Code Master Thesis: Deep Neural Extraction and Analysis of Linguistic Cues from Synchronous Multiplayer Social Deduction Gaming Streams

Tim Kleinlein

September 2024

Contents

1 Description of this Code Guide

In this document, the code related to the master thesis "Deep Neural Extraction and Analysis of Linguistic Cues from Synchronous Multiplayer Social Deduction Gaming Streams" is explained. The repository is basically divided into four main folders, `Lobby-Synchronization`, `Speech-Extraction`, `Image-Extraction`, and `Analysis`. These folders, which all describe a different part of the Information Extraction and Analysis Pipeline are presented separately in the following. For this I always present the main Python files and the order in which they should be executed. As I also provide for each Python file the Input and the Output of the file. In this way, all the other important folders / files of the repository are described as well.

For these other important folders / files it should be noted that a lot of data is not stored and displayed here in the remote repository, but is instead stored exclusively on the server. This includes, in particular, various video and audio files created during the Information Extraction and Analysis pipeline. Trained models and generated plots and logs for displaying results are also affected. The decision was made because the data volumes are too large to be stored in the remote repository. The corresponding folders are thus also not displayed here, but can be found on the server.

2 Lobby Synchronization

This is the part of the code in which I perform the Lobby Extraction and Synchronization. In the thesis, the procedure is described in Chapter 5.1, namely Temporal Synchronization of VODs by Lobby Extraction.

Order how the included Python files have to be executed:

1. `src/initial_synchronization/synchronization.py`
2. `src/final_synchronization/streamer_dictionaries.py`
3. `src/final_synchronization/lobbies_dictionaries.py`
4. `src/final_synchronization/participated_lobbies.py`
5. `src/final_synchronization/combine_results.py`
6. `src/evaluation/evaluation.py`

2.1 src/initial_synchronization/synchronization.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Lobby-Synchronization/src/initial_synchronization/synchronization.py`

What is done in the file:

- Using the events from image recognition stored in the SRT files and the metadata of the VODs from the database, this script creates first lobby proposals on individual VOD level.
- Creates session wide lobby proposals.
- Applies various heuristics for automatic lobby extraction and synchronization.
- Assigns streamer to lobbies.

Input:

- Session folder (`../../../../../../dimstore/pop520978/data`): Contains session data with SRT files.
- Database file (`../../data/streams_metadata/vods.db`): Contains metadata about VOD streams.

Output:

- Session Output Files (`../../data/initial_synchronization_output/Extraction/{session}.txt`): Logs the result of lobby extraction for each session, later used for manual examination (stores streamers and lobbies with no trustworthy lobby, also stores streamers with non-consecutive lobby assignments, etc.).

- Assigned Lobbies (../../data/initial_synchronization_output/assignedLobbiesDfs/{session}.csv): CSV file mapping streamers to the assigned lobbies.
- Streamers' Dictionary (../../data/initial_synchronization_output/streamer_dictionaries/{session}_streamer.pkl): Pickle file with trustworthy lobby times for streamers.
- Lobbies' Dictionary (../../data/initial_synchronization_output/lobbies_dictionaries/{session}_lobbies.pkl): Pickle file with trustworthy lobby times for each lobby.

2.2 src/final_synchronization/streamer_dictionaries.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Lobby-Synchronization/src/final_synchronization/streamer_dictionaries.py

What is done in the file:

- Loads and stores trustworthy lobby times for all streamers: either from programmatic or from manual extraction.
- Calculates time differences between streamers during lobbies and then solves for the most likely average time offsets between them, building a dictionary of time differences.
- Builds a system of linear equations from time differences and solves it using least-squares optimization to determine the relative offsets between all streamers.

Input:

- Streamer Dictionary Files (../../data/initial_synchronization_output/streamer_dictionaries/{session}): Pickle files containing trustworthy lobby times for each streamer.
- Excel Files (../../data/manual_lobby_extraction/Results.xlsx): Contains manually extracted lobby times to update streamers' trustworthy lobby times.
  - Sheet: Streamer_Trustworthy_Lobbies: Trustworthy lobby times for streamers.
  - Sheet: Main_Streamer_Lobbies: Main streamers' lobby times.

Output:

- Streamer Offset File (../../data/final_synchronization_output/streamer_offsets.pkl): Pickle file with the calculated average time offsets for all streamers.

- Streamer Start Times File (`../../data/final_synchronization_output/streamer_start_times.pkl`): Pickle file containing the start time of each streamer.

2.3 src/final_synchronization/lobbies_dictionaries.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Lobby-Synchronization/src/final_synchronization/lobbies_dictionaries.py`

What is done in the file:

- Loads and stores trustworthy lobby times for all lobbies: either from programmatic or from manual extraction.
- Saves this dictionary of trustworthy lobby times.

Input:

- Streamer Dictionary Files (`../../data/initial_synchronization_output/streamer_dictionaries/{session}`): Pickle files containing trustworthy lobby times for each streamer. Needed as to integrate trustworthy lobby times from manual examination, the start time of the streamer for whom it was extracted is used.
- Lobbies Dictionary Files (`../../data/initial_synchronization_output/lobbies_dictionaries/{session}`): Pickle files storing trustworthy lobby times for each lobby.
- Excel File (`../../data/manual_lobby_extraction/Results.xlsx`):
  - Sheet: `Lobbies_Trustworthy_Lobbies`: Contains manually extracted trustworthy lobby times for each session.

Output:

- Updated Lobbies Dictionary (`../../data/final_synchronization_output/trustworthy_lobbies.pkl`): Pickle file storing the updated trustworthy lobby times for each lobby after incorporating manually extracted times.

