National College of Ireland

Configuration Manual

MSc Research Project
MSc Artificial Intelligence

Abhishek Gupte
Student ID: 22183604

School of Computing
National College of Ireland

Supervisor: Anu Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Abhishek Gupte
Student ID:	22183604
Programme:	MSc in Artificial Intelligence
Year:	2025-2026
Module:	MSc Research Project
Supervisor:	Anu Sahni
Submission Due Date:	23/12/2025
Project Title:	Configuration Manual
Word Count:	1553
Page Count:	14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	30th December 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Abhishek Gupte
22183604

1 Purpose & Description

This work shall describe the purpose & structure of the configuration manual. It's a guidebook to the installation & usage of the solution artefact associated with my MS thesis topic on 'Investigating SOTA Deepfake Detection frameworks in content-driven multimodal deepfake content'. It describes key stages used in the design of the project, information on dataset structures, installations, coding-environment setup. It aims to replicate an end-to-end flow of the project.

2 System Requirements

The section will map the 3-stage project execution with respect to the environment used and associated system requirements. The project pipeline was executed in three main stages -

2.1 Scripting & Dataset Analysis

This was the first stage in building the project. The coding was done in the Pycharm IDE Community Edition IDE .[1] on a Macbook M1 Pro. The specifications of the Macbook are as given -

1. **Operating System -** Sonoma 14.6.1
2. **RAM -** 16 GB
3. **Storage Disk -** - 512 GB
4. **GPU -** ARM M1 processor

2.1.1 Storage requirements

The Mac system uses three datasets for training & testing - (a) DFDC (b) LAV-DF (c) AVSpeech. The dataset was first downloaded in .mp4 format and the audios were extracted in .wav format.

- AVSpeech - A combined 25k file-set reqiuring a total of 9.5GB of free space
- DFDC - The combined audio and video files require 7GB of free space, after sampling

[1] Available at Pycharm website

- LAV-DF (train files) require a combined 6GB of free space after sampling & storing a combined 30k files. Additionally 500MB is needed to store and sample LAV-DF test files

2.2 Data preprocessing

The second stage consisting of processing audio and video clips, extracting individual metadata and saving to disk for offline training.

2.2.1 System Requirements

- **Operating System -** Linux 22.04 LT
- **GPU Stack -** 40 GB 8xA100
- **System Storage -** NFS
- **CPU cores -** 124 vCPUs
- **RA M-** 1000GB
- **Storage Disk -** 6TB

2.2.2 Storage requirements

The process involves face detection, clipping and storage of video segments.

The dataset was first downloaded in .mp4 format and the audios were extracted in .wav format. For audio the process involves reading from the video file metadata & extracting audio tensors and storing on disk.

- AVSpeech - There are a total of 12,754 video clips used. The audio track is extracted per video. For video face crops the files are stored in .mp4 format, while for audiothe extracted log-mel tensors are stored in torch friendly .pt format. The files total $\tilde{2}9$ GB.
- DFDC - There are a total of 1475 training segments saved for each of audio and video. The audio track is extracted per video. For video face crops the files are stored in .mp4 format, while for audio the extracted log-mel tensors are stored in torch friendly .pt format. The files total $\tilde{1}$ GB.
- LAV-DF (train files + test files) - The same process is repeated for LAV-DF. There are a total of 39k for train & approximately 4000 segments for test.

2.3 Training Environment

The Lambda Labs training instance (link provided in footnote) was used for this stage. This is a multi-gpu setup utilising 8 A100 GPUs.

2.3.1 System Requirements

- **Operating System -** Linux 22.04 LT
- **GPU Stack -** 80 GB 8xA100
- **System Storage -** NFS
- **CPU cores -** 240 vCPUs
- **RA M-** 1000GB
- **Storage Disk -** 20TB

2.3.2 Storage requirements

The process involves transferring saved video crops and audio tensors from stage 2 into the new system for training. T For audio the process involves reading from the video file metadata & extracting audio tensors and storing on disk. The dataset was first downloaded in .mp4 format and the audios were extracted in .wav format.

