The audio files and transcription are necessary for the TensorFlowTTS and DC_TTS models to learn how to pronounce the 42 English phenomes. By the end, our dataset contained 460 audio sample files, totalling approximately 40 minutes.

For our project, high quality audio samples were difficult to obtain. Extracting audio from many different sources resulted in our audio samples being inconsistent. Some samples contained background noise such as music or audience sounds, and all samples had varying mic quality. Ideally, our model is trained on hours of voice samples taken from a single source in an isolated room, using a single high-quality microphone, which is difficult for our group to obtain.

Group Processes

Task Management & Development Tools

Our group's task management and development tools include Atlassian's Jira and Bitbucket, Zoom, Microsoft Office 365, Slack, TeamViewer and OneDrive. Tasks for the next weekly iteration sprint were allocated during Zoom meetings to all team members and were tracked using tickets on Jira's Kanban Board. All code contributions to the project were uploaded to BitBucket. Our automation scripts for the system pipeline were written using Python.

Since the client also provided a server machine, we were able to access it with TeamViewer and connect to the server remotely. The machine had multiple GPU's and CPU's, allowing us to run resource heavy programs quickly and easily (Refer to Appendix R).

Online Collaboration Tools

For group work, the team decided to use Slack, Microsoft Office 365, OneDrive, Jira, Miro, Zoom and Bitbucket:

- Our main communication tool was Slack where we updated each other, shared information like research papers and articles and kept others focused and on track. We had a couple channels separated by topic for easy search for example a deepfake channel and voice synthesis channel (Refer to Appendix R).
- To create presentation slides and write reports, we used Office 365. Other than for assessments, Office 365 was used to update clients and keep track of our own work (Refer to Appendix R).
- OneDrive was used to store documents and work on them simultaneously. OneDrive allowed users to edit documents as a team which is an excellent tool when everyone has to work remotely (Refer to Appendix R).
- Jira was used to assign tasks and manage sprints. Exact criteria for success was defined before a task was allocated to ensure success was achieved. A Kanban Board was necessary to see our progress and track our work. This software was used to define and assign tasks to individuals, with an equitable distribution of work always being at the focus of our efforts (Refer to Appendix R).
- For UML diagrams and visual representations of problems the team used Miro. All UML diagrams used for this report including the design of the system pipeline displayed to the client were made using this tool (Refer to Appendix R).
- Because of remote work, Zoom was a must to hold meetings.
- Bitbucket is a repository hosting platform where we share our code and work together on the software side of the project. Combining with Jira, we could link commits to Jira issues (Refer to Appendix R).

Overall, these tools were essential to this project as they not only increased productivity but also enabled us to collaborate as a group when we are all spread across the world due to COVID-19.

Client Interaction

Client interaction was done biweekly over Zoom and organised over discord (Refer to Appendix S). The main purpose of these meetings was to report on the progress the team has accomplished since the last meeting, and to discuss with the client complications encountered and their solutions.

For these meetings, we were also required to provide a one-minute presentation to summarise our progress (refer to Appendix S).

Toward the end of the project client interaction slowed to allow the team to focus on generating demos and the final deployment sprint. The client was still informed about our progress with the bi-weekly update videos prepared by the team.

Though the meetings were normally progress updates, our client helped us overcome two main obstacles:

Audio and Video syncing

An issue we faced early on in our project is what sort of pipeline our project would use due to lip-syncing issues. Initially, we believed we would have to manually create a video with the model lip-syncing the synthesised voice. The problem with this method is that this method was not feasible if dealing with multiple clients. When addressing this issue to our client, he provided a myriad of solutions, one of which was Wav2Lip, which is now an integral piece to our framework.

Hardware limitations

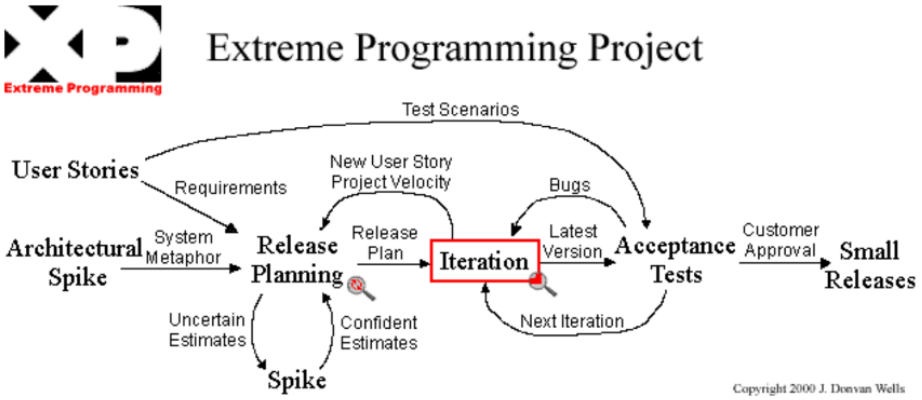
Most of our members do not have powerful GPUs for training tasks. We tried to resolve this issue by connecting to the university's UCPU server with SSH and work there, however undergraduate students only have limited working space and the training could not be done. To address this issue, the client has helped set up a remote server for us to operate with. Not only is the server significantly more powerful than any of our devices, but it's the only machine we can have "running in the background", which significantly increased our progress.

Documentation

Our project's documentation was created and hosted with Microsoft Word 365, Bitbucket Wiki, Miro and OneDrive. With the combined use of these software, our group has been able to complete documents both individually and as a team. Due to the nature of this project, pipeline design and the use of library tools was generally not in code format. Thus the documentation of work was critical and recorded mostly in documents hosted on OneDrive, or diagrams on Miro suitable to the final report. Since week 6 we have improved upon our efforts after reflection during the intermediatory report and all members now evenly contribute to the wiki page (refer to Appendix T). Weekly project updates, report writing and

meeting minutes is done as a group (refer to Appendix T). All notes and files can be found hosted on OneDrive (refer to Appendix T) which allows everyone to share and edit documents simultaneously.

Extreme Programming



J, Wells. Extreme Programming: A gentle introduction, http://www.extremeprogramming.org/map/project.html, Visited October 2020

Extreme Programming (XP) is an Agile process birthed in 1996 by Don Wells designed to handle projects with changing requirements [Wells 2013] by having an iterative process which produces small but regular releases [Wells 2013]. XP projects contain project roles ["Extreme Roles" 2006] which our team has implemented and modified to better suit our project. Roles such as Doomsayer has been allocated to all members, and roles such as Programmer has been turned into Programmer Lead to indicate that all members will be programmers within this project.

Roles:

Sidney Radwan – Project Manager

Gavin Arthur Stein – Customer Lead

Antony Tan – Tester Lead

Asim Saeed – Co-Programmer Lead

Heng Zhao – Co-Programmer Lead

Hoang Minh Le – Tracker

This does not mean people only took one XP programming role as during the project many people volunteered in other roles to help out teammates. Furthermore, in the initial stages of the project the team pair programmed on our two available desktop computers, later working in pairs on the client provided server machine keeping with the Extreme Programming practice as we found it most effective for our iteration sprints. Team members would work in pairs for task prints such as dataset collection and pre-processing, speech model training and other large sprints (Refer to Appendix U).

Group Reflections and Conclusions

Limitation Assessment

The main limiting factor for our project was the hardware demands machine learning imposed on our group. The challenges we faced due to this were:

- Testing and experimenting was difficult; a single change can take multiple days to see the effects of. Testing substitutes for components in our pipeline such as TensorflowTTS was costly time-wise and had to be put at the back of our priorities.
- Only two of the six members of our team have the hardware necessary to train our models.
- Training a model uses up all the processing power of the device, making it incapable of running other tasks. Training can only be done when the owner did not have to use it, meaning most training occurs late evening or early morning.

Though we received an additional server from our client, these problems will still be prevalent in the future by the users that our client will pass this project onto.

There are also certain limitations with our models and our pipeline. In terms of functionality, the issues we face are:

- Wav2Lip changes the lip movement of the actor to match the waveforms of the audio, not the phenomes of the speech. The lip movement is unlikely going to accurately reflect the speech.
- Speech synthesis is still being developed, and even the best of models (e.g., Google Cortana) are still monotone in nature and possess odd fluctuations in their speech. There will always be some form of “robotic-ness”; having our speech be motivational may be difficult, especially due to the subjective nature of motivation. Given current technology, motivation would be achieved by creating separate datasets consisting of a single ‘motivational’ emotion.
- TTS programs cannot replicate speech patterns like intonations, fluctuations etc. which means that it will not be able to replicate someone with a unique speaking style (e.g., Morgan Freeman)
- DeepFaceLab replaces only the head of the actor, which leads to the following limitations:
    - The actor must have a similar build to Morgan Freeman and have the same skin colour.
    - The video’s graphical fidelity, lighting quality, and lighting angle must be the same as the head-swap.

The structural, design, and implementation limitations our project face are:

- The main processing-sink the final product will face is Wav2Lip. Even though it only requires several minutes to apply, it is still a significant wait time. A high number of requests will result in significant server strain which can result in extremely long queues.

- There are a limited number of template videos we can produce. The pipeline has no way to automate new deep fake templates, so the final product will need to be constantly updated with new, interesting videos.
- The pipeline of our system is designed in such a way that the chosen models and tools can be easily swapped out. However, pivoting from chosen models and tools will take a significant number of resources due to the time it takes to train and set up models.

Extreme Programming and Group Processes

When appropriate, our team applied and exercised XP programming principles. Roles were rotated for the first 3 weeks which gave all members exposure to the different roles. We settled into more permanent roles, though in general members assisted each other when needed (e.g. Another member could act as Tracker and record meeting minutes for a certain day), though assistance was limited due to the nature of machine learning (e.g. there's no point in having multiple people train a speech model). Regardless, the team members have been able to consistently perform their allocated tasks.

In particular, the team has been very strong in Doomsaying (refer to Appendix Z). Every member was responsible in researching models and techniques to use in our project, which allowed us to coordinate a combination of models which are compatible with one another to form our pipeline. Communication has also been consistent, with members bringing up complications when encountered and willing to ask for help when needed (help was assigned via pair programming). These issues are brought up during team meetings or on Slack and tracked via meeting minutes and Jira tickets on the Kanban board. Each member has also been responsive in our communication channels, and rarely does a member fail to attend our weekly scheduled meetings.

Version control is done by uploading current states of models and its results onto OneDrive and Slack. Currently, version control has not been an important part of a project, as hardware limitations have made it difficult to edit our models, thus we do not have many versions of our work.

The team has found XP practices to be pivotal to the effective operation of teamwork and progress within this project. Pair programming being the most impactful practice to the success of our project. Although team member roles were at times fluid, having pre-determined roles kept the team focused and functioning when confusion about task allocation arose. We will continue to use Extreme Programming and Agile methodology in future projects.

Project Conclusion

This 13-week project, with the intention of delivering a system to produce motivational deepfaked videos of a chosen celebrity to our client and sponsor, Ben Sands from High Growth Ventures, has overall been a success. We have satisfied all core user stories and met all acceptance criteria for successful delivery. While the product has room for improvement, within the scope of this project and contextual limitations of existing technology, our team has considered our final delivered system a success. The team has gained invaluable experience in dealing with constantly changing client demands in an Agile environment, learning about key communication skills and expectation management. Also gaining experience working in a team of competent colleagues hailing from the same discipline. In retrospect, individual team members have gained insights to their own strengths and weaknesses, and how to leverage this in a goal-orientated team environment.

Taking the Project Further

There are several user stories suggested by our client that was not implemented as it was out of scope. The main ones were adding emotionality to our voice model with emotional tagging and generating 3D CGI models to automate stand-in actors for deepfake video creation. The main room for improvement in our project, however, is in our voice model.

To improve our model further, enlarging and improving the dataset of audio is needed to contain an equal and sufficient spread of data for every phenome. We have suggested the ideal solution to either have the celebrity or a voice impersonator read out sentences containing the 44 English phenomes, and use their recordings to train the models. If a future user of the system has the budget to pursue this route, it may be pursued.

To also help the DC_TTS model to learn to pronounce the same set of letters differently. A potential solution found is to convert our database's transcription and our user's input into ARPABET notation, but as discussed in the "Testing Conclusions" section, automating this process is difficult. An alternative would be to explore Tacotron2 as a replacement speech model, which we researched but could not implement due to time restrictions.

Furthermore, from demos of TensorflowTTS, which works with Tacotron2 or Fastspeech2, it is found that the library can supersede the quality of speech synthesis with DC_TTS and we recommend that future visitors to this project explore this option.

Individual Contributions

Antony Tan

Week 2

- Research on different DeepFake Technologies, includes First Order Motion demo

Week 3

- Research speech-to-speech models and its viability, include Real-Time-Voice-Cloning

Week 4

- Recorded 150 voice lines for testing use

Week 5

- Expanded on existing Morgan Freeman dataset with an extra 108 wave files (51-159)

Week 6

- Wrote user-experience acceptance tests
- Expanded dataset with an extra 62 files (160-222)

Week 7

- Performed Speech Accuracy tests

Week 8

- Transcribed voice lines gathered by Gavin, processed whole dataset according to requirements in Data Requirements section (271-370)
- reviewed accuracy of Google STT
- Normalisation script

Week 9-11

- Developing Tacotron2

Week 12

- Ran performance tests and speech accuracy tests
- Helped distribute and develop survey for user-experience testing
- Formatted Results with Hoang

Sidney Radwan

For all personal evidence refer to the folder here. Other important links found at head of this report.

Week 2

- Allocate the team members into research teams for deepfaking and for voice synthesis.
- Submit meeting minutes here to canvas as role rotation makes me Tracker
- Create group plan here (transferred into group contract)

Week 3

- Allocate members into research teams and demo production with DFL and voice synthesis
- Research into Deep Face Lab and begin installation and setup of environment to produce first demo. Research later collated into wiki and linked in week 6

Week 4

- Delivery of the first Morgan Freeman deepfake demo found here to the client on Monday
- Produced a better, full head demo of Morgan Freeman deepfake with Deep Face Lab for Friday's meeting with the client. Found here.

Week 5

- Initial research/demo attempts with TensorFlowTTS in pair programming with Gavin Stein. Paused from 25/9/20 to focus on presentation video. Pre-processed data here and screenshot here.
- Organise and focus team on group video presentation. Delegate sections

Week 6

- Complete or contribute to sections of the group report such as the Introduction, System Overview (in entirety), Tools and Components, Research, System Evaluation, Group Processes, Group Reflections, Conclusions and formatting
- Write the Deep Face Lab and System Architecture sections in the Wiki here and here respectively.

Week 7

- Client deployment preparation. Filmed system overview and progress video for client (~5minutes) found here

Week 8

- Deep Face Lab research on general models. Research added to wiki linked in week 6.

Week 9

- Develop improved Deep Face Lab model. Add GAN power to training. Shown to client here

Week 10

- DFL testing on server machine to develop final model and instructions. Screenshot here
- Deep Face Lab installation and use guide in deployment repo. Dfl.md readme created in dfmotiv/libraries/dfl/dfl.md found here and screenshot here

Week 11

- Transcribing audio dataset for DC_TTS. Screenshot here and folder here.

Week 12

- Create final DFL product and pair program with Asim to create final product used for user-experience testing here
- Conduct user-experience testing (36 responses). Results aggregated and presented from testing here
- Script writing and presentation; DFL sections, introduction, group processes, handling client requests(wrote), product satisfying user stories, XP programming and outro(wrote).

Week 13

- Packaged demos with commentary in zip folder for client here
- Complete sections of report including Introduction, System Architecture, Deep Face Lab (DFL) System Component, Deepfaking research, Testing Plan and Acceptance Tests.

Asim Saeed

Week 2

- Installed & Configured DeepFaceLab. Filmed + trained created the first deepfake "Roger Federer" demo as proof of concept requested by client.

Week 3

- Exploring DFL documentation for headswap.
- Recorded new footage for use as stand-in actor for headswap test
- Attempted headswap test (SAEHD training and XSeg). Unfortunately, my GPU just shy of the 6gb min of VRAM (at 4gb), Sidney to take over as his GTX 980ti with higher vram specs.
- Looked into DFL + Google Colab (cloud computing) as a solution to insufficient local specs

Week 4

- Briefly looked into Paperspace Gradient (Google Colab alternative)
- wav2lip configured on my machine (with dependencies + virtual env) and first demo

Week 5

- dc_tts configured on my machine (with dependencies + virtual env), pair programmed with Hoang to setup training workspace.
- Exported some more Wav2lip demos for client meeting (group supplied source media)

Week 6

- Helped Cian test and setup client machine ready for remote access by group.
- Brief research into suitable character rigs/3D animation packages.
- report and presentation work

Mid-semester break week

- formal evaluation cgi stand-ins and response for client, who persisted with this direction.
- new dc_tts "transfer learning" test
- ben's machine setup and tensorflowtts pair programming with Gavin + dc_tts research
- Created new dfmotiv repo and created plan for deployment

Week 7

- Setup structure of new repo, main readme and main dfmotiv.py module development
- Pair programmed with Hoang on dctts run.py updates
- First draft of installation docs readmes, collaborated with others to refine & reformat
- Continued refining transfer learning dctts model

Week 8 and 9

- Helped Heng + Antony cleanup increased merged dataset, then continued dctts .
- Created 5 motivational quotes (wavs; via dctts) for client to review and provide feedback on

Week 10 and 11

- Downloaded 2 New audiobooks (merged with the one Heng downloaded), extracted aax and converted to mp3. Paired with Heng and Sidney to transcribe. Began training new model with pure audiobook data.
- Created 5 quote videos using old model and 5 with new model for use with user survey
- Created intro MF voice with music wav (war of the worlds parody intro for presentation)

Week 12 and 13

- Recorded Demo video (edited down to ~7 minutes), cleanuped, zipped, and uploaded final deployment package. Link forwarded to Sidney.
- Presentation (demo/wav2lip/hardware restrict.), report(Implementation, CGI stand-ins)

Heng Zhao

Weekly status reports were done by Heng from week 7 to 12 (except week 8), and uploaded on canvas.