2.4 src/final_synchronization/participated_lobbies.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Lobby-Synchronization/src/final_synchronization/participated_lobbies.py`

What is done in the file:

- Corrects lobby assignments of streamers using manual examination results: handles cases where streamers have non-consecutive or missing lobby assignments using manual examination results.

- Removes incorrect or duplicate lobby assignments, fills in missing consecutive lobbies, and corrects specific streamers' data based on manual examination results.
- Saves final lobby assignments.

Input:

- Excel File (`../../data/manual_lobby_extraction/Results.xlsx`):
  - Sheet: `Substitute_Nones`: Contains manual examination results for streamers with non-consecutive or missing lobby assignments.
- Session CSV Files (`../../data/initial_synchronization_output/assignedLobbiesDfs/{session}`): CSV files containing initial lobby assignments for each streamer in different sessions from programmatic lobby extraction processes.

Output:

- Final Lobby Assignments File (`../../data/final_synchronization_output/final_lobby_assignments.pkl`): Pickle file containing the corrected and final lobby assignments for each streamer.

2.5 src/final_synchronization/combine_results.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Lobby-Synchronization/src/final_synchronization/combine_results.py`

What is done in the file:

- Calculates final lobby times for streamers in all sessions by combining trustworthy lobby times, lobby assignments of streamers, and individual streamer offsets.
- Script calculates the final lobby times for all participating streamers by taking a trustworthy lobby time for the respective lobby and adjusting it with each streamer's individual offset with the selected streamer the trustworthy lobby time is from.
- To select this streamer, builds a ranking of the most reliable streamers, i.e., streamers with most trustworthy lobby times.
- Calculated lobby times are then converted to an SRT-like format by subtracting each streamer's start time.
- Resulting lobby times are saved.

Input:

- Streamer Offsets (`../../data/final_synchronization_output/streamer_offsets.pkl`): contains the calculated time offsets for each streamer relative to others.
- Trustworthy Lobbies (`../../data/final_synchronization_output/trustworthy_lobbies.pkl`): Pickle file containing trustworthy lobby times for each session.
- Final Lobby Assignments (`../../data/final_synchronization_output/final_lobby_assignments.pkl`): Contains information about which lobbies each streamer participated in.
- Streamer Start Times (`../../data/final_synchronization_output/streamer_start_times.pkl`): Pickle file with start times for each streamer.

Output:

- Final Lobby Times (`../../data/final_synchronization_output/final_lobby_times.pkl`): Pickle file storing the final calculated lobby times for each streamer in an SRT-like format.

2.6 src/evaluation/evaluation.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Lobby-Synchronization/src/evaluation/evaluation.py`

What is done in the file:

- Randomly selects 10 streamer-lobby pairs from the final lobby times data for manual evaluation.

Input:

- Final Lobby Times (`../../data/final_synchronization_output/final_lobby_times.pkl`): Pickle file containing the final calculated lobby times for each streamer.

Output:

- Only prints the 10 streamer-lobby pairs to check. Final evaluation results are at (`../../data/evaluation/Evaluation.xlsx`).

3 Speech Extraction

This is the part of the code in which I perform the Extraction of Text Data from the Discussions. In the thesis, the procedure is described in Chapter 5.2, namely Extraction of Relevant Text Data. The code in this subdirectory can basically be divided into three parts: 1. The identification of discussion rounds and their transcription. This also comes with the identification of movement phases, from them audio snippets with utterances of individual streamers are extracted and prepared such that they can be used to make training data for the PVAD speaker diarization model. This first part contains all the Python files which lie in this subdirectory directly. 2. The training of a PVAD model. In this part, the training data for a PVAD model are created, the model is set up and trained, and the best-performing model is saved. 3. The application of the trained PVAD model on the discussion round data. In this part, the model is applied on the discussion round audios and the results are saved appropriately.

Order of execution for the included Python files:

1. `speaker_diarization.py`
2. `check_speaking_behavior.py`
3. `pyannote_to_discussion_rounds.py`
4. `finalize_discussion_rounds.py`
5. `transcribe_discussion_rounds.py`
6. `evaluation_discussion_rounds.py`
7. `extract_movement_phases.py`
8. `create_alignments.py`
9. `test_accurate_subclip_extraction.py`

Before proceeding with the second part (the setup and training of the PVAD model), I performed some Bash commands to copy the audio snippets with the individual streamers' utterances, along with their alignment and transcription files, from the `train_data_diarization` folder to the `Personal_VAD_Model/data/LibriSpeech/dev-clean` folder where the data for PVAD model training is stored.

1. `Personal_VAD_Model/src/stereo_to_mono_audio.py`
2. `Personal_VAD_Model/src/generate_abbreviations.py`
3. `Personal_VAD_Model/src/extract_embeddings.py`
4. `Personal_VAD_Model/prepare_dataset_features.sh`

5. `Personal_VAD_Model/src/concatenate_utterances.py`
6. `Personal_VAD_Model/src/extract_features.py`
7. `Personal_VAD_Model/src/combine_load_train.py`
8. `Personal_VAD_Model/prepare_dataset_features_test_data.sh`
9. `Personal_VAD_Model/src/compare_trained_models.py`

The third part is the application of the trained PVAD model on the discussion round audios. Before doing this in the `PVAD_Application` subdirectory, I copied some relevant files to `PVAD_Application/data`: the embeddings of the speakers (`embedding`), the two dictionaries for abbreviating streamer names (`abbr_dict.pkl` and `abbr_dict_reverse.pkl`), and the final PVAD model (`vad_set_ut100000.pt`). This final PVAD model was trained for six epochs, as decided based on the log output in `Personal_VAD_Model/test_log.txt`.