- AVSpeech (Ephrat et al.; 2018)- There are a total of $\tilde{1}3$ videos with a combined (audio + video) size of $\tilde{9}.5$ GB.
- DFDC (Brian Dolhansky; 2020; Dolhansky et al.; 2020) - The combined audio and video files require 7GB of free space, after sampling.
- LAV-DF (Cai et al.; 2023) (train files) require a combined 6GB of free space after sampling & storing a combined 30k files. Additionally 500MB is needed to store and sample LAV-DF test files.

2.4 External storage requirements

Alongside the mentioned storage drives, a Google Drive FOOTNOTE storage space of approximately 50 GB is needed to store the interim files. The files are processed, uploaded to google drive and downloaded to the Lambda system.

2.5 Additional system requirements

All the stages require an up-to-date installation and connection with Github. The repo is constantly pushed from the Mac system and pulled into the Lambda systems.

3 Setting up the environment

3.1 Phase 1

The PyCharm Community edition was used to code the scripts. This involves official conda installation for the ARM built from the official website, along with key pytorch, video decoding and data analysis libraries. A requirement file was generated. The main repos have been listed.

Key installations -

- Pytorch
- Pandas
- pytorch-lightning
- jupyter-notebook
- matplotlib
- tensorboard
- pyAV
- mmcv; mmaction

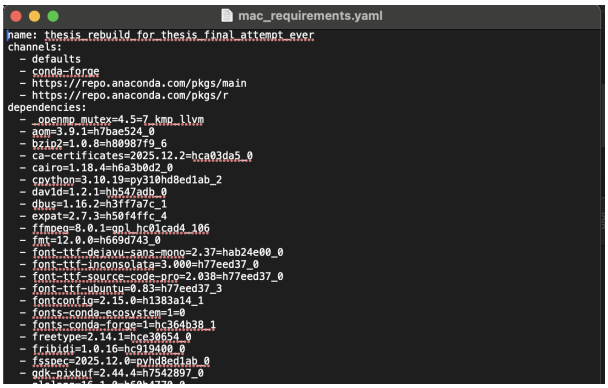

Figure 1: Mac requirements.yaml file

3.2 Phase 2

The libraries required an upgrade, installation of conda environment using pip3 and the git commands to download the project repo. Alongwith - the instructions in Phase 1 to setup the envorinment. The requirements.yaml sample has been showm.

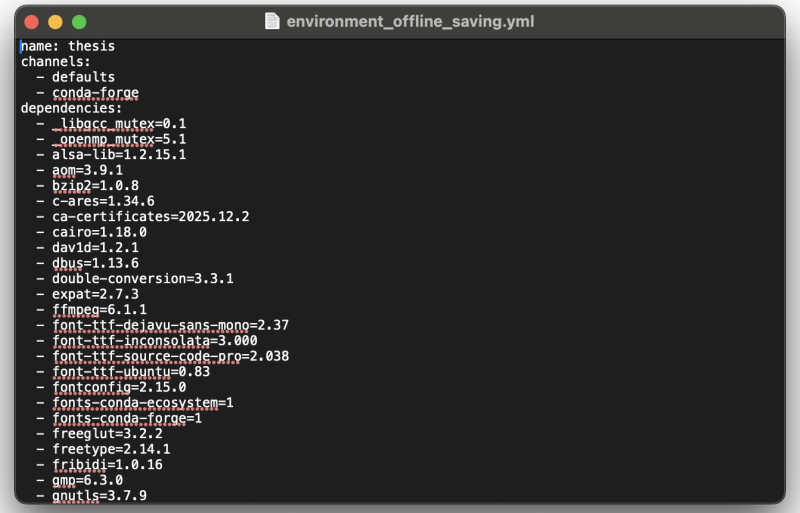

Figure 2: requirements.yml for offline saver

Key installations -

- Pytorch
- pytorch-lightning (latest release)
- tensorboard (for logging)
- pyAV (for video decoding)
- insightface (for face detection)
- mmcv; mmaction (for Swin feature extraction)

3.3 Phase 3

Identical setup to the phase 2. The libraries required an upgrade, installation of conda environment using pip3 and the git commands to download the project repo. Alongwith - the instructions in Phase 1 to setup the envorinment. The requirements.yaml sample has been showm.