Week 2

- Researched on the papers for the project overview

Week 3

- Researched and test libraries on how to convert text to speech

Week 4

- Found the whole process of training QUICK96 and produced DFL models

Week 5

- Successfully decoded and modified the DFL models
- Found a way to save Deepfake models and reapply them to new destinations

Week 6

- Researched on PaperSpace

Week 7

- Found and extracted new M.F samples

Week 8

- Made GoogleSTT tools for automate transcribing
- Tested the tools with 300 audio files

Week 9

- Refactored the tools to support large and long audio files
- Tested the new tool with large and long audio files

Week 10

- Found and extracted new M.F samples

Week 11

- Presentation work

Week 12

- Report writing

Gavin Stein

Week 2

- Setup slack channel
- Establish client communications

Week 3

- Research and test speech synthesis libraries

Week 4

- Found, cut and cleaned 34 sample audios of Morgan Freeman
- Wrote speech to text transcription module to transcribe audio samples

Week 5

- set up TensorflowTTS - can't train on my computer though

Week 6

- building dataset
- report and presentation work
- setting up tensorflowTTS venv and starting training
- writing goal statement

Week 7

- automated speech inference using TensorFlowTTS
- Increasing speech dataset of M.F

Week 8

- Sourced 300 additional samples for speech dataset from publicly available resources
- Transcribed 100 of those samples for dataset
- Training TTS models with improved dataset

Week 9

- Research improvements to TensorFlowTTS
- Training TTS models with improved dataset
- Format libritts dataset and add M.F dataset for training fastspeech2

Week 10

- Training TTS models with improved dataset from scratch
- TensorflowTTS fine-tuning

Week 11

- TensorflowTTS fine-tuning
- Presentation work
- Final client deployment/handover

Week 12

- TensorflowTTS fine-tuning
- Report writing

Hoang Minh Le

Wiki is separated by weeks and within that by sections, please refer to it. **Meeting minutes was done by Hoang from week 7 to 12 (except week 8). Link for meeting minutes. For evidence, refer to link**

Week 2

- Setting up Jira, Bitbucket repo (image 1-3 in link)
- Research on Deepfake libraries
- Writing meeting minutes

Week 3

- Find sample of Morgan Freeman video, and cut them clean them (image 4 in link above)
- Collect Morgan Freeman voice actors (image 5)

Week 4

- Setting up voice synthesis library and run an example (image 6)
- Create script for voice synthesis (150 lines) (image 7)

Week 5

- Need to work on voice synthesis, as I do not meet the pc requirement to run the training. Pair programming with Asim for voice synthesis (image 6)
- Research on emotional tagging for voice synthesis (image 8)

Week 6

- Report writing and presentation work (to find out what exactly I wrote in presentation and report, see Task Distribution section under the Week 6 part of Wiki)

Week 7

- setting up server machine and run voice synthesis (DCTTS) (image 9)
- Automation of voice synthesis part and documentation (DCTTS) (image 10)

Week 8

- Unit testing and its documentation (image 11)

Week 9

- Research noise problem of voice synthesis (image 12)

Week 10

- Finish unit testing and start creating user experience testing (write template)

Week 11

- writing out presentation guideline

Week 12 - 13

- Presentation and report writing (to find out what I wrote, see section Task Distribution section under Week 12 part of Wiki)

References

E, R, Lai. (2011), Motivation: A literature review. Retrieved from http://images.pearsonassessments.com/images/tmrs/motivation_review_final.pdf, Visited October 2020

Extreme Roles. (2006). Retrieved from http://wiki.c2.com/?ExtremeRoles, Visited October 2020

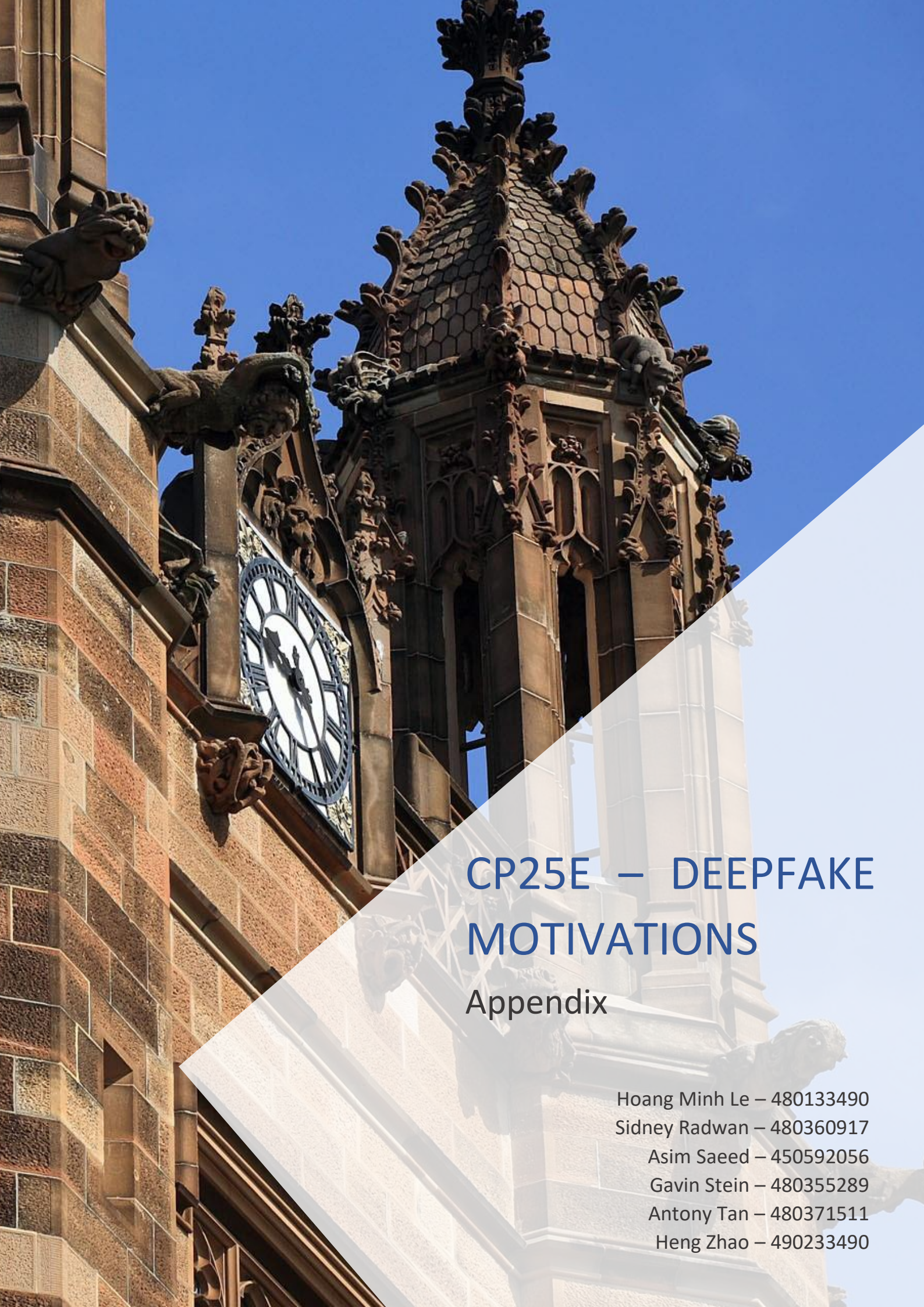
J, Wells. (2013), Extreme Programming: A gentle introduction. Retrieved from http://www.extremeprogramming.org/, Visited October 2020

Mika Westerlund. (2019), The Emergence of Deepfake Technology: A Review. Retrieved from https://timreview.ca/article/1282, Visited November 2019

Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., Skerry-Ryan, R. J., Saurous, R. A., Agiomyrgiannakis Y., Wu, Y. (2017). Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions. Retrieved from https://arxiv.org/abs/1712.05884 , Visited October 2020

T, Tse, K, Bheemaiah, M, Esposito. (2017), What is machine learning? Retrieved from https://theconversation.com/what-is-machine-learning-76759, Visited October 2020

Tachibana, H., Uenoyama, K., Aihara, S. (2017). Efficiently Trainable Text-to-Speech System Based on Deep Convolutional Networks with Guided Attention. Retrieved from https://arxiv.org/abs/1710.08969, Visited October 2020



CP25E – DEEPFAKE MOTIVATIONS

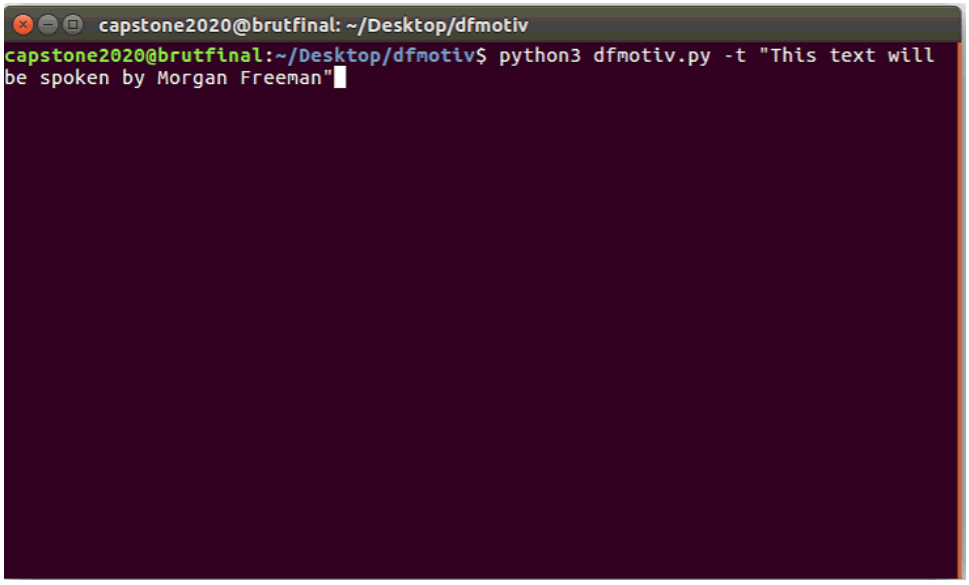
Appendix

Hoang Minh Le – 480133490
Sidney Radwan – 480360917
Asim Saeed – 450592056
Gavin Stein – 480355289
Antony Tan – 480371511
Heng Zhao – 490233490

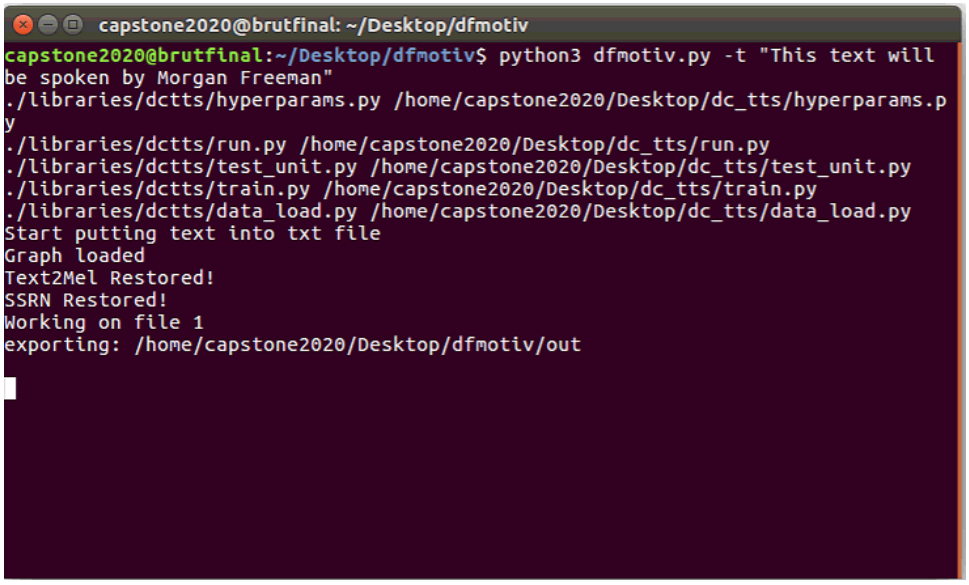
Appendices

Full evidence folder containing all evidence below can be found here.

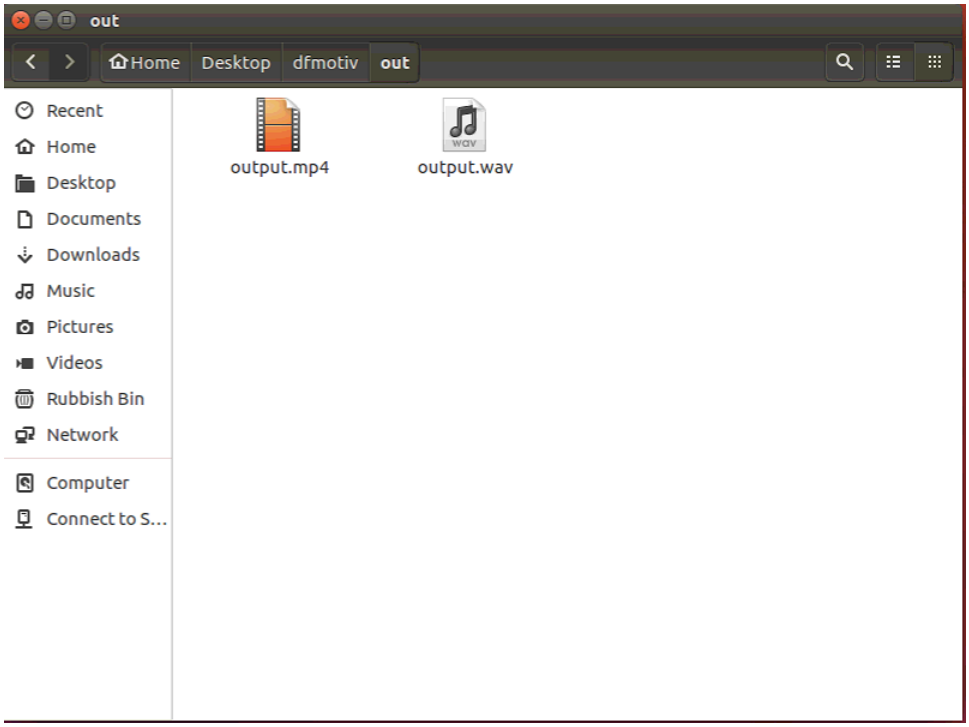
Appendix A: User story satisfaction



[User story 2 & 3] Command from user to start system – requires text input for script using *–t* flag. Video has already been produced and provided by user.

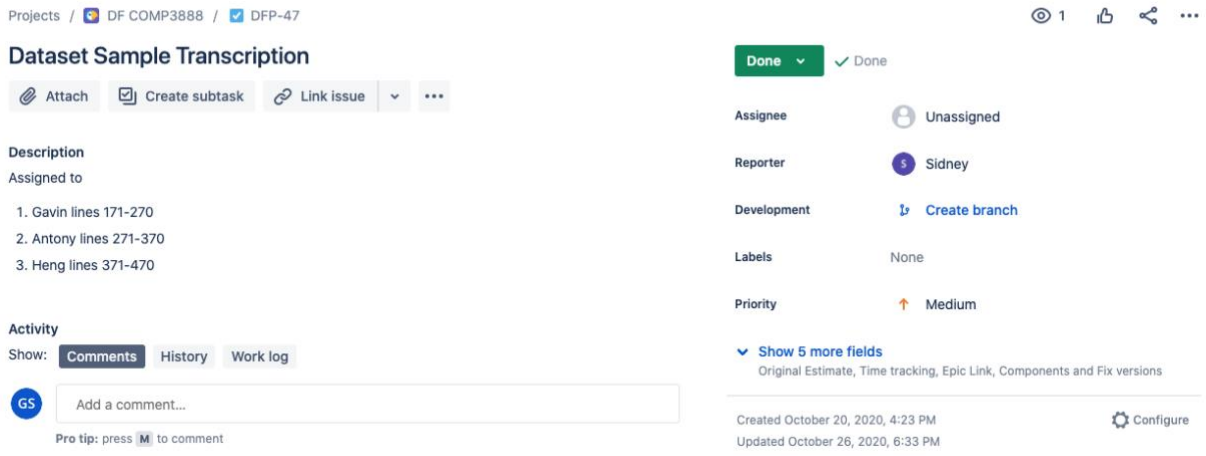


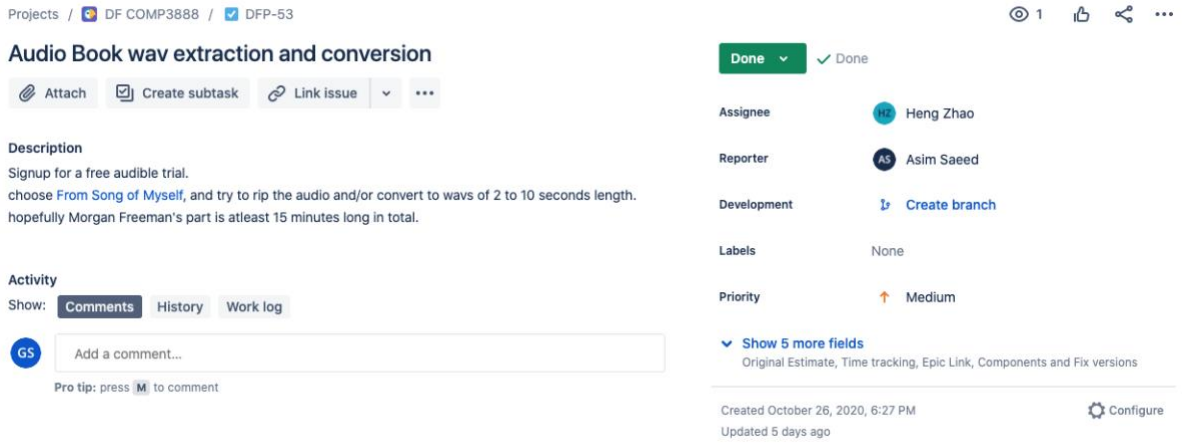
[User story 2, 3, 4 & 5] Process synthesising audio and video, and combining the two, then exporting output files to given out directory for user. All automated after user calls dfmotiv.py.