1. `PVAD_Application/prepare_dataset_features.sh`
2. `PVAD_Application/src/create_alignments_application.py`
3. `PVAD_Application/src/extract_features_application.py`
4. `PVAD_Application/src/apply_model.py`
5. `PVAD_Application/src/analyze_model_output.py`
6. `PVAD_Application/src/check_results.py`
7. `PVAD_Application/src/extract_final_transcriptions.py`
8. `PVAD_Application/src/evaluate_final_transcriptions.py`

3.1 speaker_diarization.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/speaker\protect\penalty\z@_diarization.py`

What is done in the file:

- Speaker diarization of all identified lobbies to identify discussion rounds
- Extracts relevant video sections according to identified lobbies, converts to audio file, and then uses Pyannote speaker diarization model on them
- Audio segments of streamers who missed the start of the lobby are adjusted by adding silence
- Identifies likely discussion rounds on the streamer level by the output of the speaker diarization model based on speaker turns, applying heuristics

- Saves identified likely discussion round candidates on the streamer level

Input:

- Video Files (`../../../../dimstore/pop520978/data/{session}/{streamer}.mkv`): The MKV video files for each streamer in each session.
- Final Lobby Times (`../Lobby_Synchronization/data/final_synchronization_output/final_lobby_times.pkl`): Pickle file containing the final lobby times, used to identify the start and end times for the audio extraction.

Output:

- Extracted Audio Files (`{streamer}_l{lobby}_audio.wav`): WAV files containing the extracted audio segments for each streamer in each lobby.
- Final Discussion Rounds (`final_discussion_rounds_s_{session}_l_{lobby}.pkl`): Pickle files storing the extracted discussion rounds for each streamer in each session and lobby.

3.2 check_speaking_behavior.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/check\protect\penalty\z@_speaking\protect\penalty\z@_behavior.py`

What is done in the file:

- Script extracts for each session one lobby as a video file to check manually how the settings regarding the maximum length of the discussion rounds are (there are two popular options that the streamers usually agree on at the start of a session).
- These are then manually checked and saved to `StreamsSpeakingBehavior.xlsx` to be used later on for discussion identification.

Input:

- Final Lobby Times (`../Lobby_Synchronization/data/final_synchronization_output/final_lobby_times.pkl`): Contains the lobby times for each session and streamer.
- MKV Video Files (`../../../../dimstore/pop520978/data/{session}/{streamer}.mkv`): The game recordings for each session and streamer.

Output:

- Extracted Video Files (`{streamer}_l2.mkv`): The video segment of one streamer and one lobby for each session.

3.3 pyannote_to_discussion_rounds.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/pyannote\protect\penalty\z@_to\protect\penalty\z@_discussion\protect\penalty\z@_rounds.py

What is done in the file:

- Loads and refines discussion round candidates on streamer level by merging short discussion rounds based on the proximity of their timestamps.
- Then groups these potential discussion starts and ends on streamer level into clusters of identified discussion starts and ends on lobby level.
- Uses these clusters to extract likely discussion start and end points on lobby level by applying multiple heuristics (e.g., using quantiles on clusters, checking for identified single discussion starts and ends if they are realistic, etc.).
- Saves refined discussion rounds on lobby level and generates Excel files to facilitate manual inspection of any uncertain cases.

Input:

- Final Lobby Times (../Lobby_Synchronization/data/final_synchronization_output/final_lobby_times.pkl): Contains the lobby times for each session and is used to determine when to extract discussion rounds.
- Discussion Lengths (StreamsSpeakingBehavior.xlsx): Excel file specifying the typical lengths of discussion rounds for different sessions (created by manual inspection beforehand).
- Extracted Discussion Rounds (output_pyannote/final_discussion_rounds_s_{session}_l_{lobby}.pkl): Pickle files storing the raw discussion rounds extracted using the Pyannote pipeline in speaker_diarization.py.

Output:

- Refined Discussion Rounds (output_discussion_rounds/final_discussion_rounds_from_pyannote.pkl): Pickle file storing the refined discussion rounds for each session and lobby after the merging and splitting processes.
- Excel File for Manual Inspection (ManualExaminationDiscussionRounds.xlsx): Excel file listing the discussion rounds with potential issues (e.g., missing start or end points) for easier manual inspection.
- Extracted MKV Clips: When the script encounters incomplete discussion rounds (with missing start/end times), it extracts video clips of the discussion from the MKV files for manual review.

3.4 finalize_discussion_rounds.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/finalize\protect\penalty\z@_discussion\protect\penalty\z@_rounds.py

What is done in the file:

- Finalizes discussion rounds by combining results from programmatic extraction with manual examination results.
- Saves final identified discussion rounds.

Input:

- Manual Extraction Discussion Rounds (`output_discussion_rounds/ME_DR.xlsx`): Excel file containing the manually extracted discussion rounds with corrected start and end times.
- Automated Discussion Rounds (`output_discussion_rounds/final_discussion_rounds_from_pyannote.pkl`): Pickle file storing the programmatically extracted discussion rounds from the Pyannote pipeline plus refinement.

Output:

- Final Discussion Rounds (`output_discussion_rounds/final_discussion_rounds.pkl`): Pickle file storing the combined and finalized discussion rounds with both automated and manually corrected timestamps.

3.5 transcribe_discussion_rounds.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/transcribe\protect\penalty\z@_discussion\protect\penalty\z@_rounds.py

What is done in the file:

- Extracts discussion round segments from game recordings with some buffer (see thesis) and converts them to audio files.
- Uses `stable_whisper` model to transcribe them.
- Saves the results as `srt` files, renames, and organizes the files for easy access.