Key installations -

- Pytorch
- lightning (latest release)
- tensorboard (for logging)
- pyAV (for video decoding)
- insightface (for face detection)
- mmcv; mmaction (for Swin feature extraction)

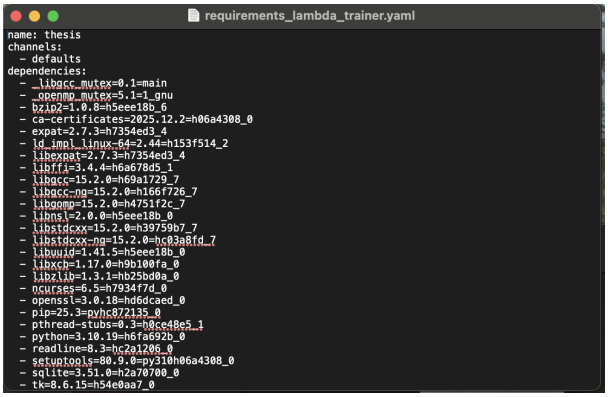

Figure 3: offline training requirements.yaml

4 Project Structure

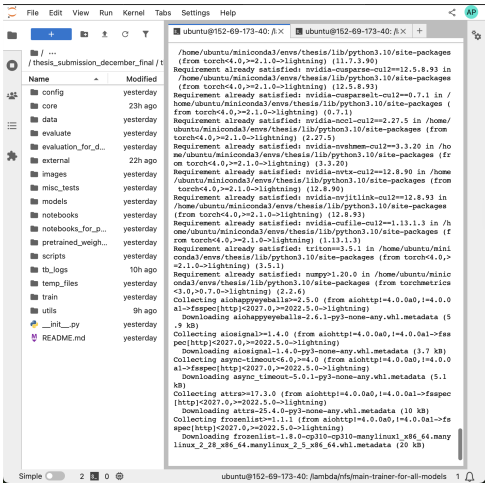

Figure 4: Project structure on Lambda AI

The images below show the project structure on the Lambda AI cloud service and on the local Macbook.