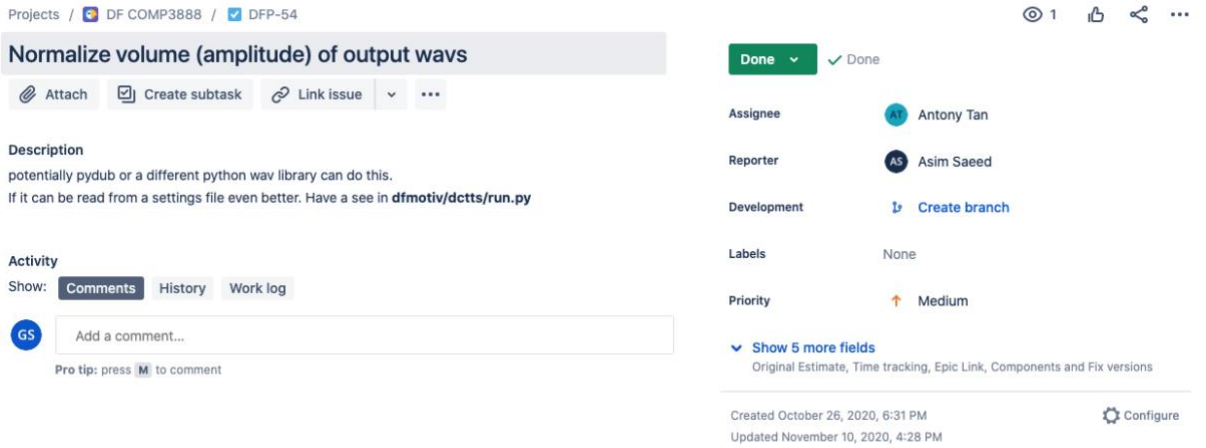


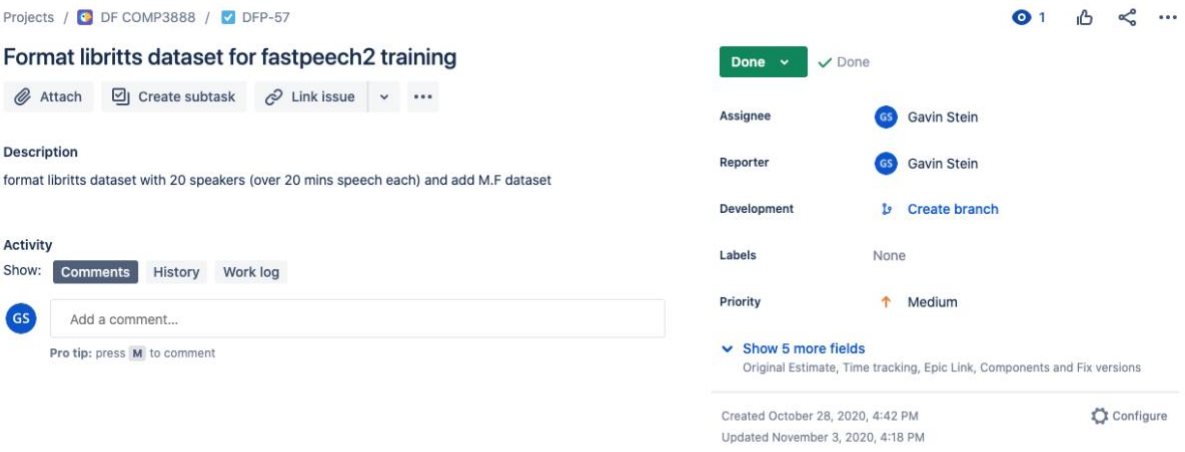
[User story 1] Out directory with produced video (mp4) and audio (wav). These are what will be provided to a customer of the service.

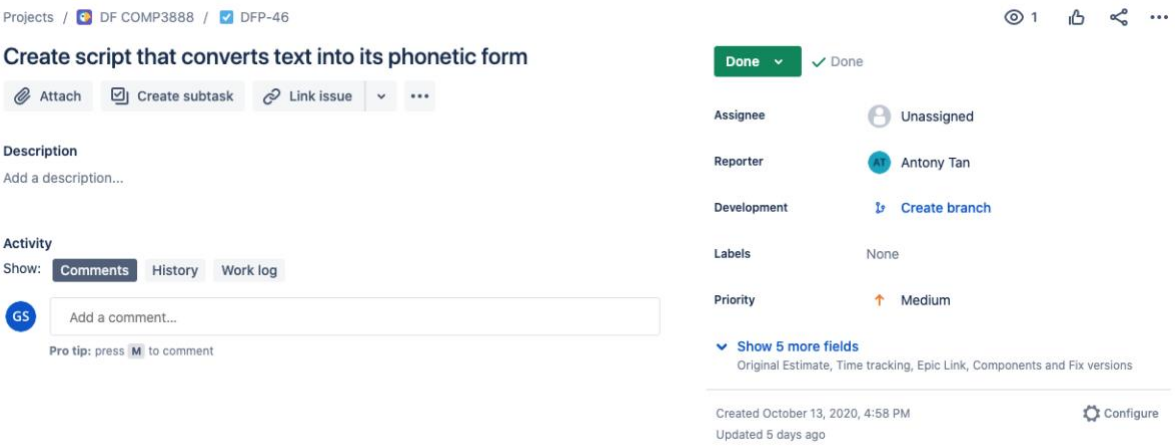
Appendix B: Jira evidence of dataset refinement



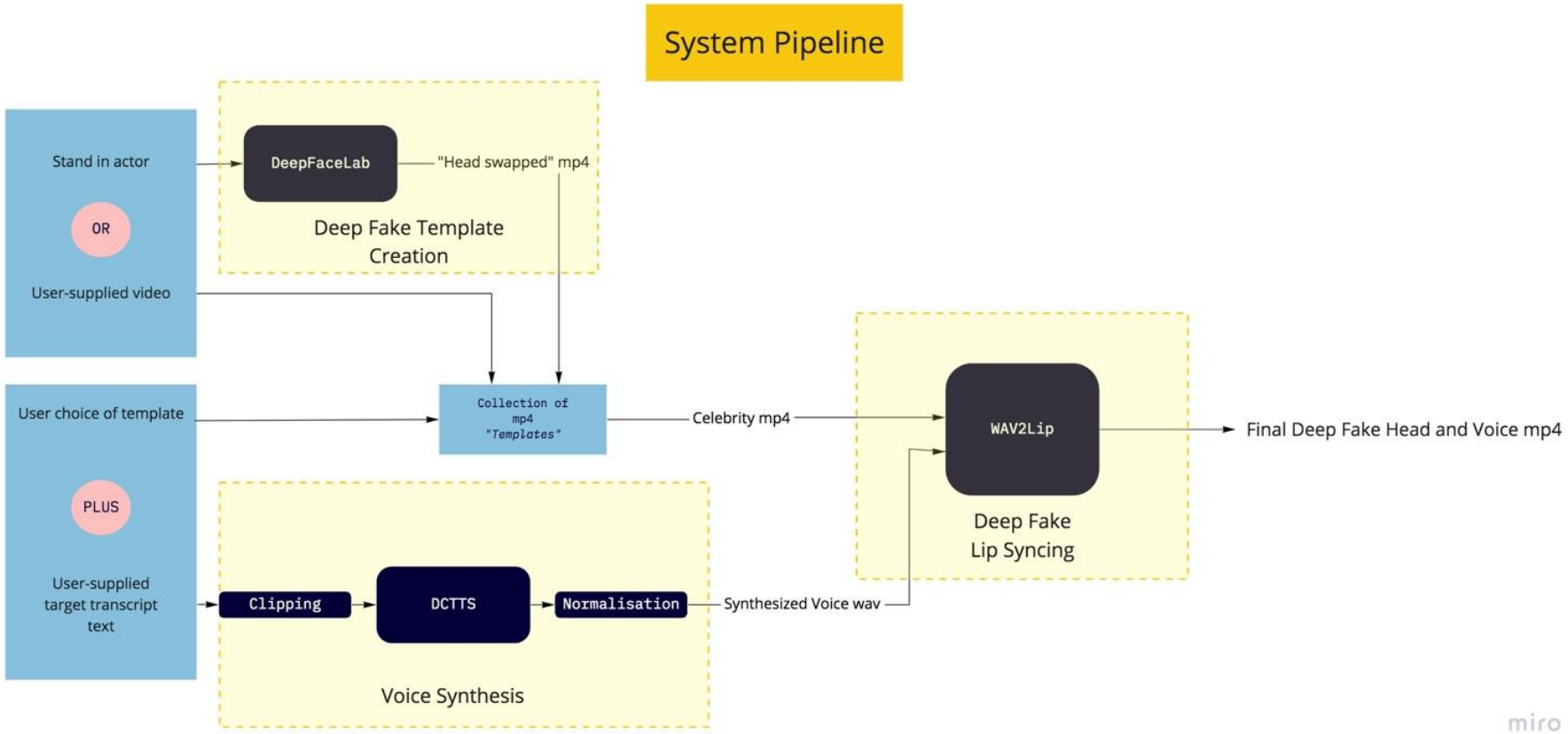




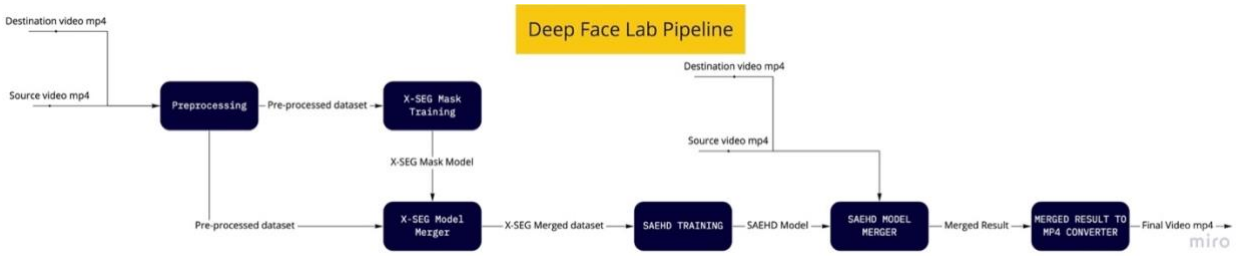

Projects / DF COMP3888 / DFP-57
1
Format libritts dataset for fastpeech2 training
Attach
Create subtask
Link issue
Description
format libritts dataset with 20 speakers (over 20 mins speech each) and add M.F dataset
Activity
Show:
Comments
History
Work log
GS
Add a comment...
Pro tip: press M to comment
Done
Done
Assignee
Gavin Stein
Reporter
Gavin Stein
Development
Create branch
Labels
None
Priority
Medium
Show 5 more fields
Original Estimate, Time tracking, Epic Link, Components and Fix versions
Created October 28, 2020, 4:42 PM
Updated November 3, 2020, 4:18 PM
Configure


Projects / DF COMP3888 / DFP-46
1
Create script that converts text into its phonetic form
Attach
Create subtask
Link issue
Description
Add a description...
Activity
Show:
Comments
History
Work log
GS
Add a comment...
Pro tip: press M to comment
Done
Done
Assignee
Unassigned
Reporter
Antony Tan
Development
Create branch
Labels
None
Priority
Medium
Show 5 more fields
Original Estimate, Time tracking, Epic Link, Components and Fix versions
Created October 13, 2020, 4:58 PM
Updated 5 days ago
Configure

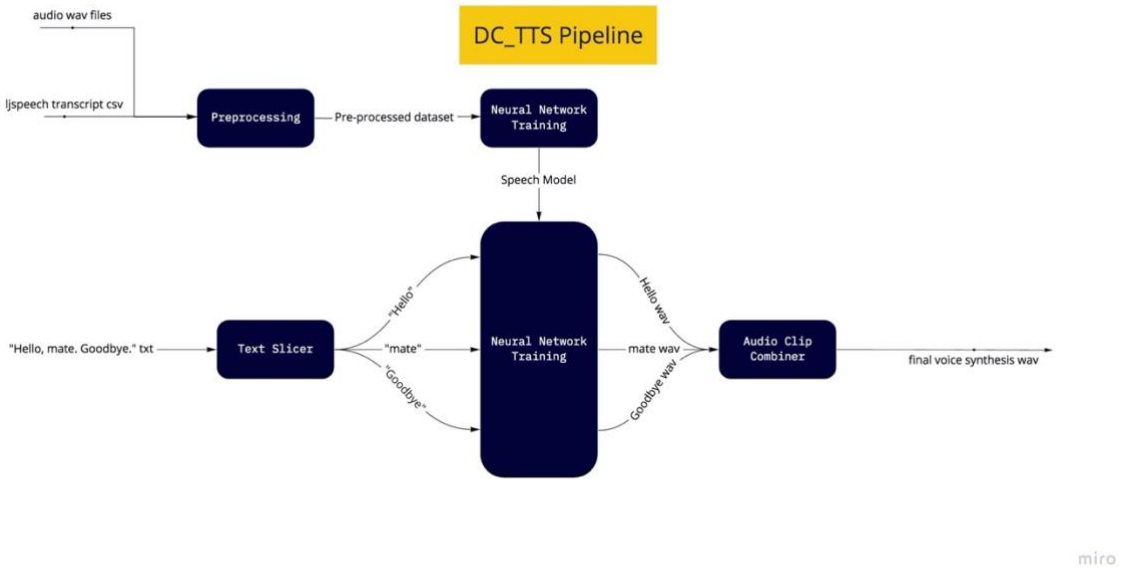
Appendix C: Final System Pipeline full size



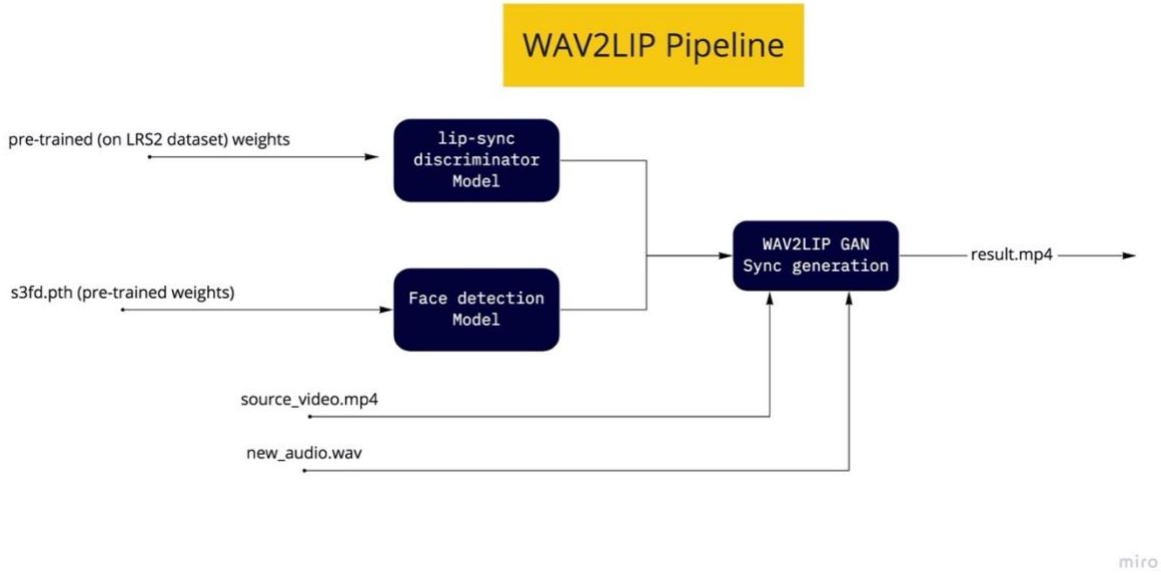
Appendix D: Component and Tool UML Diagrams



Deep Face Lab UML Diagram

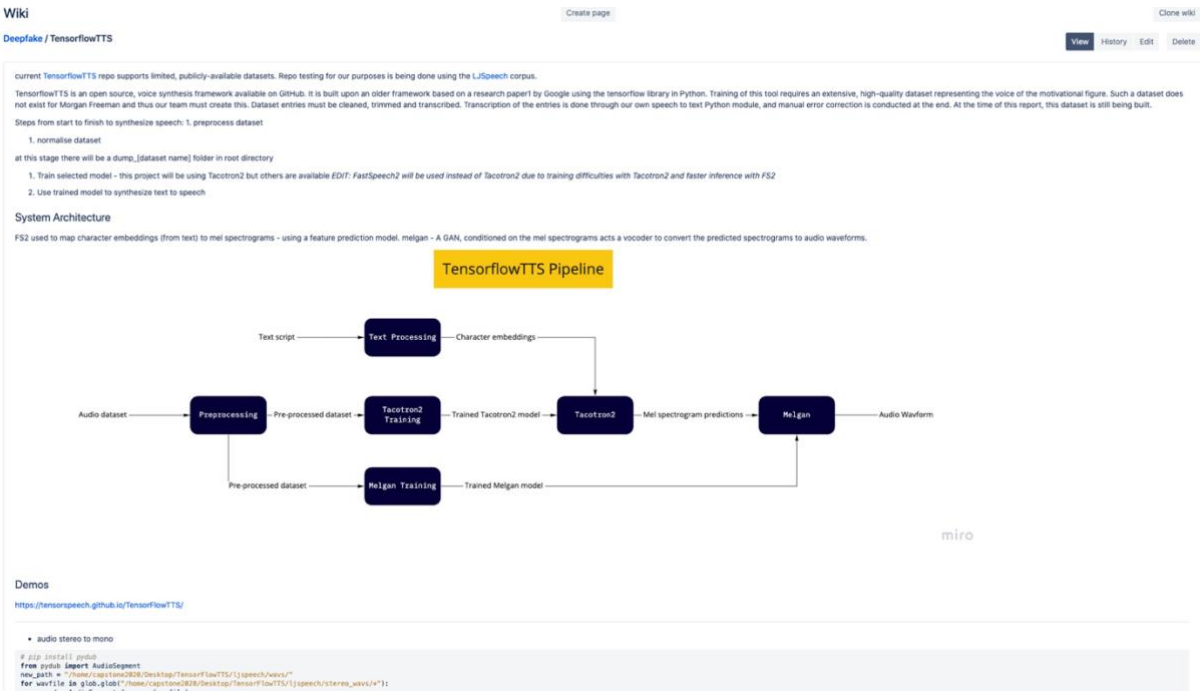


DC_TTS UML Diagram



WAV2Lip UML Diagram

Appendix E: TensorflowTTS wiki



https://bitbucket.org/mile3901/deepfake/wiki/TensorflowTTS

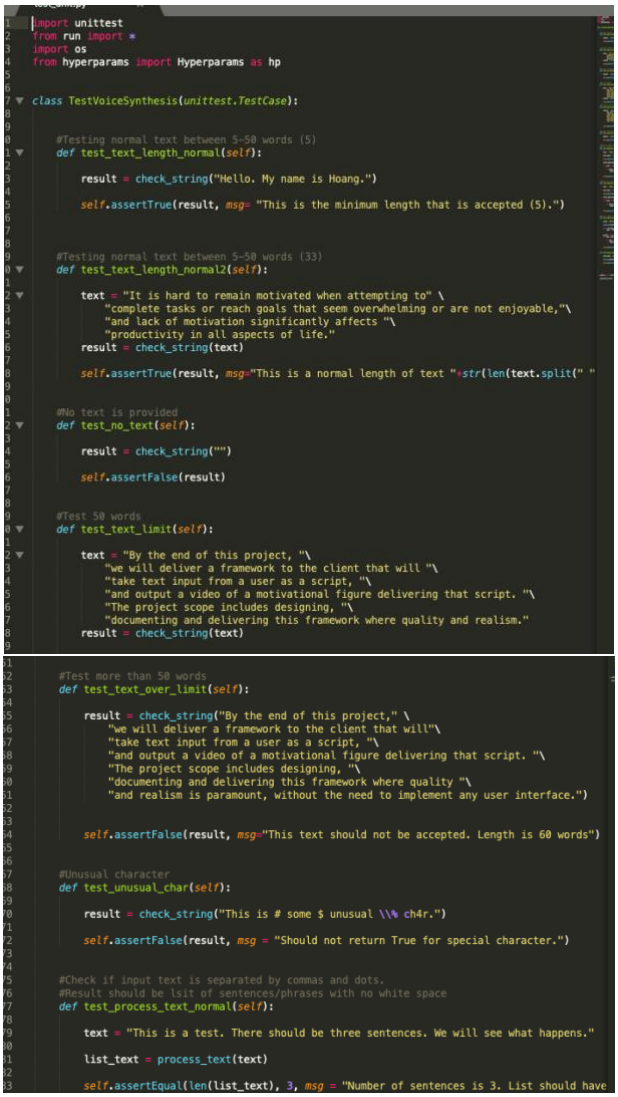
This picture is for reference only – please follow link for content.