Input:

- Final Discussion Rounds (`output_discussion_rounds/final_discussion_rounds.pkl`): Pickle file containing the finalized discussion rounds.

- Final Lobby Times (`../Lobby_Synchronization/data/final_synchronization_output/final_lobby_times.pkl`): Contains the lobby times for each session and streamer.
- Streamer Audio Data (`Personal_VAD_Model/data/LibriSpeech/dev-clean`): Directory containing the relevant streamers for transcription (only streamers for which enough utterances could be defined in movement phase are transcribed here).

Output:

- Discussion Round Audio Files (`relevant_discussion_rounds_audio/{session}_{lobby}_{round}_{streamer}.wav`): Extracted audio segments for each streamer's discussion rounds.
- Transcription Files (`relevant_transcriptions/{session}_{lobby}_{round}_{streamer}.srt`): Subtitle files containing the transcriptions for each discussion round, saved in SRT format.

3.6 evaluation_discussion_rounds.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/evaluation\protect\penalty\z@_discussion\protect\penalty\z@_rounds.py`

What is done in the file:

- Script randomly selects and prints 10 discussion rounds from the extracted data for manual evaluation.

Input:

- Final Discussion Rounds (`output_discussion_rounds/final_discussion_rounds.pkl`): Pickle file containing the finalized discussion rounds.
- Final Lobby Times (`../Lobby_Synchronization/data/final_synchronization_output/final_lobby_times.pkl`): Contains the lobby times for each session and streamer.

Output:

- Prints 10 random discussion round-streamer pairs, along with their start and end times, for manual evaluation. These can be used to check the quality of the extracted discussion rounds.

3.7 extract_movement_phases.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/extract\protect\penalty\z@_movement\protect\penalty\z@_phases.py`

What is done in the file:

- Script extracts movement phases, i.e., periods between discussions where streamers move around in the game.
- First calculates movement phases based on the time gaps between the loaded final identified discussion rounds.
- Filters out short movement phases and retains only those longer than 60 seconds.
- Script creates audio segments corresponding to these movement phases identified in this way.
- Runs a speaker diarization pipeline over these extracted audio files to identify utterances made during movement phases, discarding segments with multiple speakers. Extracts and saves these utterances (these audio snippets will later on be used to create training data for Personal VAD speaker diarization model).

Input:

- Final Discussion Rounds (`output_discussion_rounds/final_discussion_rounds.pkl`): Pickle file containing the finalized discussion rounds.
- Final Lobby Times (`../Lobby_Synchronization/data/final_synchronization_output/final_lobby_times.pkl`): Contains the lobby times for each session and streamer.

Output:

- Extracted Audio Segments (`train_data_diarization/{streamer}/{session}_{lobby}_{movement_phase}.wav`): Audio files extracted for movement phases where the streamer is likely the only speaker.
- Extracted Utterances (`train_data_diarization/{streamer}/utterances/{movement_phase_utterance}.wav`): Audio segments of individual utterances extracted from movement phases.

3.8 create_alignments.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/create\protect\penalty\z@_alignments.py`

What is done in the file:

- Script performs transcription and word alignment for audio utterances extracted from the movement phases of streamers.
- For each of the streamer's audio snippets with the individual utterances, it first converts the WAV files to FLAC format.

- Then generates transcriptions of the audio snippets.
- Processes the transcription results to extract words and their corresponding timestamps in the format required to be used for the PVAD training. Silences between words are accounted for.
- Saves created alignments and transcriptions.

Input:

- Utterance Audio Files (`train_data_diarization/{streamer}/utterances/{utterance}.wav`): Audio snippets of each streamer's utterances, extracted from the movement phases.

Output:

- Word Alignment Files (`train_data_diarization/{streamer}/{streamer}.alignments.txt`): Text file containing the word-by-word alignments with their respective timestamps.
- Transcription Files (`train_data_diarization/{streamer}/{streamer}.trans.txt`): Text file storing the transcriptions of each utterance for a given streamer.
- SRT Files (`train_data_diarization/{streamer}/utterances_srt_files/{utterance}.srt`): Files generated for each utterance, storing transcription and timestamp information.

3.9 test_accurate_subclip_extraction.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/src/test\protect\penalty\z@_accurate\protect\penalty\z@_subclip\protect\penalty\z@_extraction.py`

What is done in the file:

- File to test and evaluate subclip extraction from MKV video files using FFmpeg with precise start and end times.
- Result: it is not exact to the millisecond.

Input:

- MKV Video File (`../../pop520978/data/2022-02-23_S1/2022-02-23_S1_vikramafc_1307860812.mkv`): The video file from which a segment is extracted as a test.

Output:

- Extracted Video Clip (`extract.mkv`): A video file containing the extracted subclip.
- Duration of Extracted Segment: Prints the duration of the extracted segment in milliseconds to the console.

3.10 Personal_VAD_Model/src/stereo_to_mono_audio.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/Personal\protect\penalty\z@_VAD\protect\penalty\z@_Model/src/stereo\protect\penalty\z@_to\protect\penalty\z@_mono\protect\penalty\z@_audio.py

What is done in the file:

- Script is responsible for converting the audio snippets of the individual streamer utterances to the required format for further processing in the Personal VAD model pipeline.
- Stereo audio files are converted to mono by averaging the audio channels.
- It also resamples the audio to a 16,000 Hz sample rate.

Input:

- FLAC audio files from the individual streamers dataset (`data/LibriSpeech/dev-clean`): The source directory containing the stereo FLAC files.