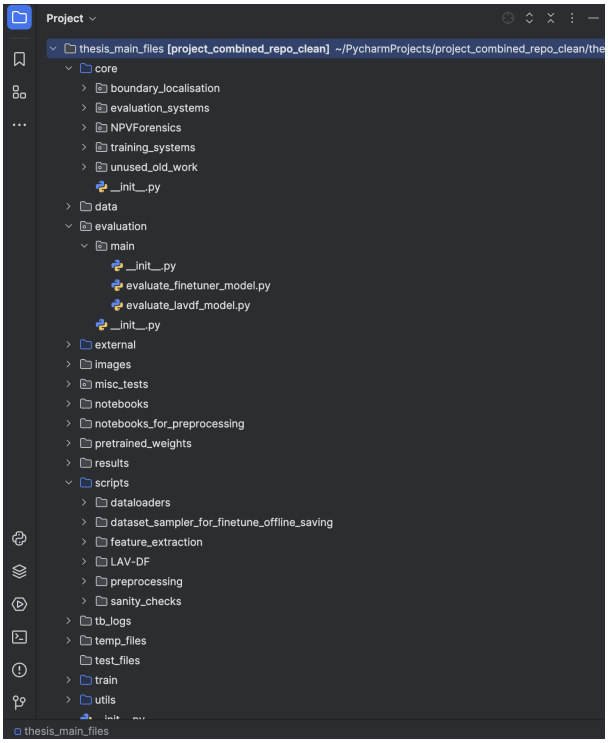

Figure 5: Mac project structure - I

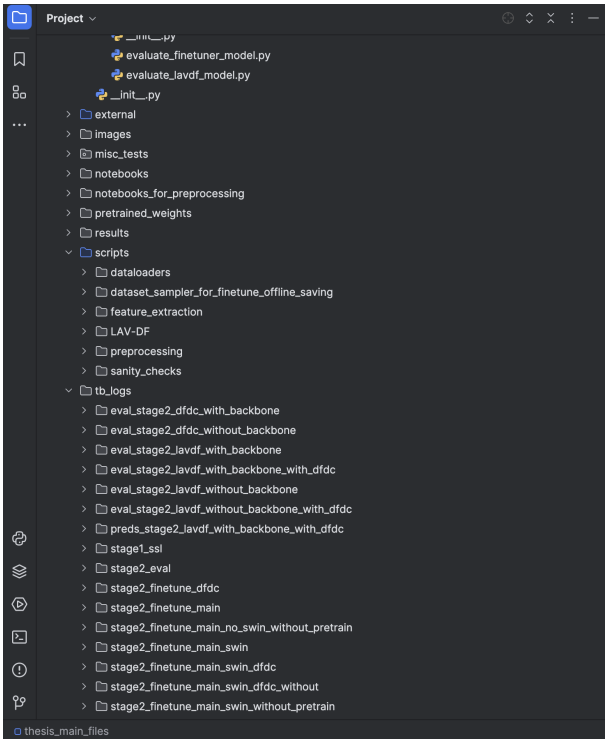

Figure 6: Mac project structure - II

The project files consist mainly of -

1. **core** - The storehouse of the core module sripts involved in the architcture - VACL block, training architecture, system training files, common projection block, BA-TFD+ model builder.

2. **data** contains the csv and video files for train and val

3. **pretrained_weights** houses the pretrained ba_tfdplus default model from the official github CITE.

4. **scripts** contains the feature extraction blocks, including swin model builds; The data preprocessors, the dataloader, the LAV-DF wrapper and various sanity checks

5. **external** houses the LAV-DF CITE and the Video Swin Transformer CITE, and Swin Transformer official repositories

6. **tb_logs** is a directory used for storing the tensorboard logs for the stage 1 and stage 2 models downloaded from the lambda offline saving/trainer systems.

5 Main execution

This section describes an execution flow and enlists scripts/installations needed. It is divided into the 3 phases described earlier.

5.1 Phase 1

1. Following scripting changes on the Mac system, the changes are pushed to github. A git instance is prior initialised after cloning the repo using the command "git clone...." and connecing using authentication tokens

5.2 Phase 2

1. A lambda instance is created as shown in figure.

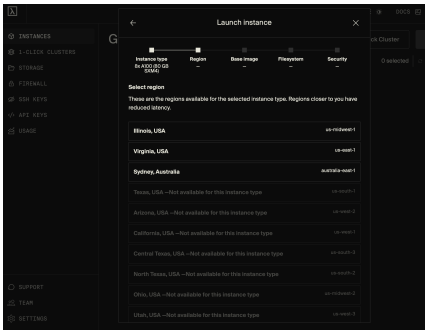

Figure 7: Lambda AI system launch

2. The system is installed with latest packages using "sudo apt-get.."
A mini-conda version is installed, along with creating new environment and installation of the files cited in the requirements file.