Appendix F: Usability Survey

Accessibility	*Strongly Disagree 1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	Strongly agree
The steps needed to run the project were easy for me to do										
The steps needed to run the project were simple										
I am satisfied with the number of steps needed to receive my motivational video										
I am satisfied with how long it takes to receive my motivational video										
I am willing to go through the whole process again to receive another video										

Screenshot of usability survey

Appendix G: Unit Testing and Regression Testing



```
import unittest
from run import *
import os
from hyperparams import Hyperparams as hp


class TestVoiceSynthesis(unittest.TestCase):


    #Testing normal text between 5-50 words (5)
    def test_text_length_normal(self):

        result = check_string("Hello. My name is Hoang.")

        self.assertTrue(result, msg= "This is the minimum length that is accepted (5).")



    #Testing normal text between 5-50 words (33)
    def test_text_length_normal2(self):

        text = "It is hard to remain motivated when attempting to" \
            "complete tasks or reach goals that seem overwhelming or are not enjoyable,"\
            "and lack of motivation significantly affects "\
            "productivity in all aspects of life."
        result = check_string(text)

        self.assertTrue(result, msg="This is a normal length of text "+str(len(text.split(" "


    #No text is provided
    def test_no_text(self):

        result = check_string("")

        self.assertFalse(result)


    #Test 50 words
    def test_text_limit(self):

        text = "By the end of this project, "\
            "we will deliver a framework to the client that will "\
            "take text input from a user as a script, "\
            "and output a video of a motivational figure delivering that script. "\
            "The project scope includes designing, "\
            "documenting and delivering this framework where quality and realism."
        result = check_string(text)
```

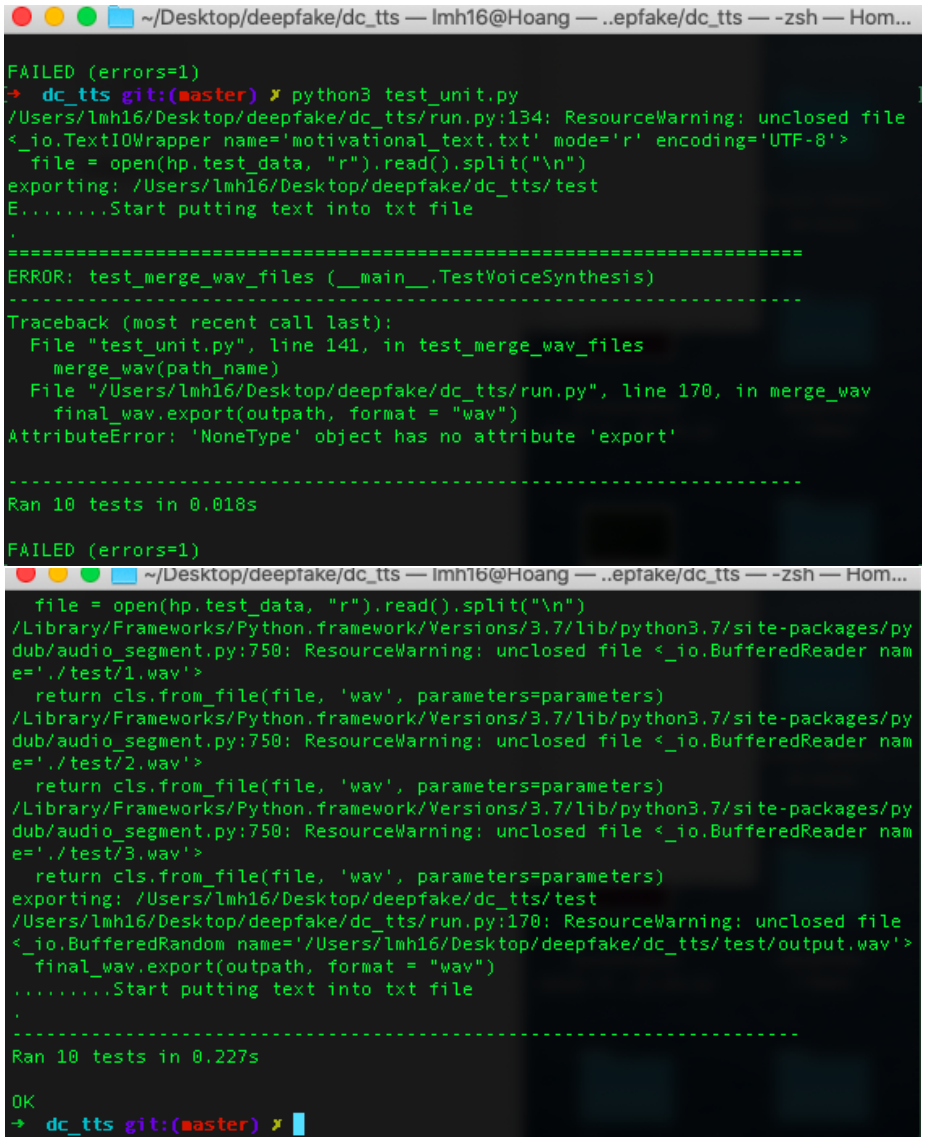
```
    #Test more than 50 words
    def test_text_over_limit(self):

        result = check_string("By the end of this project," \
            "we will deliver a framework to the client that will"\
            "take text input from a user as a script, "\
            "and output a video of a motivational figure delivering that script. "\
            "The project scope includes designing, "\
            "documenting and delivering this framework where quality "\
            "and realism is paramount, without the need to implement any user interface.")


        self.assertFalse(result, msg="This text should not be accepted. Length is 60 words")


    #Unusual character
    def test_unusual_char(self):

        result = check_string("This is # some $ unusual \\% ch4r.")

        self.assertFalse(result, msg = "Should not return True for special character.")


    #Check if input text is separated by commas and dots.
    #Result should be lsit of sentences/phrases with no white space
    def test_process_text_normal(self):

        text = "This is a test. There should be three sentences. We will see what happens."

        list_text = process_text(text)

        self.assertEqual(len(list_text), 3, msg = "Number of sentences is 3. List should have
```

Screenshots of unit and regression testing code. Testing file test_unit.py can be found in Bitbucket repository. Failing cases indicate regression occurred when a change was introduced.

How to run unit testing

1. Fork the DCTTS library
2. Clone our project repository; dfmotiv
3. From dfmotiv/libraries/dctts folder copy paste everything into the DCTTS folder (allow replacement)
4. In the forked DCTTS folder, create a folder called *samples* and put some wav files
5. Run test_unit.py

Proof of Testing



Appendix H: User Experience Testing Survey

Deepfake	*Strongly Disagree 1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	Strongly agree
The head replacement looked like Morgan Freeman										
The quality and resolution of the head was consistent with the rest of the video										
The deepfake looked realistic										
Speech Synthesis	*Strongly Disagree 1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	Strongly agree
The synthesised voice sounds like Morgan Freeman										
The synthesised voice has similar intonations and enunciations to Morgan Freeman										
The speech synthesis sounded realistic										
Video	*Strongly Disagree 1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	Strongly agree
The lip-syncing looked realistic										
The video looked realistic										
The video was motivational										

Screenshot of user-experience survey template

Appendix I: Unit Testing and Regression Testing Results

Scenario	Attempt Number	Test Nature	Test File	Code Source	Code function	Expected result	Success	Error
User provides text with length between 5 and 50 words	1	normal case	dcttu/unit_test.py	dcttu/run.py	check_string	return True and accept text		Test case issue: example/test text input was incorrect
User provides text with length between 5 and 50 words	2	normal case	dcttu/unit_test.py	dcttu/run.py	check_string	return True and accept text		
User provides a text, 5 words	1	boundary case	dcttu/unit_test.py	dcttu/run.py	check_string	return True and accept text		
User provides a text with 50 words	1	boundary case	dcttu/unit_test.py	dcttu/run.py	check_string	return True and accept text		Test case issue: example/test text input was incorrect
User provides a text with 50 words	2	boundary case	dcttu/unit_test.py	dcttu/run.py	check_string	return True and accept text		
User provided no text	1	abnormal case	dcttu/unit_test.py	dcttu/run.py	check_string	return False and reject		
User provide unusual character (e.g. # $)	1	abnormal case	dcttu/unit_test.py	dcttu/run.py	check_string	return false and reject		
User provides a text longer than 50 words	1	abnormal case	dcttu/unit_test.py	dcttu/run.py	check_string	return false and reject text		
Check if text is preprocessed: sentences are separated and put into a list	1	normal case	dcttu/unit_test.py	dcttu/run.py	process_text	return a list of strings where every element must not start with whitespace		
Check if text is preprocessed: comma separated phrases should be split and put into list	1	normal case	dcttu/unit_test.py	dcttu/run.py	process_text	return list of string where every element must not start or end with whitespace		
Check if text is written to the right file	1	normal case	dcttu/unit_test.py	dcttu/run.py	write_text	create a new file with text in it. Each sentences are in one line		
Check if wav files are combined into one and saved into the right place	1	normal case	dcttu/unit_test.py	dcttu/run.py	merge_wav	one wav file should be created from multiple ones with specified path name		Test case issue: wrong attribute was used in testcase (oops)
Check if wav files are combined into one and saved into the right place	2	normal case	dcttu/unit_test.py	dcttu/run.py	merge_wav	one wav file should be created from multiple ones with specified path name		Test case issue: forgot to set up the correct path before testing
Check if wav files are combined into one and saved into the right place	3	normal case	dcttu/unit_test.py	dcttu/run.py	merge_wav	one wav file should be created from multiple ones with specified path name		
No template video provided to system	1	abnormal case	dfmotiv_test.py	dfmotiv.py	main	return False and reject		Test case issue: no check for missing template video
No template video provided to system	2	abnormal case	dfmotiv_test.py	dfmotiv.py	main	return False and reject		
Template video provided to system	1	normal case	dfmotiv_test.py	dfmotiv.py	main	return True and accept		

Screenshot is for reference only - for log of testing refer to link: https://unisydneyedu-my.sharepoint.com/:x:/g/personal/gste6115_uni_sydney_edu_au/ETRRmWEVVttInE_Mskw4XRcBoM8q_k06NeHHkj3Zt38mFQ?e=NST9Hx

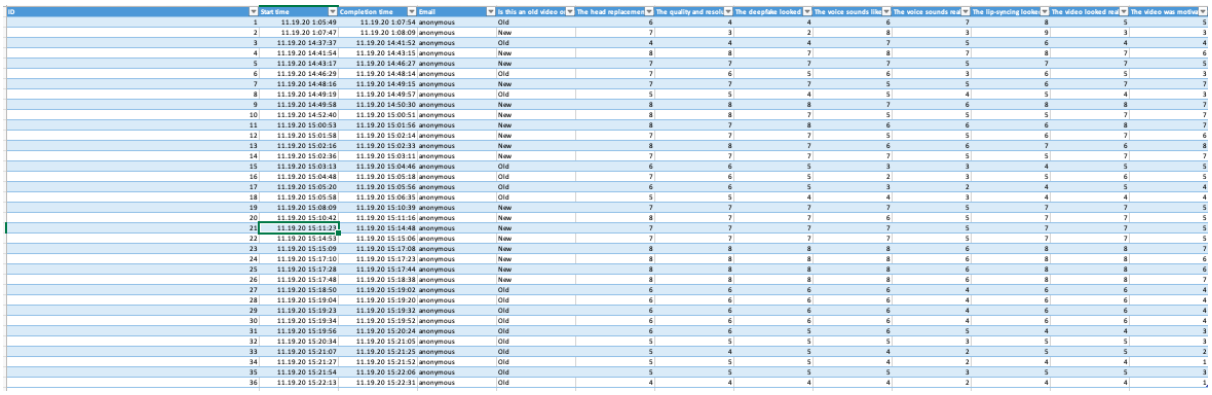
Appendix J: User-Experience Testing Products

Screenshot of five quote video with old deepfake and voice synthesis used for user-experience testing. Link: https://unisydneyedu-my.sharepoint.com/:v:/g/personal/gste6115_uni_sydney_edu_au/EUdk5jO2wK5AqMaLm-npjI4BTJhcutWx_ZpVGO_pat2gnQ?e=noA4YF

Screenshot of five quote video with final deepfake and voice synthesis used for user-experience testing. Link: https://unisydneyedu-my.sharepoint.com/:v:/g/personal/gste6115_uni_sydney_edu_au/EXvOlqtWJ0JOjYZG5dagRgsBcCUHEUCmz7ZakhJ-4Auvag?e=C1uWa7

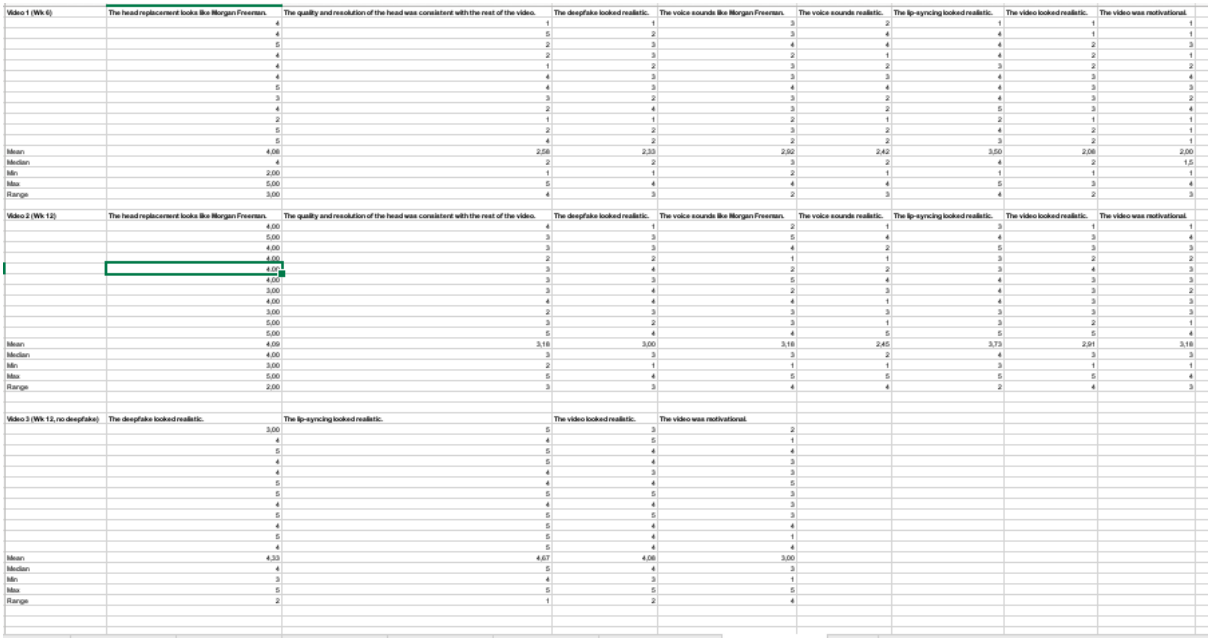
Screenshot of five quote video with The Shawshank Redemption clip and final voice synthesis used for user-experience testing. Link: https://unisydneyedu-my.sharepoint.com/:v:/g/personal/gste6115_uni_sydney_edu_au/EV4xbZS8Mm5IvudCse3QEGABZTQ7C51rt8z_IR-b0hLCsw?e=cxerFI