Output:

- Converted mono FLAC files: The original stereo files are overwritten with their mono counterparts, resampled to 16,000 Hz.

3.11 Personal_VAD_Model/src/generate_abbreviations.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/Personal\protect\penalty\z@_VAD\protect\penalty\z@_Model/src/generate\protect\penalty\z@_abbreviations.py

What is done in the file:

- This script generates unique abbreviations for the names of subdirectories (representing speakers) in the streamer utterances dataset. This is needed such that later on when concatenated utterances are created, the file name of these concatenated utterances is not too long.
- Also creates a reverse mapping from abbreviations to full names.
- The resulting abbreviation dictionaries are saved as pickle files for later use in the Personal VAD model.

Input:

- Directory path to the Streamer utterances dataset (`Personal_VAD_Model/data/LibriSpeech/dev-clean`): The directory containing speaker subdirectories for which abbreviations are generated.

Output:

- Abbreviation Dictionary Pickle File (`Personal_VAD_Model/data/abbr_dict.pkl`): A dictionary mapping full names of subdirectories to their abbreviations.
- Reverse Abbreviation Dictionary Pickle File (`Personal_VAD_Model/data/abbr_dict_reverse.pkl`): A dictionary mapping abbreviations back to the full names.

3.12 Personal_VAD_Model/src/extract_embeddings.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/Personal\protect\penalty\z@_VAD\protect\penalty\z@_Model/src/extract\protect\penalty\z@_embedding.py`

What is done in the file:

- Script extracts speaker embeddings for each of the eight relevant speakers (speakers for whom I have extracted at least 100 audio snippets).
- Calculates d-vectors using the Resemblyzer library and saves them in the Kaldi archive format (ark/scp).

Input:

- Dataset with the relevant streamer's audio snippets (`data/LibriSpeech`): Directory containing FLAC audio snippets with individual utterances for all relevant streamers.

Output:

- Embedding files in `data/embeddings`:
  - `dvectors.ark` and `dvectors.scp`: Files containing d-vector embeddings in Kaldi format.

Execution: The script must be executed with the following command:

```
python extract_embedding.py --dvector
```

3.13 Personal_VAD_Model/prepare_dataset_features.sh

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/Personal\protect\penalty\z@_VAD\protect\penalty\z@_Model/prepare\protect\penalty\z@_dataset\protect\penalty\z@_features.sh`

What is done in the file:

- Bash script that handles all data preparation steps to create training data for the PVAD model coming from individual audio snippets.

- Generates concatenated utterances from the audio snippets using `src/concatenate_utterances.py`.
- Extracts features required for training from the concatenated utterances using `src/extract_features.py`.
- In the current setting, 100,000 concatenated utterances are created for model training.

Execution: This script must be executed as part of the process to prepare the dataset and extract features using the following command:

```
bash prepare_dataset_features.sh 0
```

Input (everything is already set correctly when executed as stated before):

- Audio snippet dataset (`data/LibriSpeech`): Directory containing FLAC files for each speaker.

Output:

- Concatenated utterances in `data/clean`
- Extracted features saved to `data/features_demo`

3.14 Personal_VAD_Model/src/concatenate_utterances.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/Personal\protect\penalty\z@_VAD\protect\penalty\z@_Model/src/concatenate\protect\penalty\z@_utterances.py`

What is done in the file:

- Script generates 100,000 concatenated utterances using the audio snippets from the eight relevant streamers.
- Additionally, it creates corresponding alignment files for the concatenated utterances by using the alignment files of the individual utterances.

(Additional) Input:

- `abbr_dict.pkl`: A dictionary for abbreviating filenames, loaded from `data/abbr_dict.pkl`. This is needed because otherwise the filenames of the concatenated utterances would become too long.

Output:

- Concatenated audio files saved as FLAC in `data/clean`.
- `wav.scp`: A Kaldi-compatible file that lists all the audio files generated.

- `utt2spk`: A Kaldi-specific file mapping each utterance to the corresponding speaker (needed as I have the concatenated utterances with multiple speakers now).
- `text`: A file containing the aligned transcripts and their respective timestamps for each generated utterance.

3.15 Personal_VAD_Model/src/extract_features.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/Personal\protect\penalty\z@_VAD\protect\penalty\z@_Model/src/extract\protect\penalty\z@_features.py`

What is done in the file:

- Randomly chooses the target speaker for concatenated utterances and then, according to this, labels the concatenated utterances.
- Script is responsible for the feature extraction process from the concatenated utterances: loads the audio waveform data of the concatenated utterances and extracts acoustic features (e.g., filterbanks) from them.
- Also extracts frame-level d-vector embeddings by processing these features combined with the embeddings of the target speaker. Then calculates respective frame-level and window-level similarity scores with the target speaker.
- Saves for each concatenated utterance the chosen target speaker, the respective labels, the extracted features, and the calculated similarity scores.

(Additional) Input:

- D-vector embeddings (`data/embeddings/`): Directory containing speaker embeddings used to calculate cosine similarities.

Output:

- Filterbank acoustic features (`fbanks.scp/.ark`): Acoustic features saved as filterbanks.
- Cosine similarity scores (`scores.scp/.ark`): Cosine similarity scores for each frame of the extracted features.
- Ground truth labels (`labels.scp/.ark`): PVAD ground truth labels for each frame in the concatenated utterances.
- Target speaker information (`targets.scp`): Contains the target speaker for each concatenated utterance.