3. The datapreprocessor file **"offline_trainer_video.py"**is executed with the command as shown to create video file segments, stored in .mp4 format. The is done by leveraging the 8-GPU setup. The script creates a dataloader type instance to distrubute the video clip shards to 8 GPUs for seamless and streamlined execution. It is saved into an offine root with a batch name for reproducibility.

```
python offline_trainer_video.py \
  --video-root proactiveworkfilesystem/
thesis_submission_december_final/thesis_main_files/data/
processed/video_files/AVSpeech/video \
  --timestamps-json proactiveworkfilesystem/
thesis_submission_december_final/thesis_main_files/data/
processed/AVSpeech/
AVSpeech_timestamp_json_for_offline_training/
AVSpeech_timestamp_json_for_offline_training.json \
  --batch-name avspeech_video_stage1 \
  --devices 8
```

Figure 8: offline_trainer_video.py execution script to create video segments

4. The same process is repeated to save audio tensors from **offine_audio_trainer.py** file script.

```
PYTHONPATH="$PWD/thesis_main_files" \
python thesis_main_files/scripts/preprocessing/main/
offline_trainer_audio.py \
  --audio-root thesis_main_files/data/processed/audio_files/
AVSpeech/audio \
  --offline-root thesis_main_files/scripts/preprocessing/main/data/
processed/AVSpeech/AVSpeech_offline_training_files \
  --batch-name avspeech_video_stage1 \
  --devices 8
```

Figure 9: Offline audio trainer script execution

   The script is executed with the PWD command to ensure import correctness within the script environment.

5. The python scripts are executed and the GPUs are monitored to ensure the scripts are working. The generated video segment clips and audio tensors are downloaded to the Mac and uploaded to google drive.