Appendix K: User-Experience Testing Results



ID	Start time	Completion time	Email	Is this an old video o	The head replacement	The quality and resol	The deepfake looked	The voice sounds like	The voice sounds rea	The lip-syncing looke	The video looked rea	The video was motiva
1	11.19.20 1:05:49	11.19.20 1:07:54	anonymous	Old	6	4	4	6	7	8	5	5
2	11.19.20 1:07:47	11.19.20 1:08:09	anonymous	New	7	3	2	8	3	9	3	3
3	11.19.20 14:37:37	11.19.20 14:41:52	anonymous	Old	4	4	4	7	5	6	4	4
4	11.19.20 14:41:54	11.19.20 14:43:15	anonymous	New	8	8	7	8	7	8	7	6
5	11.19.20 14:43:17	11.19.20 14:46:27	anonymous	New	7	7	7	7	5	7	7	5
6	11.19.20 14:46:29	11.19.20 14:48:14	anonymous	Old	7	6	5	6	3	6	5	3
7	11.19.20 14:48:16	11.19.20 14:49:15	anonymous	New	7	7	7	5	5	6	7	7
8	11.19.20 14:49:19	11.19.20 14:49:57	anonymous	Old	5	5	4	5	4	5	4	3
9	11.19.20 14:49:58	11.19.20 14:50:30	anonymous	New	8	8	8	7	6	8	8	7
10	11.19.20 14:52:40	11.19.20 15:00:51	anonymous	New	8	8	7	5	5	5	7	7
11	11.19.20 15:00:53	11.19.20 15:01:56	anonymous	New	8	7	8	6	6	6	8	7
12	11.19.20 15:01:58	11.19.20 15:02:14	anonymous	New	7	7	7	5	5	6	7	6
13	11.19.20 15:02:16	11.19.20 15:02:33	anonymous	New	8	8	7	6	6	7	6	8
14	11.19.20 15:02:36	11.19.20 15:03:11	anonymous	New	7	7	7	7	5	5	7	7
15	11.19.20 15:03:13	11.19.20 15:04:46	anonymous	Old	6	6	5	3	3	4	5	5
16	11.19.20 15:04:48	11.19.20 15:05:18	anonymous	Old	7	6	5	2	3	5	6	5
17	11.19.20 15:05:20	11.19.20 15:05:56	anonymous	Old	6	6	5	3	2	4	5	4
18	11.19.20 15:05:58	11.19.20 15:06:35	anonymous	Old	5	5	4	4	3	4	4	4
19	11.19.20 15:08:09	11.19.20 15:10:39	anonymous	New	7	7	7	7	5	7	7	5
20	11.19.20 15:10:42	11.19.20 15:11:16	anonymous	New	8	7	7	6	5	7	7	5
21	11.19.20 15:11:21	11.19.20 15:14:48	anonymous	New	7	7	7	7	5	7	7	5
22	11.19.20 15:14:51	11.19.20 15:15:06	anonymous	New	7	7	7	7	5	7	7	5
23	11.19.20 15:15:09	11.19.20 15:17:08	anonymous	New	8	8	8	8	6	8	8	7
24	11.19.20 15:17:10	11.19.20 15:17:23	anonymous	New	8	8	8	8	6	8	8	6
25	11.19.20 15:17:28	11.19.20 15:17:44	anonymous	New	8	8	8	8	6	8	8	6
26	11.19.20 15:17:48	11.19.20 15:18:38	anonymous	New	8	8	8	8	6	8	8	7
27	11.19.20 15:18:50	11.19.20 15:19:02	anonymous	Old	6	6	6	6	4	6	6	4
28	11.19.20 15:19:04	11.19.20 15:19:20	anonymous	Old	6	6	6	6	4	6	6	4
29	11.19.20 15:19:23	11.19.20 15:19:32	anonymous	Old	6	6	6	6	4	6	6	4
30	11.19.20 15:19:34	11.19.20 15:19:52	anonymous	Old	6	6	6	6	4	6	6	4
31	11.19.20 15:19:56	11.19.20 15:20:24	anonymous	Old	6	6	5	6	5	4	4	3
32	11.19.20 15:20:34	11.19.20 15:21:05	anonymous	Old	5	5	5	5	3	5	5	3
33	11.19.20 15:21:07	11.19.20 15:21:25	anonymous	Old	5	4	5	4	2	5	5	2
34	11.19.20 15:21:27	11.19.20 15:21:52	anonymous	Old	5	5	5	4	2	4	4	1
35	11.19.20 15:21:54	11.19.20 15:22:06	anonymous	Old	5	5	5	5	3	5	5	3
36	11.19.20 15:22:13	11.19.20 15:22:31	anonymous	Old	4	4	4	4	2	4	4	1

Screenshot is for reference only – for survey feedback 1 refer to link: https://unisydneyedu-my.sharepoint.com/:x:/g/personal/gste6115_uni_sydney_edu_au/EdQ7mqs3ZZRDpEfHHjjKfN4BX4ZHm767oAGFPp-6GDxXFw?e=iMMwhz



Video 1 (Wk 6)	The head replacement looks like Morgan Freeman.	The quality and resolution of the head was consistent with the rest of the video.	The deepfake looked realistic.	The voice sounds like Morgan Freeman.	The voice sounds realistic.	The lip-syncing looked realistic.	The video looked realistic.	The video was motivational.
	4	1	1	3	2	1	1	1
	4	5	2	3	4	4	1	1
	5	2	3	4	4	4	2	3
	4	2	3	2	1	4	2	1
	4	1	2	3	2	3	2	2
	4	4	3	3	3	4	3	4
	5	4	3	4	4	4	3	3
	3	3	2	3	2	4	3	2
	4	2	4	3	2	5	3	4
	2	1	1	2	1	2	1	1
	5	2	2	3	2	4	2	1
	5	4	2	2	2	3	2	1
Mean	4,06	2,58	2,33	2,92	2,42	3,50	2,06	2,00
Median	4	2	2	3	2	4	2	1,5
Min	2,00	1	1	2	1	1	1	1
Max	5,00	5	4	4	4	5	3	4
Range	3,00	4	3	2	3	4	2	3

Video 2 (Wk 12)	The head replacement looks like Morgan Freeman.	The quality and resolution of the head was consistent with the rest of the video.	The deepfake looked realistic.	The voice sounds like Morgan Freeman.	The voice sounds realistic.	The lip-syncing looked realistic.	The video looked realistic.	The video was motivational.
	4,00	4	1	2	1	3	1	1
	5,00	3	3	5	4	4	3	4
	4,00	3	3	4	2	5	3	3
	4,00	2	2	1	1	3	2	2
	4,00	3	4	2	2	3	4	3
	4,00	3	3	5	4	4	3	3
	3,00	3	4	2	3	4	3	2
	4,00	4	4	4	1	4	3	3
	3,00	2	3	3	3	3	3	3
	5,00	3	2	3	1	3	2	1
	5,00	5	4	4	5	5	5	4
Mean	4,09	3,18	3,00	3,18	2,45	3,73	2,91	3,18
Median	4,00	3	3	3	2	4	3	3
Min	3,00	2	1	1	1	3	1	1
Max	5,00	5	4	5	5	5	5	4
Range	2,00	3	3	4	4	2	4	3

Video 3 (Wk 12, no deepfake)	The deepfake looked realistic.	The lip-syncing looked realistic.	The video looked realistic.	The video was motivational.
	3,00	5	3	2
	4	4	5	1
	5	5	4	4
	4	5	4	3
	4	4	3	3
	5	4	4	5
	5	5	5	3
	4	4	4	3
	5	5	5	3
	4	5	4	4
	5	5	4	1
	4	5	4	4
Mean	4,33	4,67	4,08	3,00
Median	4	5	4	3
Min	3	4	3	1
Max	5	5	5	5
Range	2	1	2	4

Screenshot is for reference only – for survey feedback 2 refer to link: https://unisydneyedu-my.sharepoint.com/:x:/g/personal/gste6115_uni_sydney_edu_au/EdQ7mqs3ZZRDpEfHHjjKfN4BX4ZHm767oAGFPp-6GDxXFw?e=iMMwhz

Video 1 (Wk 6)	Mean	Median	Range	Max	Min
The head replacement looks like Morgan Freeman.	4,82	6	5	7	2
The quality and resolution of the head was consistent with the rest of the video.	3,93	5,5	5	6	1
The deepfake looked realistic.	3,64	5	5	6	1
The voice sounds like Morgan Freeman.	3,90	5	5	7	2
The voice sounds realistic.	2,96	3	5	7	2
The lip-syncing looked realistic.	4,33	5	7	8	1
The video looked realistic.	3,51	5	5	6	1
The video was motivational.	2,72	4	4	5	1

Video 2 (Wk 13)	Mean	Median	Range	Max	Min
The head replacement looks like Morgan Freeman.	5,82	8	5	8	3
The quality and resolution of the head was consistent with the rest of the video.	5,20	7	6	8	2
The deepfake looked realistic.	5,03	7	7	8	1
The voice sounds like Morgan Freeman.	5,01	7	7	8	1
The voice sounds realistic.	3,92	5	6	7	1
The lip-syncing looked realistic.	5,39	7	6	9	3
The video looked realistic.	4,98	7	7	8	1
The video was motivational.	4,62	6	4	5	1

Video 3 (Wk 12, no deepfake)	Mean	Median	Min	Max	Range
The deepfake looked realistic.	4,33	4	3	5	2
The lip-syncing looked realistic.	4,67	5	4	5	1
The video looked realistic.	4,08	4	3	5	2
The video was motivational.	3,00	3	1	5	4

Screenshot of aggregated results of user-experience testing

Appendix L: Benchmark Speech Accuracy Testing

1. He invested some skill in charisma and strength. Static
2. I like to leave work after my eight-hour work break
3. Tomorrow will bring something new, so leave today as a memory
4. He had concluded that pigs must be able to fly in Hog heaven
5. The white-water rafting trip was suddenly halted by an unexpected brick wall
6. He was willing to find the depths of the rabbit hole in order to be with her.
7. He's in a boy band which doesn't make much sense for a snake
8. Wisdom is easily acquired when hiding under the bed with a saucepan on your head.
9. When he had to picnic on the beach, he purposefully put sand in other people's food.
10. The waitress was not amused when he ordered green eggs and ham
11. Three generations with six decades of life experience.
12. The ants enjoyed the barbecue more than the family.
13. She had some amazing news to share but nobody to share it with.
14. It's not possible to convince a monkey to give you a banana by promising it infinite bananas when they die.
15. Iguanas were falling out of the trees.

Words in red are mostly incomprehensible, the other red words would have extreme fluctuations within the same word or are just straight up wrong (short stop-words like the and to aren't well learned).

More training time will help the fluctuations and the model failing to learn letters

83 incorrectly pronounced words out of 185 words. 51.35% of words pronounced correctly.

1. He invested some skill in charisma and strength. Oove.
    a. Pronounced it cha-risma instead of ka-risma
    b. Strange noise at the end
2. I like to leave work after my eight-hour work break
    a. "to" is not well pronounced
3. Tomorrow will bring something new, so leave today as a memory
4. He had concluded that pigs must be able to fly in Hog heaven
    a. "Pigs" is pronounced as flights
    b. "y" is in "fly" is not treated as a vowel
    c. "Hog" is pronounced "he"
5. The white-water rafting trip was suddenly halted by an unexpected brick wall
6. He was willing to find the depths of the rabbit hole in order to be with her.
7. He's in a boy band which doesn't make much sense for a snake
    a. "snake" is pronounced "snae"
8. Wisdom is easily acquired when hiding under the bed with a saucepan on your head.
9. When he had to picnic on the beach, he purposefully put sand in other people's food.
10. The waitress was not amused when he ordered green eggs and ham
    a. 'ggs" is not pronunced in "eggs"
11. Three generations with six decades of life experience.
12. The ants enjoyed the barbecue more than the family.
13. She had some amazing news to share but nobody to share it with.
14. It's not possible to convince a monkey to give you a banana by promising it infinite bananas when they die.
15. Iguanas were falling out of the trees.
    a. Pronounced "Iguanas" like "Iginners"

10 incorrectly pronounced words out of 185 words. 94.59% of words pronounced correctly.

Results of Speech Accuracy Testing in week 7 and 12 respectively, red text indicates pronunciation errors

Appendix M: Benchmark Performance Testing

													Short = 10 characters
One fullstop											**Average**		Medium = 25 characters
Short sentence	25.662	25.194	25.182	25.331	26.223	26.245	26.816	26.016	26.329	25.142	25.814		Long = 50 characters
Medium sentence	25.599	26.009	26.243	25.676	25.657	25.925	25.399	25.81	25.785	25.202	25.7305		
Long Sentence	25.609	25.431	25.833	25.923	25.617	25.863	26.004	25.672	25.637	25.961	25.755		Unit is seconds
Two fullstops													
Short sentence	35.1	34.915	34.232	34.998	34.262	34.462	34.957	34.175	35.021	34.162	34.6284		
Medium sentence	34.124	34.95	34.982	35.019	34.126	34.992	34.669	35.112	34.744	34.562	34.728		
Long Sentence	34.591	35.356	34.83	35.273	34.963	34.87	35.928	35.554	35.224	34.863	35.1452		
One fullstop, one comma													
Short sentence	34.389	34.533	34.671	34.14	34.641	34.228	34.14	34.126	34.416	34.095	34.3379		
Medium sentence	34.939	34.184	34.577	34.622	34.43	34.664	34.268	34.873	34.263	34.465	34.5285		
Long Sentence	34.548	34.749	34.86	34.85	34.333	34.986	35.021	34.87	34.827	34.908	34.7952		
One fullstop, two commas													
Short sentence	43.155	43.354	43.653	42.927	43.121	43.121	42.998	43.376	43.03	42.674	43.1409		
Medium sentence	43.383	43.81	43.374	42.751	43.246	43.179	43.172	43.189	42.92	43.237	43.2261		
Long Sentence	42.847	43.226	43.611	43.49	43.116	43.541	43.207	43.501	43.195	43.46	43.3194		
Two Fullstops, two commas													
Short sentence	50.613	50.064	50.976	50.684	50.913	50.782	50.946	50.474	50.209	50.342	50.6003		
Medium sentence	50.623	50.829	50.487	50.973	51.846	51.536	51.215	50.602	50.978	51.091	51.018		
Long Sentence	50.598	51.139	50.802	51.066	51.07	51.314	50.632	51.198	51.259	50.967	51.0045		

Screenshot of performance testing results

Appendix N: Libraries, Models and GitHub demos

Deep Face Lab:
https://github.com/iperov/DeepFaceLab

Wav2Lip:
github.com/Rudrabha/Wav2Lip

Speech_Recognition:
https://pypi.org/project/SpeechRecognition/

Google's Wavenet Text-to-Speech:
https://cloud.google.com/text-to-speech

DCTTS:
https://github.com/Kyubyong/dc_tts

Demo of DC_TTS based from an existing implementation (not ours):
https://www.youtube.com/watch?v=NBOm5bPJJU4&t=549s

Tacotron2:
https://arxiv.org/abs/1712.05884

TensorflowTTS:
https://github.com/TensorSpeech/TensorFlowTTS

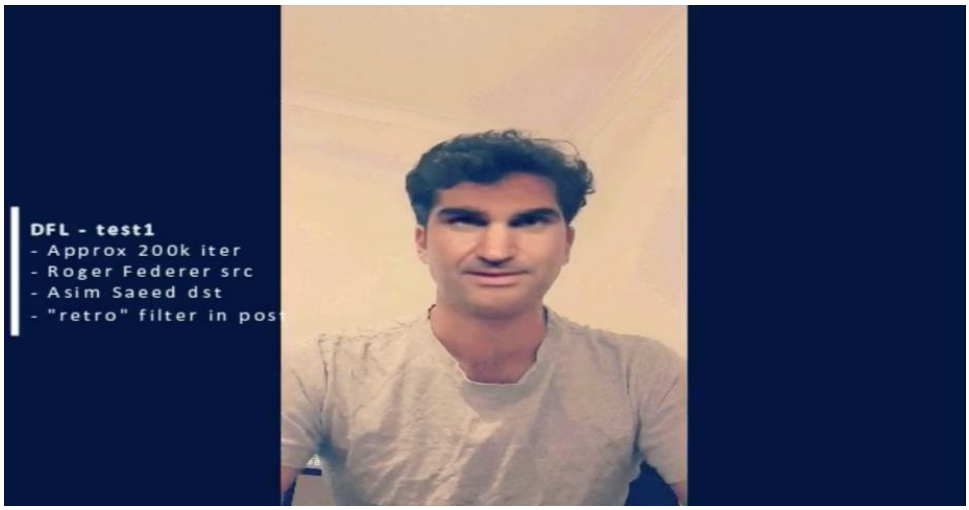
Demo of TensorflowTTS from an existing implementation (not ours):
https://tensorspeech.github.io/TensorFlowTTS/

Fastspeech2:
https://arxiv.org/abs/1905.09263

Dl-for-emo-tts (Emotional Tagging):
https://github.com/Emotional-Text-to-Speech/dl-for-emo-tts

Demo of Emotional Tagging:
https://colab.research.google.com/github/Emotional-Text-to-Speech/dl-for-emo-tts/blob/master/Demo_DL_Based_Emotional_TTS.ipynb#scrollTo=3jKM6GfzlgpS

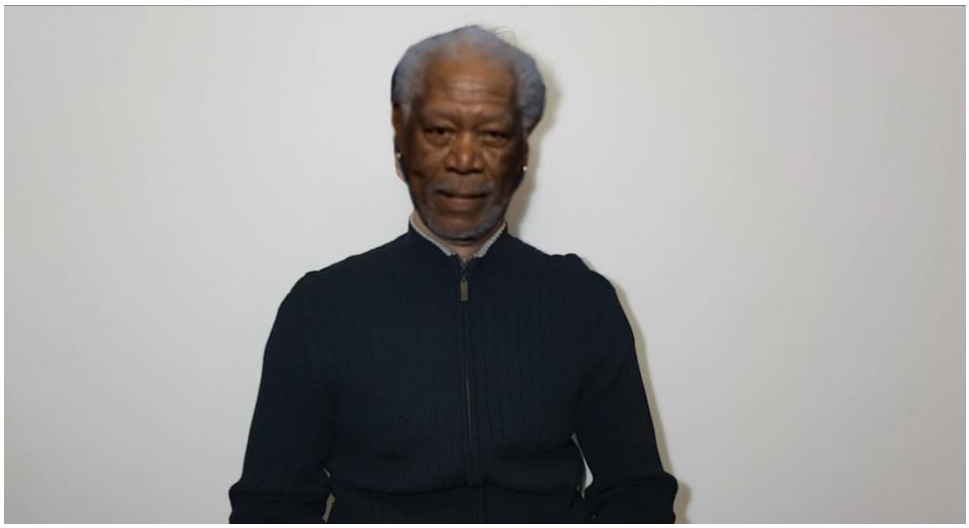
Appendix O: Deepfake Demos



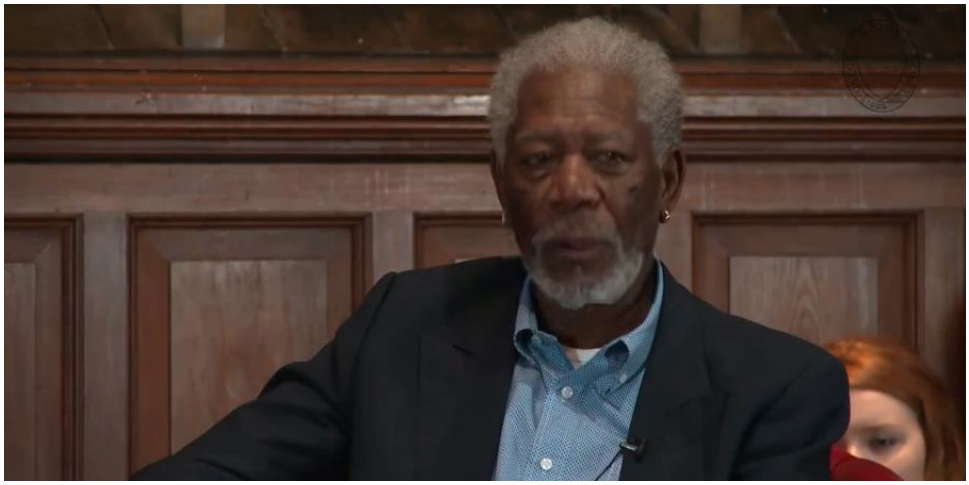
First Deep Face Lab demo achieved with full face, Quick96 training and 150k training iterations. Full video found at: https://unisydneyedu-my.sharepoint.com/:v:/g/personal/gste6115_uni_sydney_edu_au/EcBaUwhPOepBlImmNW78yVQBK4m0HiXslM1Eo-gUV0bjSQ?e=xk5Wev