3.16 Personal_VAD_Model/src/combine_load_train.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/Personal\protect\penalty\z@_VAD\protect\penalty\z@_Model/src/combine\protect\penalty\z@_load\protect\penalty\z@_train.py

What is done in the file:

- This script loads the preprocessed features from Kaldi files I generated for model training with the training of a Personal Voice Activity Detection (VAD) model.
- The loaded training data includes the extracted filterbank features, d-vectors, and labels from the concatenated audio dataset.
- Then the file sets up the PVAD model by defining a class with the model architecture.
- The model is trained for 10 epochs and after each epoch, the intermediate models are saved.
- The logs from the model training are written in a log file.

Input:

- `data/features_demo/fbanks.scp`: The feature file containing filterbank features for each utterance.
- `data/features_demo/scores.scp`: A file containing similarity scores of the speaker embeddings.
- `data/features_demo/labels.scp`: The label file, with ground truth information about which parts of the audio correspond to the target speaker, non-target speakers, or silence.
- `data/embeddings/dvectors.scp`: The d-vector embeddings of speakers used to determine the target speaker's embedding.
- `data/features_demo/targets.scp`: The file mapping each utterance to its corresponding target speaker.

Output:

- `data/vad_set_ut100000_final.pt`: The final trained model is saved to this file.
- Intermediate models are saved after each epoch, e.g., `data/vad_set_ut100000_{epoch}.pt`.
- `training_log.txt`: The log file containing training progress and performance metrics such as accuracy, precision, and recall for each label across all epochs.

3.17 Personal_VAD_Model/prepare_dataset_features_test_data.sh

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/Personal\protect\penalty\z@_VAD\protect\penalty\z@_Model/prepare\protect\penalty\z@_dataset\protect\penalty\z@_features\protect\penalty\z@_test\protect\penalty\z@_data.sh

What is done in the file:

- Same as in `prepare_dataset_features.sh`, only that I now create additional 10,000 concatenated utterances as test data.

Output:

- Concatenated test utterances in `data/test`
- Extracted features saved to `data/features_test`

3.18 Personal_VAD_Model/src/compare_trained_models.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/Personal\protect\penalty\z@_VAD\protect\penalty\z@_Model/src/compare\protect\penalty\z@_trained\protect\penalty\z@_models.py

What is done in the file:

- Script evaluates the performance of several trained Personal VAD models (model training saved the models after each epoch) on created test dataset to find the best number of epochs for model training.
- Compares their accuracy, precision, and recall metrics across three possible labels: no speech, non-target speaker speech, and target speaker speech.

Input:

- `data/features_test/fbanks.scp`: The feature file containing filterbank features for each test utterance.
- `data/features_test/scores.scp`: A file containing scores representing speaker embeddings.
- `data/features_test/labels.scp`: The label file, with ground truth information about which parts of the test audio correspond to the target speaker, non-target speakers, or silence.
- `data/embeddings/dvectors.scp`: The d-vector embeddings of speakers used to determine the target speaker's identity.

- data/features_test/targets.scp: The file mapping each test utterance to its corresponding target speaker.
- Multiple pre-trained models: vad_set_ut100000_6.pt, vad_set_ut100000_7.pt, vad_set_ut100000_8.pt, vad_set_ut100000_9.pt, vad_set_ut100000_10.pt.

Output:

- Logs: Model performance metrics (accuracy, precision, recall, and loss) for each of the pre-trained models are saved to test_log.txt.

3.19 PVAD_Application/prepare_dataset_features.sh

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/PVAD\protect\penalty\z@_Application/prepare\protect\penalty\z@_dataset\protect\penalty\z@_features.sh

What is done in the file:

- Automates the process of preparing the discussion round audios for the Personal VAD application by creating audio alignments, converting the audios to the required format, and performing feature extraction. It splits the dataset for parallel processing and extracts features like Mel-spectrograms and cosine similarity scores.
- The two scripts used are PVAD_Application/src/create_alignments_application.py and PVAD_Application/src/extract_features_application.py.

Execution: This script is executed as part of the Personal VAD application pipeline to prepare datasets in different stages. It is run as follows:

```
bash prepare_dataset_features.sh 0
```

Input:

- data/wav.scp: SCP file containing paths to the audio files.
- data/alignments: Directory containing the alignment files.
- data/embeddings/dvectors.scp: D-vector embeddings for speaker features.

Output:

- data/features_application/fbanks.scp: Concatenated Mel-spectrogram features.
- data/features_application/scores.scp: Concatenated cosine similarity scores.
- data/features_application/targets.scp: Concatenated target speaker identifiers.

3.20 PVAD_Application/src/create_alignments_application.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/PVAD\protect\penalty\z@_Application/src/create\protect\penalty\z@_alignments\protect\penalty\z@_application.py

What is done in the file:

- Script processes the transcriptions and audio files from the discussion rounds to create alignment files for each discussion round in the specific format (such that the PVAD model can be applied on the discussion rounds).
- Audio streams are converted to a mono audio stream at a 16000 Hz sampling rate, which is needed for PVAD Application.
- Also, discussion rounds are removed/skipped, where transcription/alignment file creation does not work.

Input:

- `../relevant_transcriptions`: A directory containing the transcriptions of the discussion rounds in `srt` format.
- `../relevant_discussion_rounds_audio`: The corresponding audio files in `wav` format.

Output:

- `data/audios`: Directory containing the converted audio files in `flac` format.
- `data/alignments/alignments.txt`: A text file containing the word alignments for each transcription.
- `data/audios/trans.txt`: A text file with the transcriptions.
- `data/wrongly_aligned_discussion_rounds.txt`: A log file for any discussion rounds that have alignment issues and were thus removed.
- `data/wav.scp` and `data/text`: Files with the aligned transcriptions and their corresponding timestamps.