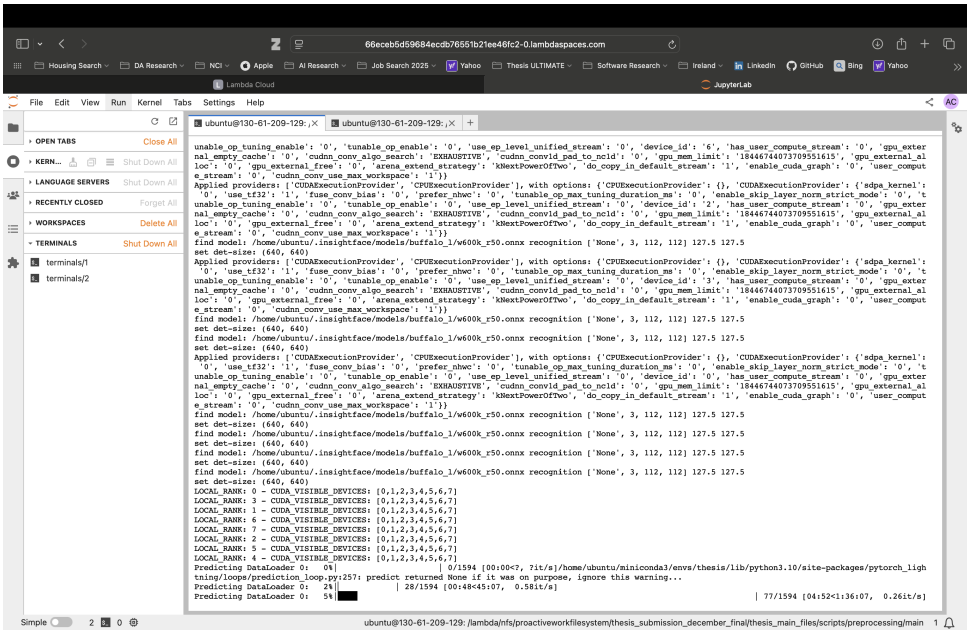

Figure 10: Video preprocessing execution

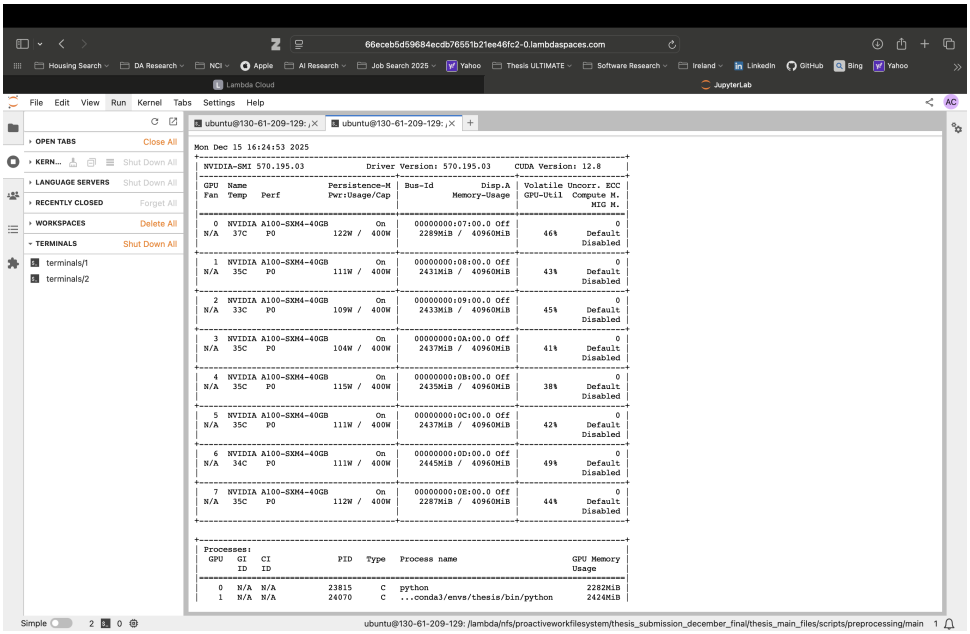

Figure 11: Nvidia GPU usage - video preprocessor

5.3 Phase 3

This phase explains an end-to-end example of script usage in training a model.

1. A lambda instance is created as shown in figure.

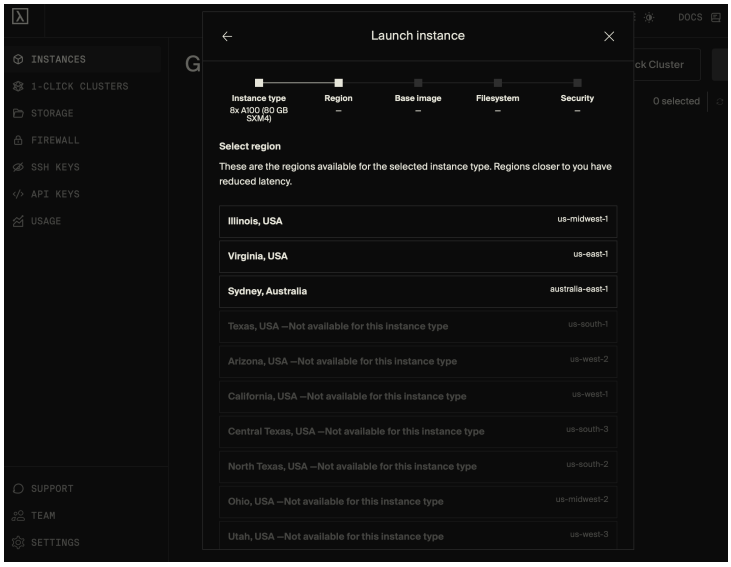

Figure 12: Lambda AI system launch

2. The new 80 GB system is re-installed with latest packages using "sudo apt-get.." A mini-conda version is installed, along with creating new environment and installation of the files cited in the requirements file.