Second Deep Face Lab demo achieved with full face, X-SEG Mask training, SAEHD training and 100k training iterations. Full video found at: https://unisydneyedu-my.sharepoint.com/:v:/g/personal/gste6115_uni_sydney_edu_au/ETy5oTdgUAROnL9z5oGEFacBbHzkvg6L8hc7jSrXt0oF-g?e=8tUHoQ

Third Deep Face Lab demo achieved with full head, X-SEG Mask training, SAEHD training, and 150k training iterations. Full video found at: https://unisydneyedu-my.sharepoint.com/:v:/g/personal/gste6115_uni_sydney_edu_au/EcD6emGPTCZDtRKrjI4q0m4Be2obPC3PJI3BDdK0Ig3W5Q?e=UVYoQo

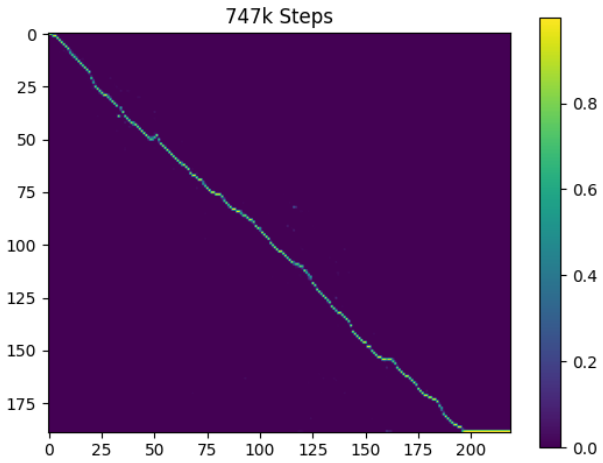


Fourth and final Deep Face Lab demo achieved with full head, X-SEG Mask training, SAEHD training, and 150k training iterations. Additional 15k training iterations with GAN power and learning rate dropout enabled. Full video found at: https://unisydneyedu-my.sharepoint.com/:v:/g/personal/gste6115_uni_sydney_edu_au/EVdlFZMU281Li_coGiuGkrkBQvy9kG4Tym56KD92zuAMow?e=VD6kNd

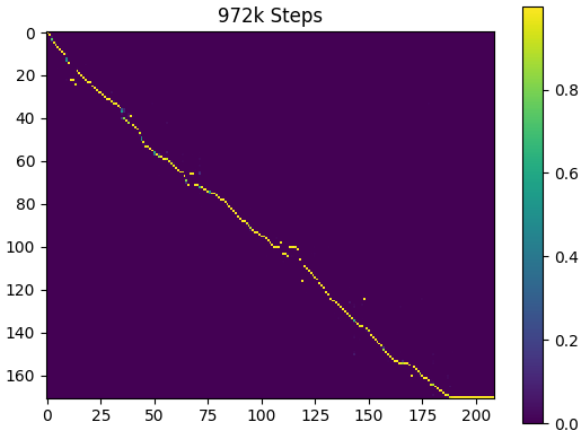


Folder of WAV2Lip demos: https://unisydneyedu-my.sharepoint.com/:f:/g/personal/gste6115_uni_sydney_edu_au/EuhYtOGYOvBCvQoZRdfE77cBuglPY5xpefmbF3qx5oOBcg?e=mPuE4z

Appendix P: DC_TTS Attention Plots

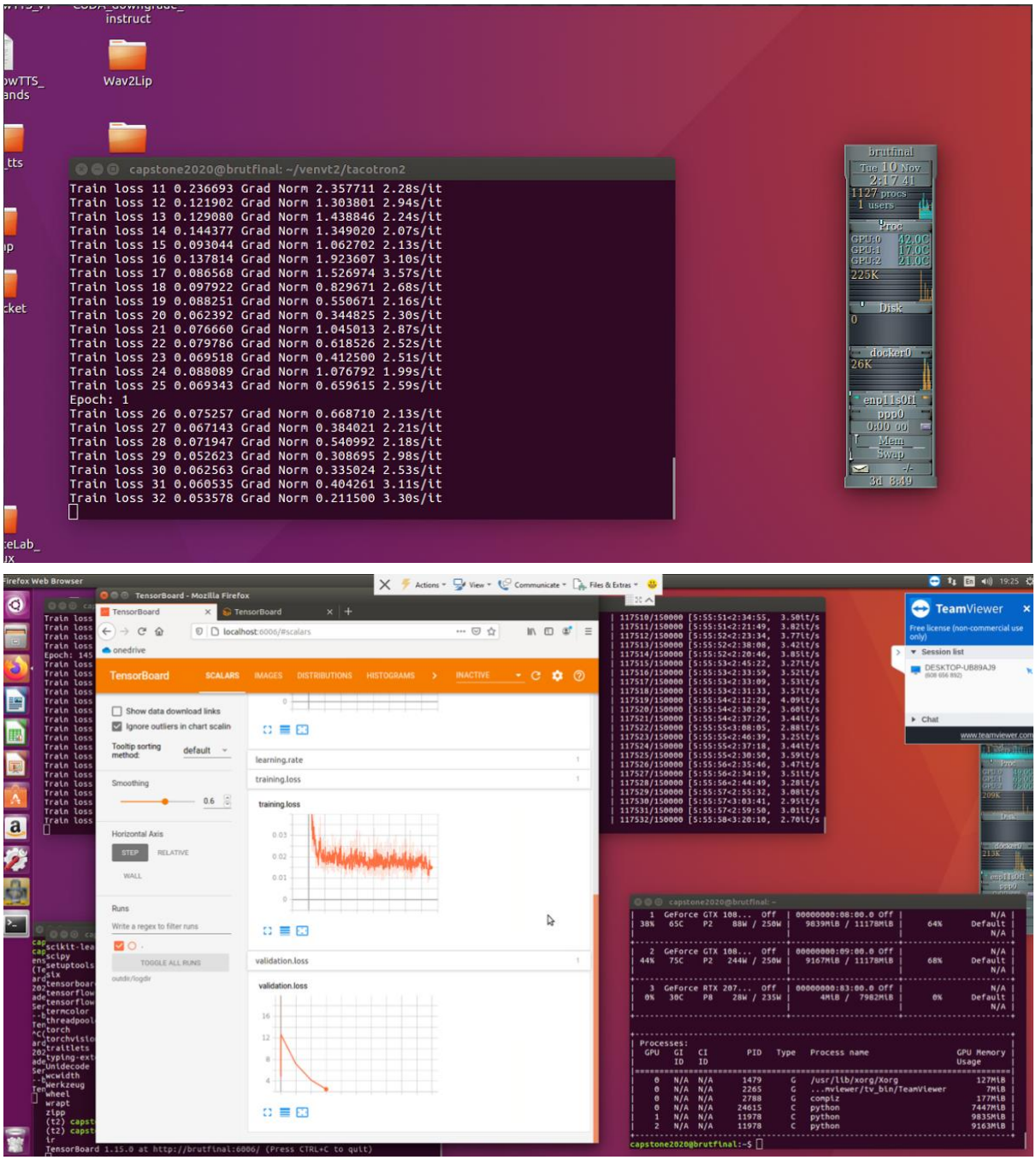


Screenshot of attention plot at 747k iteration steps



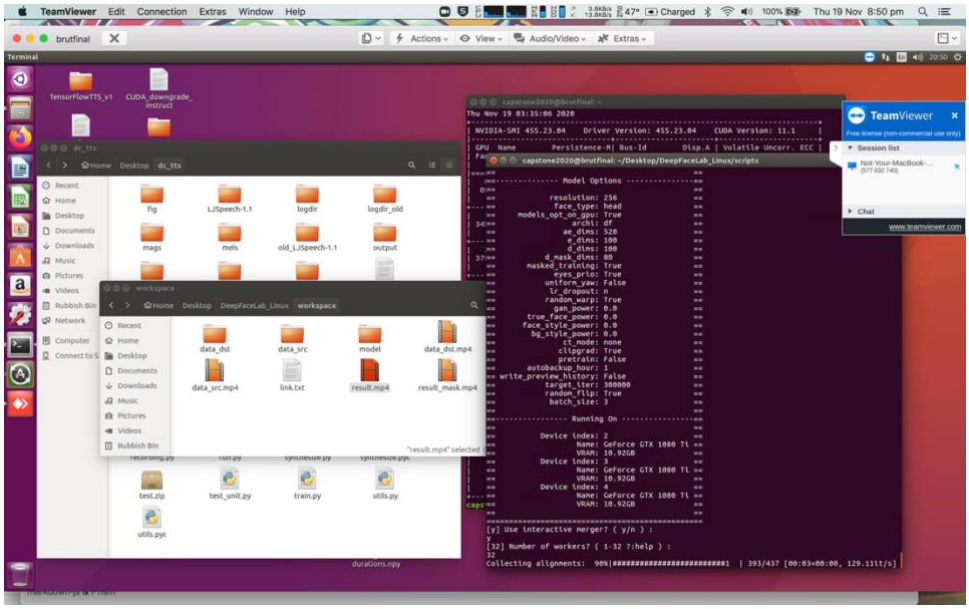
Screenshot of attention plot at 972k iteration steps showing clearly the additional training produced a worse plot than the 747K iteration (less straight)

Appendix Q: Training Tacotron2

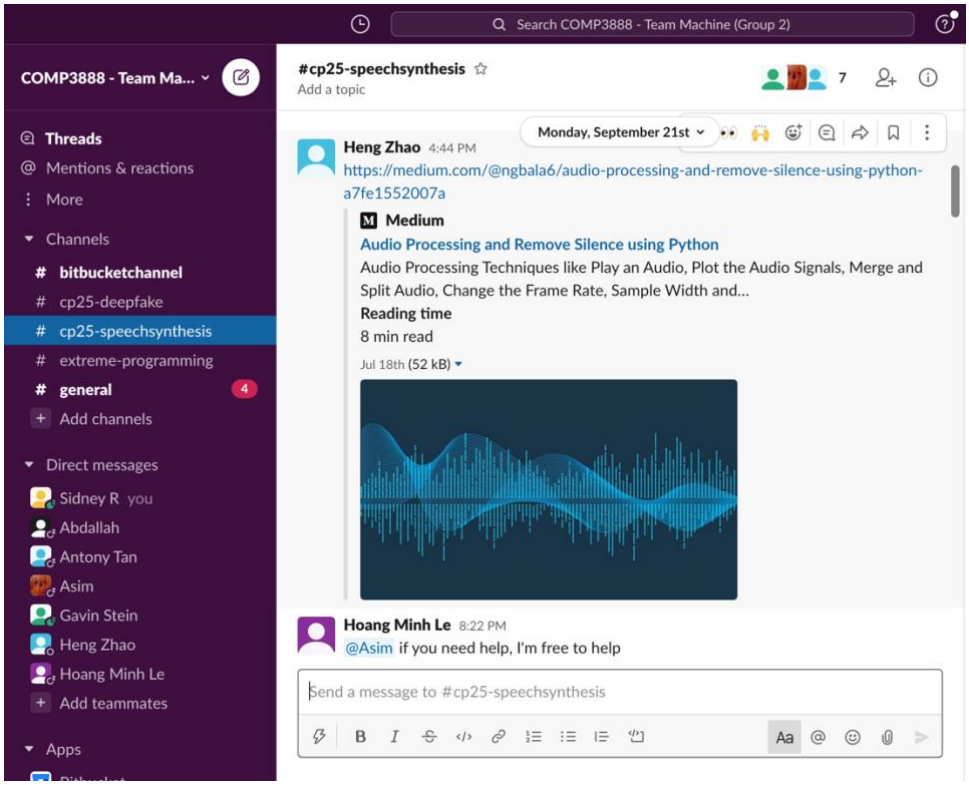


The trained RNN model for Tacotron2 can be found in this link.
https://unisydneyedu-my.sharepoint.com/:u:/g/personal/gste6115_uni_sydney_edu_au/ETMV30QuV4FCqmIIqQGsWQ0BzpJUfb4vK3_Z0C49PJxiuw?e=RPnUU2
The model is not included in the bitbucket wiki due to its size.

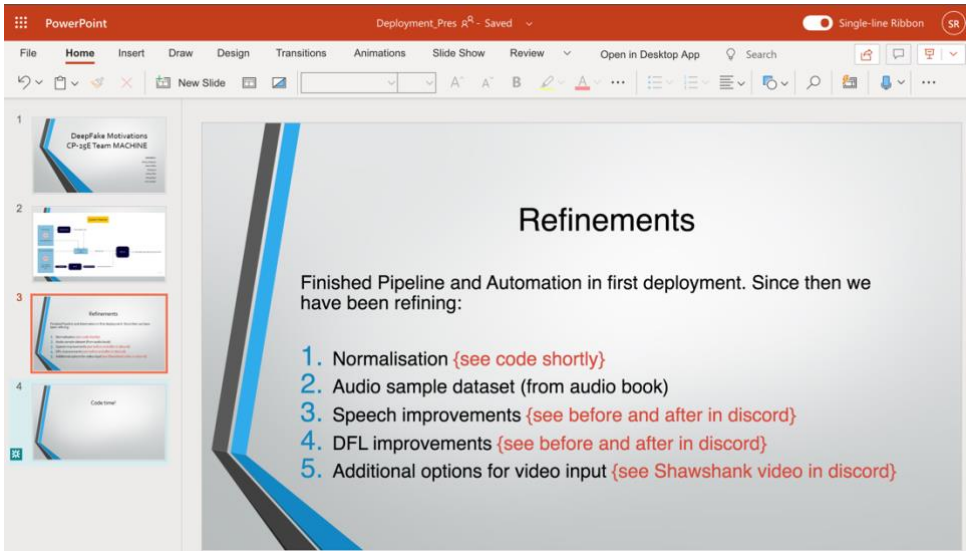
Appendix R: Group Processes Evidence



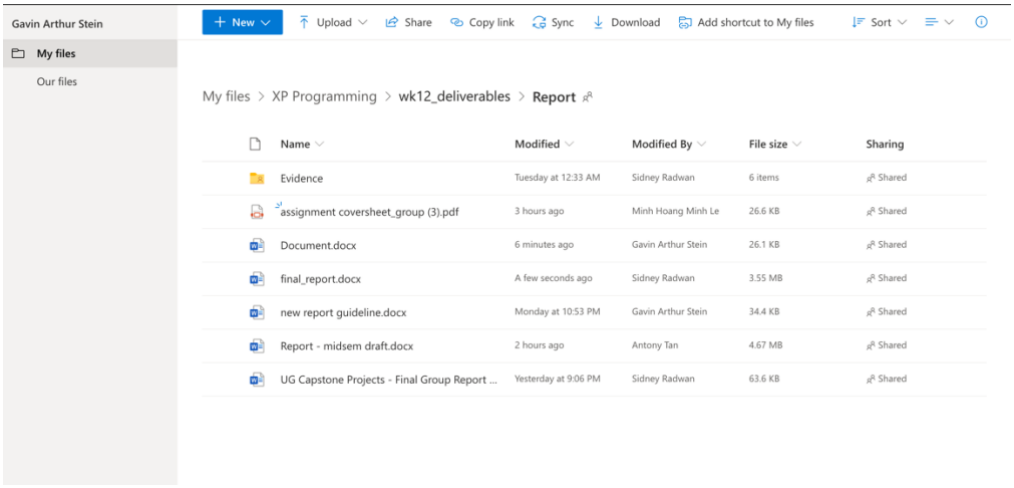
Evidence of client machine with multiple GPU's being used for development of Deep Face Lab model.