3.21 PVAD_Application/src/extract_features_application.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/PVAD\protect\penalty\z@_Application/src/extract\protect\penalty\z@_features\protect\penalty\z@_application.py

What is done in the file:

- Script is responsible for extracting features from the discussion round audio files for Personal VAD application.
- It computes acoustic features and uses d-vector embedding of the respective target speaker (for each streamer, I have an own discussion round audio which is used with his target speaker embedding) to calculate respective target speaker similarity scores.
- Saves the features, scores, and targets (assignment of respective target speaker to discussion round audio, i.e., the streamer to the discussion round audio extracted from his VOD) for later use in PVAD application.

Input:

- `data/<split_*.scp>`: Input files containing paths to audio data for processing.
- `data/abbr_dict.pkl`, `data/abbr_dict_reverse.pkl`: Abbreviation dictionaries for speaker and file name mapping.
- `data/embeddings/dvectors.scp`: Speaker embeddings for d-vector extraction.

Output:

- `data/features_application/fbanks_.ark/scp`: Mel-spectrogram filterbank features for each processed audio file.
- `data/features_application/scores_.ark/scp`: Cosine similarity scores for speaker embeddings.
- `data/features_application/targets_*.scp`: Target speaker identifiers for each utterance.

3.22 PVAD_Application/src/apply_model.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/PVAD\protect\penalty\z@_Application/src/apply\protect\penalty\z@_model.py`

What is done in the file:

- Apply a trained PVAD model to the discussion rounds, generating predictions for 10ms intervals for each discussion round.
- Classifies each frame into one of three labels (no speech: 0, non-target speaker speech: 1, or target speaker speech: 2).
- The results are saved: one dictionary which contains, for each frame, probabilities for each label.

Input:

- `data/features_application/fbanks.scp`: Filterbank features for the discussion rounds.
- `data/features_application/scores.scp`: Cosine similarity scores of the discussion rounds.
- `data/features_application/targets.scp`: Target speaker identifiers of the discussion rounds.
- `data/embeddings/dvectors.scp`: Embeddings for respective streamers.

Output:

- `data/pvad_output_files/{utt_id}_labeled_intervals.pkl`: Pickle files containing, for all classified discussion rounds, the probability predictions of each label for each frame.

3.23 PVAD_Application/src/analyze_model_output.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/PVAD\protect\penalty\z@_Application/src/analyze\protect\penalty\z@_model\protect\penalty\z@_output.py`

What is done in the file:

- Analyzes the output generated from the PVAD model, specifically analyzing the frame-level predictions across all discussion rounds.
- Calculates statistics such as the ratio of different labels (0, 1, 2), file lengths, and how many predictions of the label 2 (target speaker) are retained at different probability thresholds.
- The analysis results are visualized in several plots and then are used to define probability thresholds for creating final transcriptions, i.e., the minimum probability threshold for label 2 predictions to be kept.

Input:

- `data/pvad_output_files/*labeled_intervals.pkl`: Pickle files containing the frame-level probability predictions for each discussion round.
- `data/pvad_output_files/*nontarget_label_dict.pkl`: Pickle files containing timestamp intervals where the label is not 2 or has a probability lower than a given threshold (only used for naming here, not analyzed).

Output:

- `final_plots/ratio__percent_distribution.png`: Histograms showing the distribution of the ratio of target speaker predictions at various probability thresholds (50%, 60%, etc.).

- final_plots/mean_ratio_label_2_predictions_kept.png: Bar plot showing the mean ratio of target speaker predictions kept for each probability threshold.
- final_plots/total_frequency_predictions_per_label.png: Bar plot showing the total frequency of predictions for each label (0: no speech, 1: non-target speaker, 2: target speaker).
- final_plots/ratio_distribution_label_.png: Histograms showing the distribution of each label (0, 1, 2) within each discussion round.
- final_plots/file_lengths_distribution.png: Histogram showing the distribution of discussion round lengths in seconds.

3.24 PVAD_Application/src/check_results.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/PVAD\protect\penalty\z@_Application/src/check\protect\penalty\z@_results.py

What is done in the file:

- The script processes the labeled intervals from the label_dict.pkl file and extracts the corresponding audio segments (target speaker speech) from some of the original discussion audio files.
- These extracted audio segments are saved as new .flac files so one can analyze how well the model worked.

Input:

- label_dict.pkl: A pickle file containing a dictionary of filenames with their corresponding target speaker time intervals (start, end in seconds).
- data/audios/*flac: The original audio files from the discussion rounds in flac format from which the target speaker audio segments will be extracted.

Output:

- data/target_audios/*_target.flac: Extracted audio segments for each file, containing only the portions of the audio where the target speaker is detected.

3.25 PVAD_Application/src/extract_final_transcriptions.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/PVAD\protect\penalty\z@_Application/extract\protect\penalty\z@_final\protect\penalty\z@_transcriptions.py

What is done in the file:

- Script combines transcriptions of discussion rounds (from `srt` files) with the labeled intervals from the output of the PVAD model to obtain final transcriptions for each target speaker.
- First combines all utterances from transcriptions using the proposals for concatenated utterances from ts-whisper.
- Checks for the created concatenated utterances if they belong to the target speaker. Thereby, two thresholds have to be defined: the probability threshold for a target speaker speech frame classification to be valid and a minimum ratio of such target speaker frame classifications a concatenated utterance needs to be classified as a target speaker utterance.
- Three precision modes as described in the thesis are used for assigning final transcriptions: `low_acc`, `mid_acc`, and `high_acc`.
- Saves the transcriptions for each discussion round for all three precision modes in a dictionary.