3. The python script is executed using for stage 1 training with -

```
cd /lambda/nfs/main-trainer-for-all-models/
thesis_submission_december_final/thesis_main_files

PYTHONPATH="$PWD" \
python train/main/main_trainer_pretrain.py \
  --offline-root data/processed/AVSpeech/
AVSpeech_offline_training_files \
  --batch-name avspeech_video_stage1 \
  --batch-size 4\
  --devices 8 \
  --max-epochs 3 \
  --precision bf16-mixed \
  --accumulate-grad-batches 1 \
  --mem-log-every 200
--enable-energy-tracking
--enable-flops-profile
```

Figure 13: Python script execution stage1 train

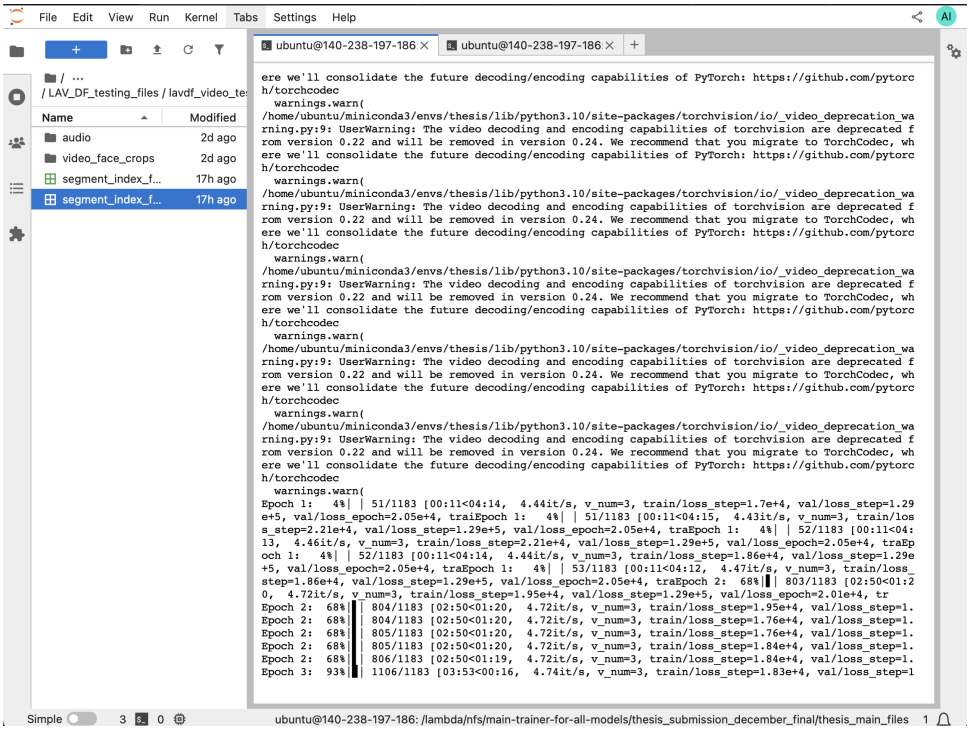

Figure 14: Sample training run

4. The script command is shown for stage 2 command -

```
PYTHONPATH="$PWD" \
python evaluation/main/evaluate_finetuner_model.py \
  --evaluation-root data/processed/LAV_DF/LAV_DF_testing_files/
LAV_DF_testing_files \
  --batch-name lavdf_video_test_stage1 \
  --ckpt tb_logs/stage2_finetune_main_without_swin_dfdc/version_1/
checkpoints/last.ckpt \
  --mode validate \
  --save-preds-path outputs/dfdc_on_lavdf/
preds_stage2_lavdf_without_backbone_with_dfdc.jsonl
```