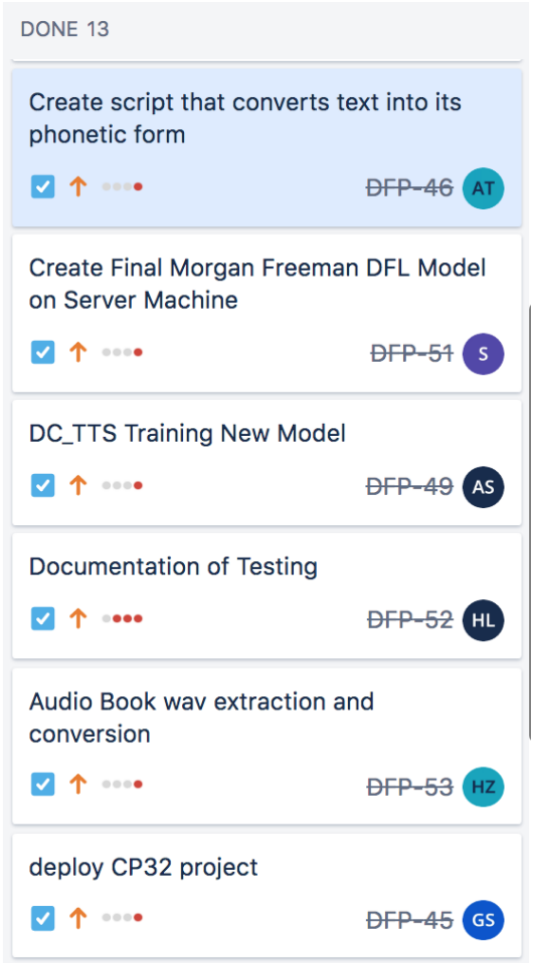
Evidence of Slack use with multiple channels for crucial project topics and research



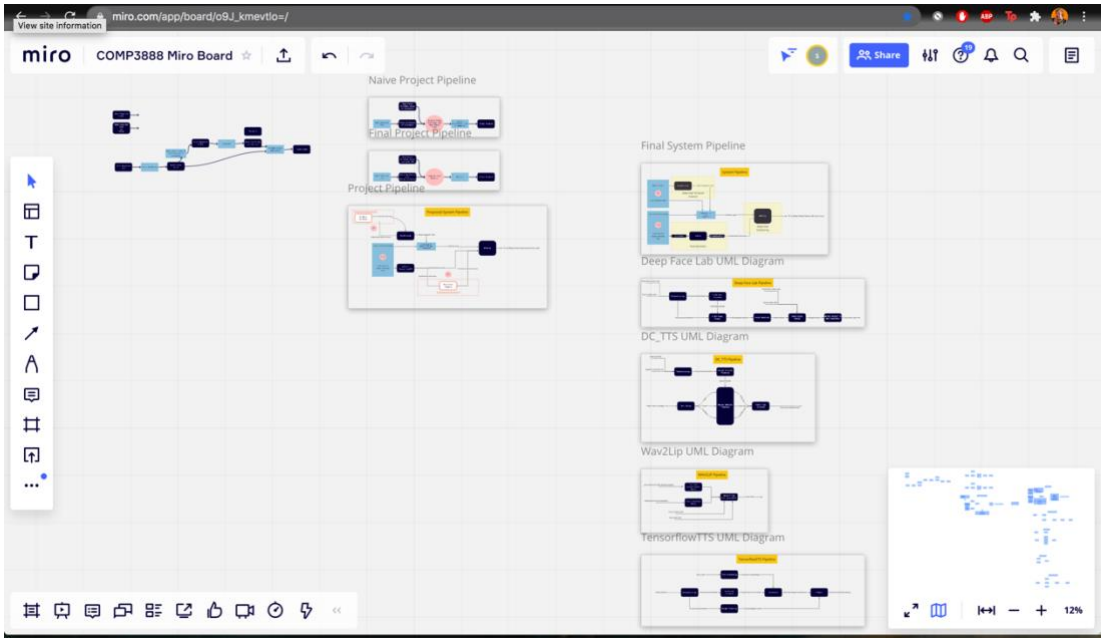
Evidence of Office 365 use for presentation to client



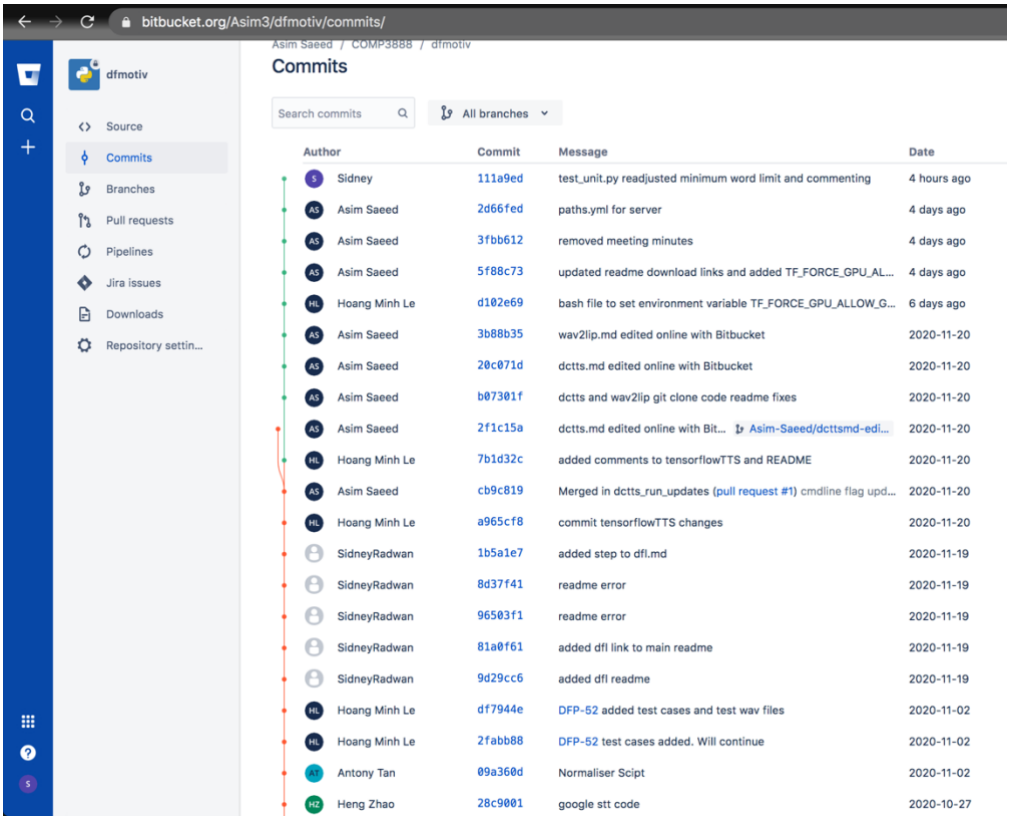
Evidence of OneDrive use for collaboration on shared documents



Evidence of Jira Kanban Board being used to equitably distribute work in week 8, practiced nearly perfectly every week.

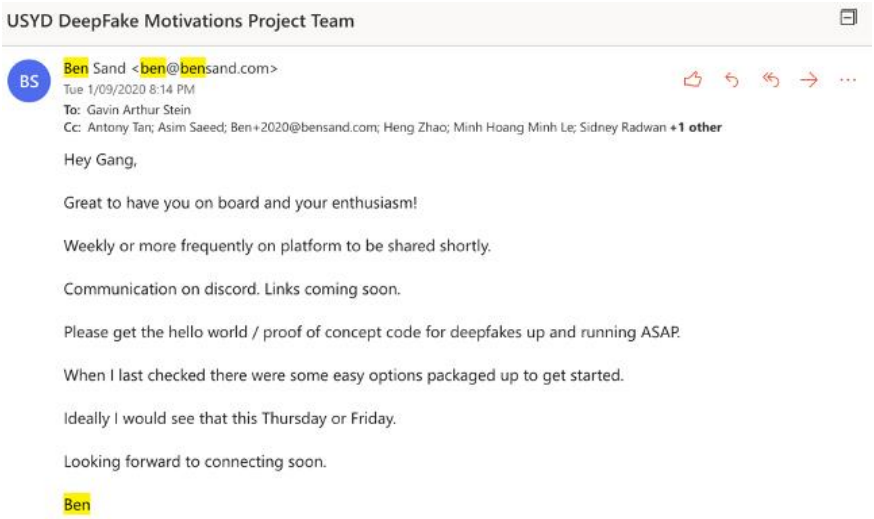


Screenshot of Miro board used to make all UML diagrams in this report. Link: https://miro.com/app/board/o9J_kmevtlo=/



Screenshot of Bitbucket utilisation and commits

Appendix S: Interaction with Client



USYD DeepFake Motivations Project Team

BS

Ben Sand <ben@bensand.com>
Tue 1/09/2020 8:14 PM
To: Gavin Arthur Stein
Cc: Antony Tan; Asim Saeed; Ben+2020@bensand.com; Heng Zhao; Minh Hoang Minh Le; Sidney Radwan **+1 other**

Hey Gang,

Great to have you on board and your enthusiasm!

Weekly or more frequently on platform to be shared shortly.

Communication on discord. Links coming soon.

Please get the hello world / proof of concept code for deepfakes up and running ASAP.

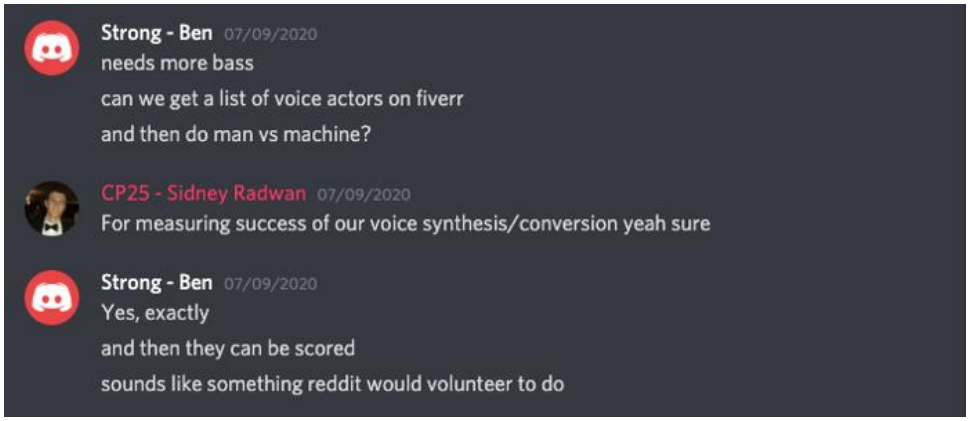
When I last checked there were some easy options packaged up to get started.

Ideally I would see that this Thursday or Friday.

Looking forward to connecting soon.

Ben

Screenshot of email with client

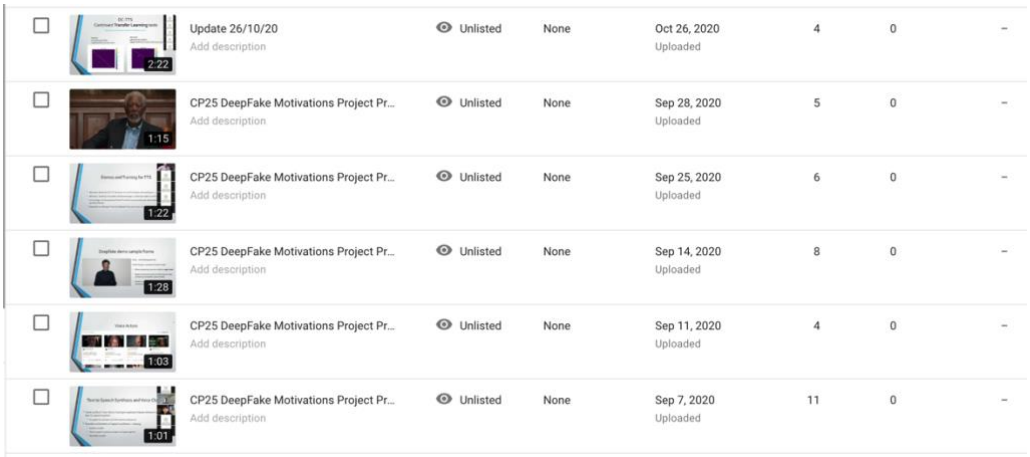


Strong - Ben 07/09/2020
needs more bass
can we get a list of voice actors on fiverr
and then do man vs machine?

CP25 - Sidney Radwan 07/09/2020
For measuring success of our voice synthesis/conversion yeah sure

Strong - Ben 07/09/2020
Yes, exactly
and then they can be scored
sounds like something reddit would volunteer to do

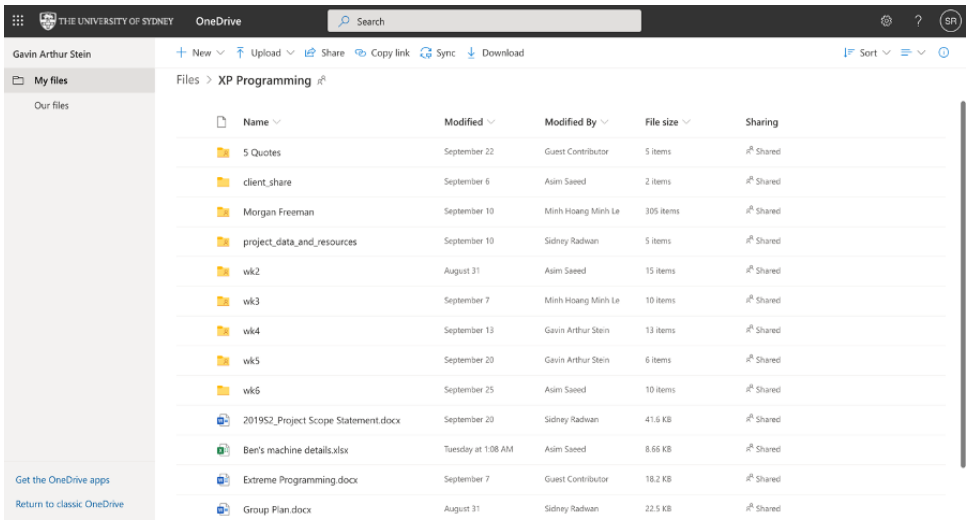
Screenshot of Discord with client



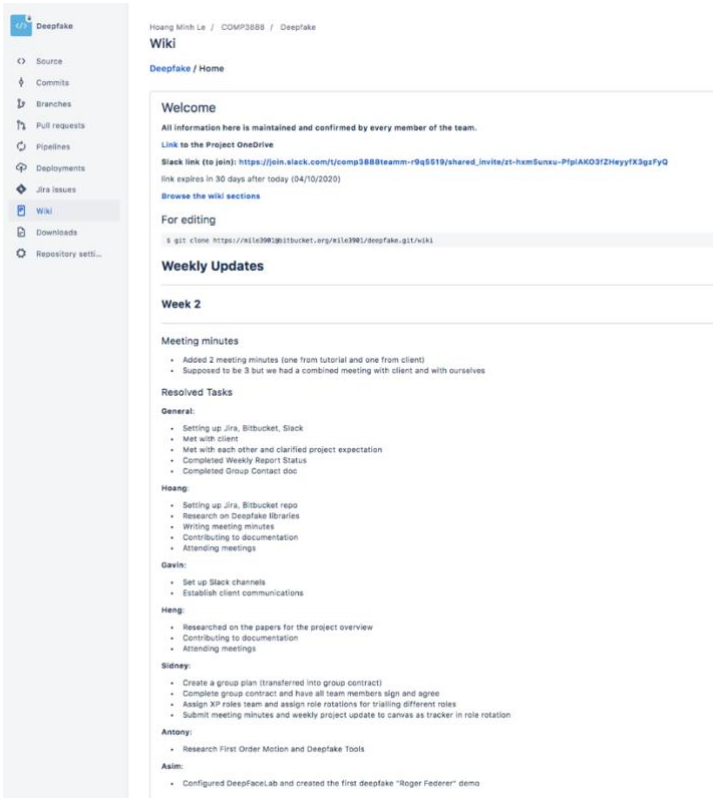
Update 26/10/20 (2:22)	Add description	Unlisted	None	Oct 26, 2020	Uploaded	4	0
CP25 DeepFake Motivations Project Pr... (1:15)	Add description	Unlisted	None	Sep 28, 2020	Uploaded	5	0
CP25 DeepFake Motivations Project Pr... (1:22)	Add description	Unlisted	None	Sep 25, 2020	Uploaded	6	0
CP25 DeepFake Motivations Project Pr... (1:28)	Add description	Unlisted	None	Sep 14, 2020	Uploaded	8	0
CP25 DeepFake Motivations Project Pr... (1:03)	Add description	Unlisted	None	Sep 11, 2020	Uploaded	4	0
CP25 DeepFake Motivations Project Pr... (1:01)	Add description	Unlisted	None	Sep 7, 2020	Uploaded	11	0

Screenshot of bi-weekly update videos prepared for client

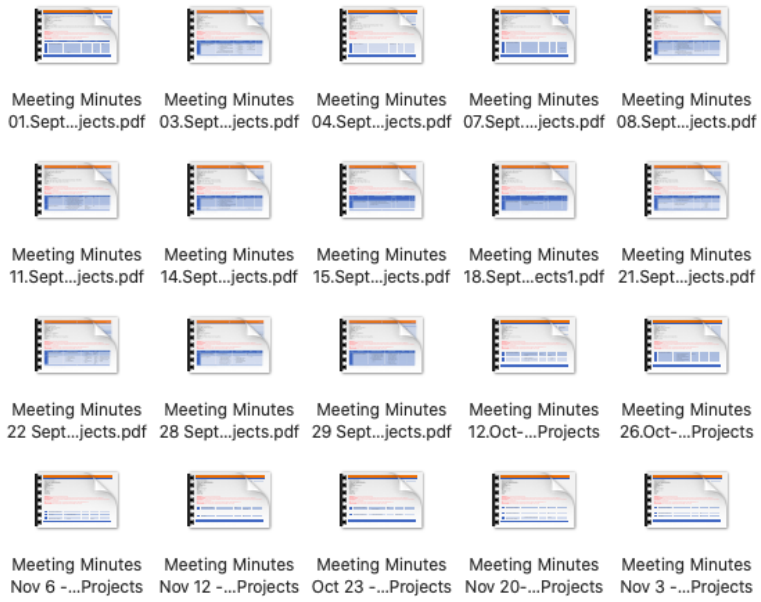
Appendix T: Documentation



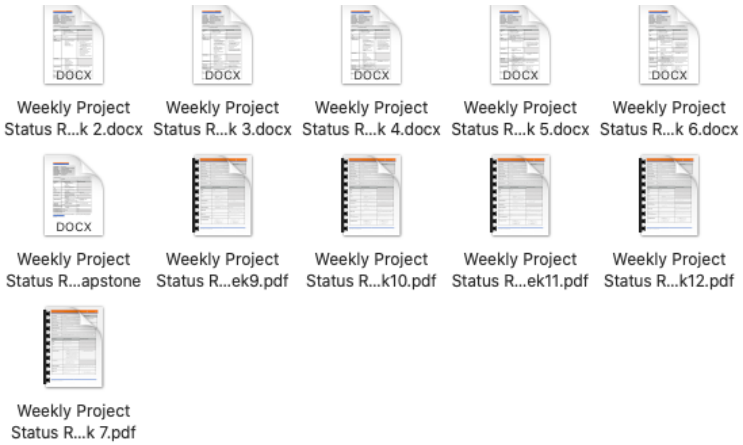
Screenshot of team OneDrive folder. Link : https://unisydneyedu-my.sharepoint.com/:f:/g/personal/gste6115_uni_sydney_edu_au/Evk34y29RzBCtL5NeciIYbAB5vaZ0AXLPOawqN4vIJWfiA?e=5hIWVP



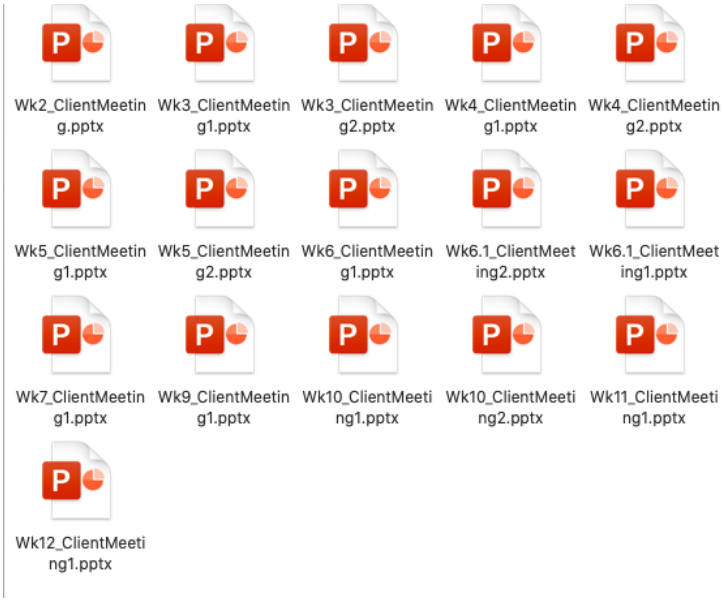
Screenshot is for reference only – for Bitbucket wiki page refer to wiki link in executive summary