Figure 15: Python script execution stage 2

5. The script run is shown as below -

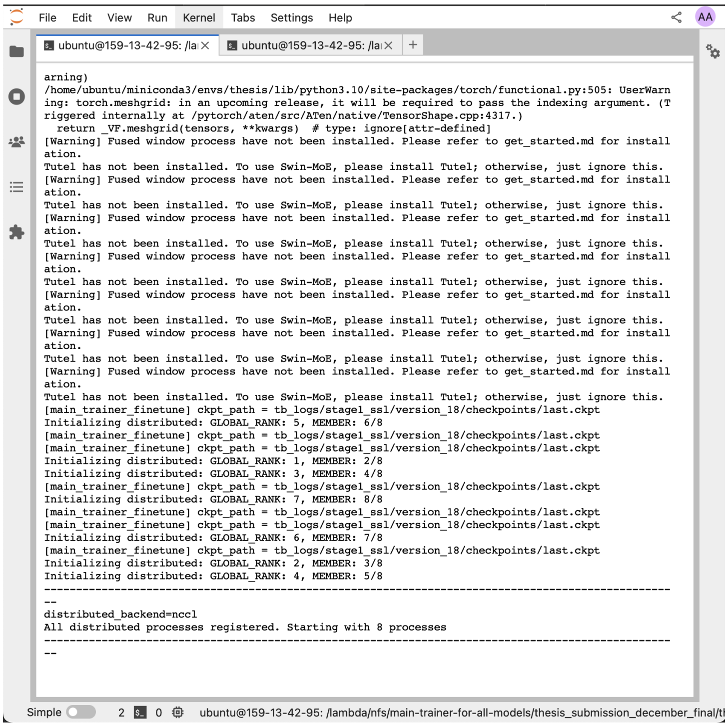

Figure 16: Stage 2 in progress

6. The GPU memory and usage are monitored to ensure maximum utilisation and efficiency -

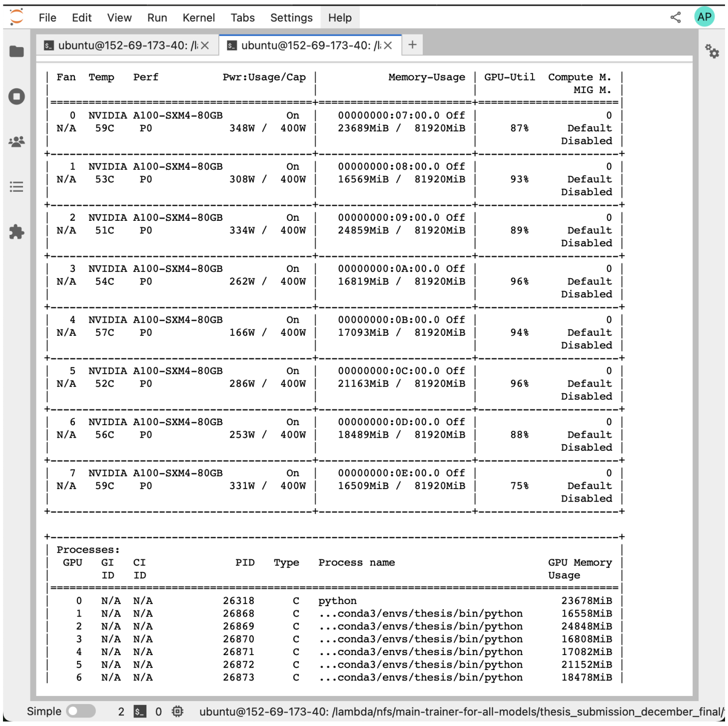

Figure 17: GPU memory utilisation for the stage 1/2 training

References

Brian Dolhansky, Joanna Bitton, B. P. (2020). The DeepFake detection challenge dataset.

Cai, Z., Ghosh, S., Dhall, A., Gedeon, T., Stefanov, K. and Hayat, M. (2023). Glitch in the matrix: A large scale benchmark for content driven audio–visual forgery detection and localization, *Computer Vision and Image Understanding* **236**: 103818.

Dolhansky, B., Bitton, J., Pflaum, B., Lu, J., Howes, R., Wang, M. and Ferrer, C. C. (2020). The DeepFake Detection Challenge (DFDC) Dataset.

Ephrat, A., Mosseri, I., Lang, O., Dekel, T., Wilson, K., Hassidim, A., Freeman, W. T. and Rubinstein, M. (2018). Looking to Listen at the Cocktail Party: A Speaker-Independent Audio-Visual Model for Speech Separation, *ACM Transactions on Graphics* **37**(4): 1–11.