Screenshot is for reference only – for meeting minutes refer to link: https://unisydneyedu-my.sharepoint.com/:f:/g/personal/gste6115_uni_sydney_edu_au/EhnjHuoUZTpBvOmCPppCmfIBnU2IkHU9vWFIK68v5mUigQ?e=5s58gk

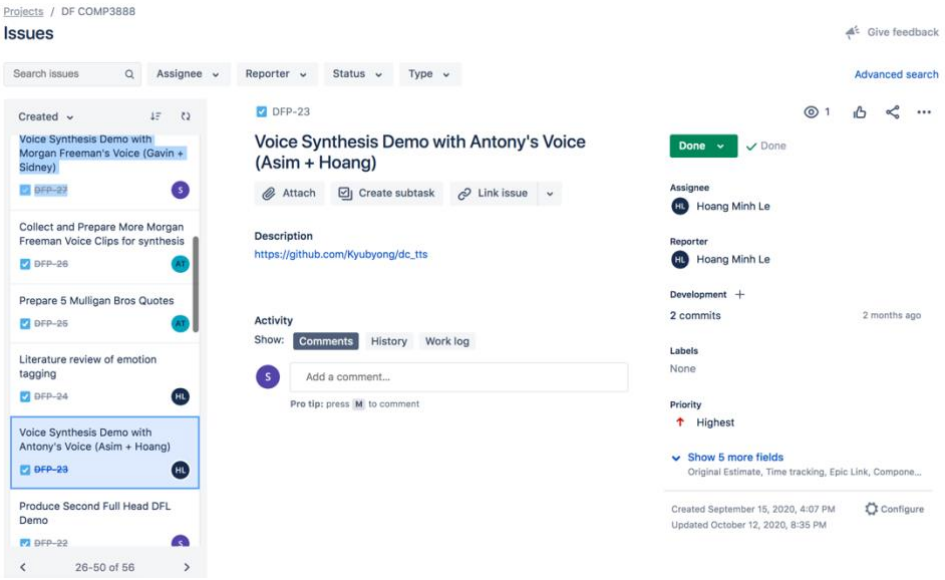


Screenshot is for reference only – for project status report refer to link: https://unisydneyedu-my.sharepoint.com/:f:/g/personal/gste6115_uni_sydney_edu_au/EtK2g49Sf4JJtLTqXk8-OjgBxUjsDwLUneyZy4wun-P_KA?e=YtDiNi

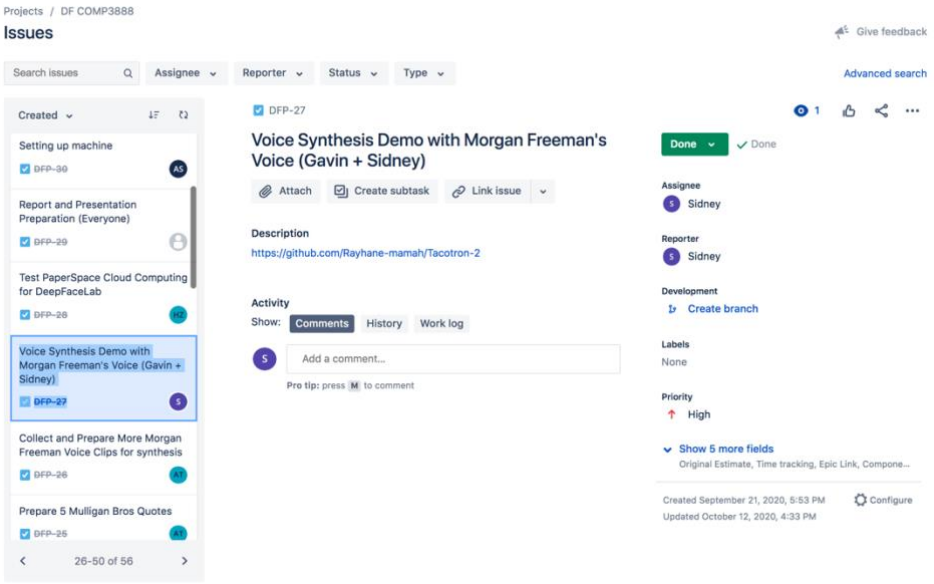


Screenshot is for reference only – for client meeting presentation refer to link: https://unisydneyedu-my.sharepoint.com/:f:/g/personal/gste6115_uni_sydney_edu_au/EhEvATTYeiVKg7bvAMibE8IB8kaeoXT22kM7MzzJGg0SxA?e=LVHoBf

Appendix U: Programming Pairs

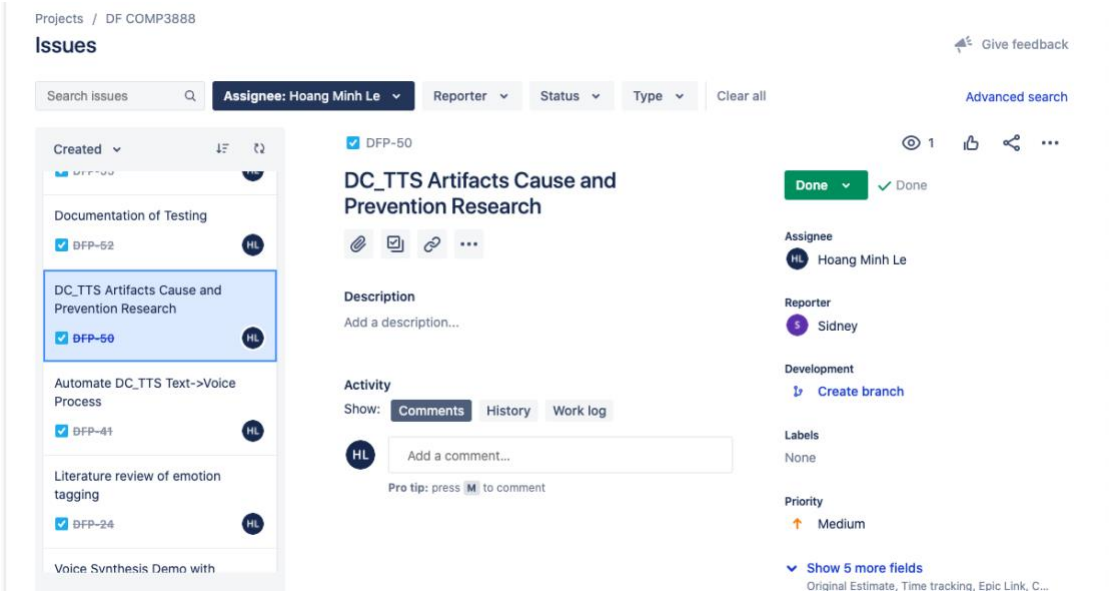


Screenshot of Jira showing programming pair for weekly task sprint (Asim + Hoang)



Screenshot of Jira showing programming pair for weekly task sprint (Gavin + Sidney)

Appendix V: DCTTS Noise Problem



Ticket for the problem

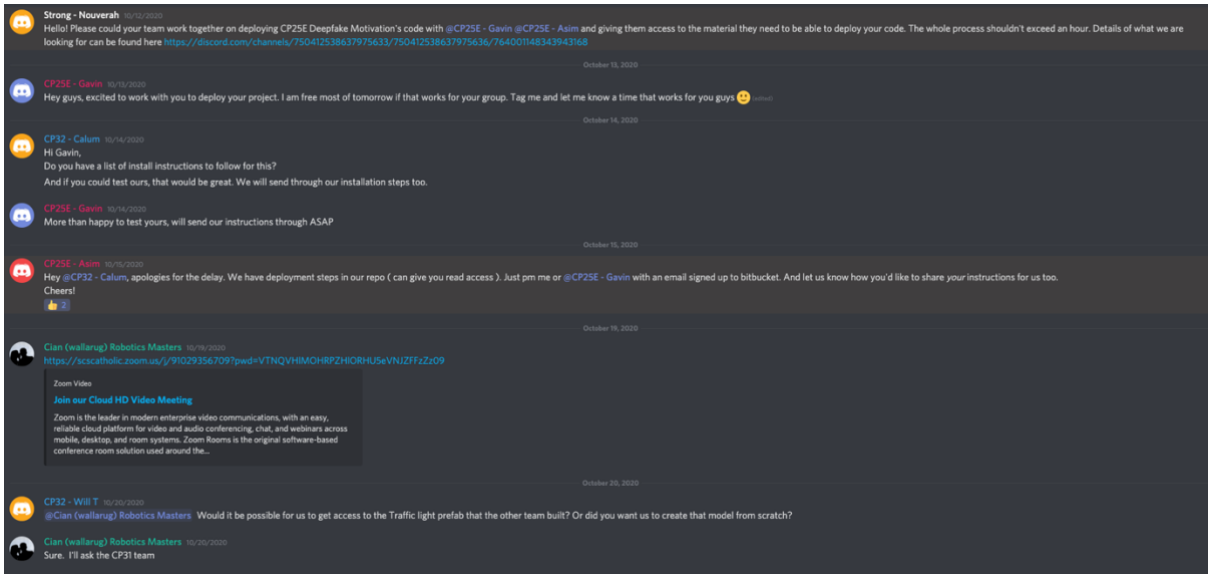
```
else: # synthesize on unseen test text.
    # Parse
    lines = codecs.open(hp.test_data, 'r', 'utf-8').readlines()
    sents = [text_normalize(line).strip() + "E" for line in lines] # text normalization, E: EOS
    texts = np.zeros((len(sents), hp.max_N), np.int32)
    for i, sent in enumerate(sents):
        texts[i, :len(sent)] = [char2idx[char] for char in sent]
    return texts
```

Issue tracked in ***/libraries/dctts/load_data.py*** file

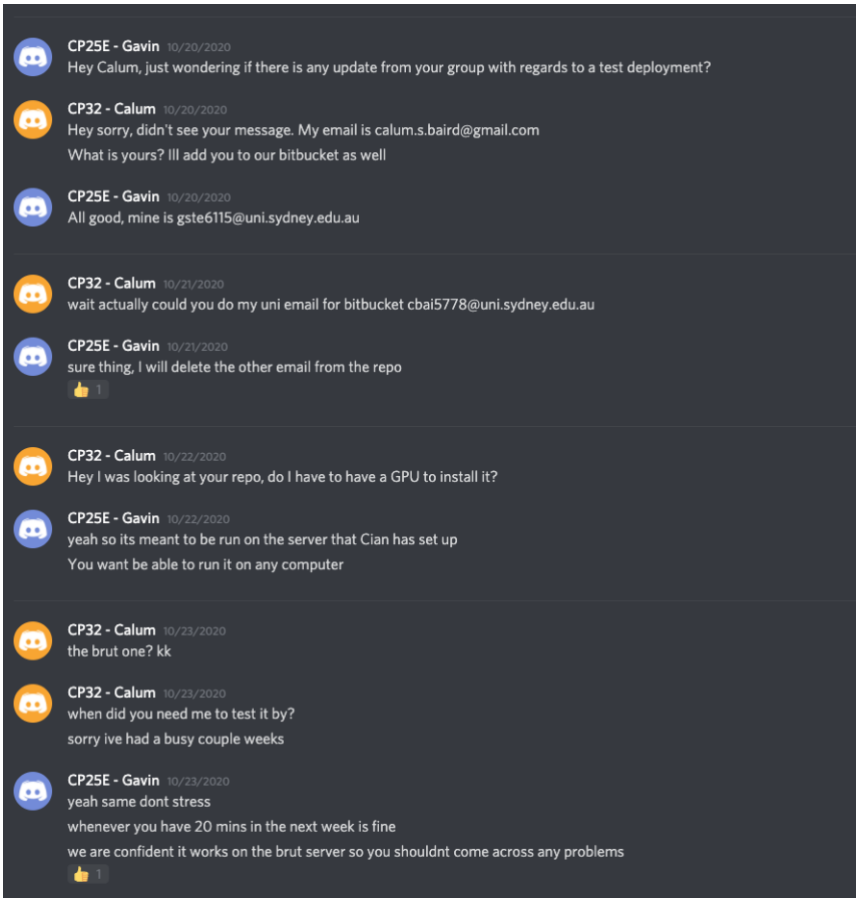
```
182 def format_wav_files():
183
184     lines = open(hp.test_data, "r").read().split("\n")
185
186     for i, line in enumerate(lines):
187         wav = AudioSegment.from_wav("./"+hp.sampledir+"/" + str(i+1) +".wav")
188         newdur = len(line) * 100 #Trim
189
190         if not newdur > len(wav):
191             # wav = wav[:- int(len(wav)/6)]
192             # else:
193             wav = wav[:newdur]
194
195         wav.export("./"+hp.sampledir+"/" + str(i+1) +".wav", format = "wav")
196
197
```

Solution to the problem. Refer to the Bitbucket repository ***libraries/dctts/run.py*** file

Appendix W: System Accessibility Testing issues



Client provided team (CP32) allocated to test our system's deployment and provide feedback (12/10/2020).



CP32 not responding with email for 8 days (20/10/2020), then after agreeing to install system disappears from contact (23/10/2020)

Appendix X: Main dfmotiv README

README.md

Welcome to Deepfake Motivations (CP25E)

This README contains deployment information for this project.

1. Clone this repo
2. Dependencies *(links for setting up these **packages** and the **virtual environments** they need to run in)*
    - dctts
    - wav2lip
3. Ensure the paths in **dfmotiv/paths.yml** point the relevant dirs.
4. Download DFL mp4 or shawshank clip mp4 as dfmotiv/**src/template.mp4**.

```
python3 dfmotiv.py -t "Hello World, this is Morgan Freeman"
```

voila, you've created a deepfake video. see dfmotiv/out/**output.mp4**

Making DeepFake templates

- DeepFaceLab

Other packages (not yet a requirement)

- Tensorflow_tts

Other flags

```
// Specify output path
python3 dfmotiv.py -t "This is Morgan Freeman" -o /home/asim/Videos/myDeepfake.mp4

// Export audio only
python3 dfmotiv.py -t "This is Morgan Freeman" -a
```

Main readme (blue links lead to more specific & detailed readme files deeper in repo).

Appendix Y: Assessment of CGI stand-ins

Assessment of introducing a CGI pipeline into our proposed Deepfake system

Here we discuss potential challenges in creating **CGI stand-ins** instead of **video captured stand-ins** for our Deepfake motivations system.

Creating 3D animation is a complex process, requiring a team of specialized technical artists to model, rig, texture, animate, and light/render. With limited resources and team expertise, we can try and source a readymade Morgan Freeman model or a readymade generic male face rig (to be completely deepfake faceswapped). But we likely won't luck upon a Morgan Freeman face rig (especially one ideally mimicking the actions of face muscle groupings identified by FACS; an industry standard in 3d face rigs).

Either the Morgan Freeman model will need rigging (no simple task) or the generic male face rig needs re-texturing + lighting to help blend the eventual deepfake faceswap. Moreover, such rigs will have very fine controls. So, a talented animator will need several hours or days to setup an array of different expression poses.

Realistic face animation requires more nuance than just hitting facial expression targets in a uniform linear way (the likely outcome of applying expression poses programmatically/procedurally from emotion tags in user supplied transcript text). An animator or mocap data needed for detailed subtlety, even with the lip-sync taken care of by Wav2Lip.

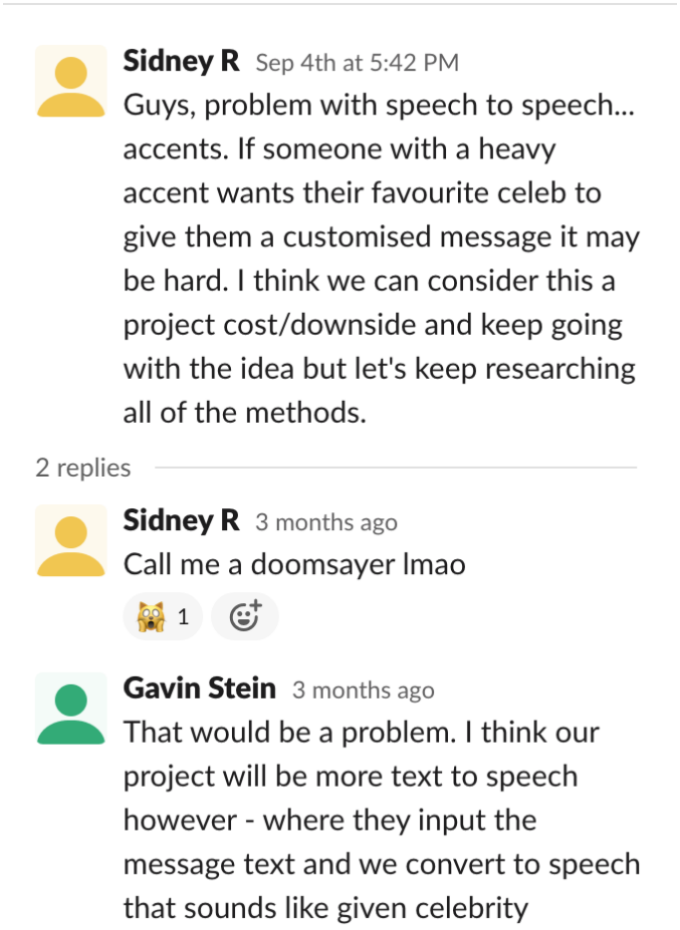
While, there are apps out there that democratise facial mocap using phone cameras, it will require extensive setup to work with a generic face rig as well as work to integrate into our eventual system. Finally, though a CGI stand-in will offer customization to the user, it will cost rendering time plus DFL faceswap computation time.

As you can see, **without an *off-the-shelf* solution**, introducing a 3d animation / mocap pipeline will **blow out the scope and what's achievable for our team in the remaining 7 weeks**. It may be ideal for our team to focus on improving deepfakes with simply captured video stand-ins and leaving the pipeline open for you to introduce a 3D animation system later down the track.

If more control of expressions and movement Is needed (to better fit the voice transcript) and speed is an issue i.e. this system is servicing a phone app, it may even be worth another look at **First-Order-Motion** in place of DeepFaceLab. The user must supply video rather than choosing from a template.

See the following link to full assessment of CGI stand-ins.pdf

Appendix Z: Doomsaying



Screenshot of effective Doomsaying