Input:

- `data/pvad_output_files/*_labeled_intervals.pkl`: The labeled intervals for each discussion round generated by the PVAD model.
- `../relevant_transcriptions/*srt`: Subtitle files (`srt`) containing the created transcriptions for the discussion rounds.

Output:

- `data/final_transcriptions.pkl`: A pickle file containing the extracted final transcriptions for each discussion round, categorized by the three accuracy thresholds: `low_acc`, `mid_acc`, and `high_acc`. For each discussion round (streamer) all the final utterances assigned to him are stored.

3.26 PVAD_Application/src/evaluate_final_transcriptions.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Speech-Extraction/PVAD\protect\penalty\z@_Application/evaluate\protect\penalty\z@_final\protect\penalty\z@_transcriptions.py`

What is done in the file:

- Script randomly selects and evaluates utterances from the final transcriptions, which were previously categorized into three precision levels: `low_acc`, `mid_acc`, and `high_acc`. For each precision level, 10 random utterances are tested.

- For each utterance, the corresponding discussion round is identified, and the relevant video segment from the lobby recording is extracted. This video recording is then used for manual examination, the results of this examination are saved in `evaluation/Evaluation.xlsx`.

Input:

- `data/final_transcriptions.pkl`: The final transcriptions assigned to each streamer (for all discussion rounds) containing the extracted utterances categorized by precision levels.
- `../../Lobby_Synchronization/data/final_synchronization_output/final_lobby_times.pkl`: The lobby times, specifying when each lobby starts and ends.
- `../../../pop520978/data/{session}/{streamer}.mkv`: The original VODs from which the relevant lobby segments are extracted for manual examination.

Output:

- `evaluation/videos/{discussion_round}.mkv`: The extracted video segments corresponding to the selected utterances from each discussion round, saved in the `evaluation/videos` directory.

4 Image Extraction

This is the part of the code in which I detect the player's roles via Image Recognition. In the thesis, this procedure is described in Chapter 5.3, namely Player Role Identification via Image Recognition.

Order of execution for the included Python files:

1. `src/extract_images.py`
2. `src/ocr.py`
3. `src/finalize_manual_examination.py`
4. `src/combine_manual_automatic_examination.py`
5. `src/evaluation.py`

4.1 Image_Extraction/src/extract_images.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Image-Extraction/src/extract\protect\penalty\z@_images.py`

What is done in the file:

- Script extracts relevant images for all lobbies of the relevant streamers at various timestamps around the start of each lobby and saves them as images (to later use them for role detection).
- Extracts 16 images for each discussion lobby (5 seconds before the identified start of the lobby to 10 seconds after).

Input:

- `../Lobby-Synchronization/data/final_synchronization_output/final_lobby_times.pkl`: The timing information for each lobby, specifying when each streamer's video starts and ends.
- `../../pop520978/data/{session}/{streamer}.mkv`: The original video files for each streamer, which are used to extract frames.
- `../Speech-Extraction/Personal_VAD_Model/data/LibriSpeech/dev-clean`: The directory containing the relevant streamers only to filter which streamers are included in the frame extraction.

Output:

- `images/{session}/{lobby}/{streamer}/{timestamp}.jpg`: Extracted image frames from the video, saved for each lobby and streamer. Frames are extracted at timestamps ranging from 5 seconds before the start of the lobby to 10 seconds after.

4.2 Image_Extraction/src/ocr.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Image-Extraction/src/ocr.py

What is done in the file:

- Performs Optical Character Recognition (OCR) on the extracted frames to detect the roles of streamers in all lobbies.
- Processes each frame, extracts the relevant part of the image containing the role information, and uses Tesseract OCR to convert the image into text.
- Calculates Levenshtein distance with all roles for each frame to find the best candidate for this frame.
- For a lobby, uses a majority vote over its frames to determine the proposed role.
- If nothing can be detected, notes with None to indicate the need for manual examination.
- The detected roles are saved in a dictionary.

Input:

- `../Lobby-Synchronization/data/streams_metadata/PlayedGames.xlsx`: Metadata about each game session, including the active roles for each session.
- `images/{session}/{lobby}/{streamer}/{timestamp}.jpg`: The extracted images for each lobby of the relevant streamers, used for role detection.

Output:

- `identified_roles.pkl`: A dictionary that maps each lobby of a streamer to the most frequently detected role based on OCR results from the extracted frames.

4.3 Image_Extraction/src/finalize_manual_examination.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Image-Extraction/src/finalize\protect\penalty\z@_manual\protect\penalty\z@_examination.py

What is done in the file:

- Before this script, manual examination was applied: manually inspected and labeled all lobbies of a streamer where role assignment was None, Spy, or Seer.

- In some cases, the identified lobby start was wrong (either individually for one streamer or for the entire lobby) → delete this lobby for individual streamer or entire lobby.
- In some cases, the identified lobby start was too early (either individually for one streamer or for the entire lobby) → extract later images for manual inspection.

Input:

- `ManualExamination.xlsx`: Contains manual examination results indicating necessary adjustments such as deleting incorrect lobbies, re-assigning roles, or extracting additional images.
- `identified_roles.pkl`: A dictionary of identified roles for streamers, generated by the OCR process.
- `../Lobby-Synchronization/data/final_synchronization_output/final_lobby_times.pkl`: Contains the synchronized start times for each lobby, used for extracting additional images when necessary.

Output:

- `identified_roles.pkl`: The updated dictionary of identified roles, with manual corrections applied.
- `additional_images/{session}/{lobby}/{streamer}/{timestamp}.jpg`: Additional images extracted at later timestamps for lobbies that require further role detection attempts.

4.4 Image_Extraction/src/combine_manual_automatic_examination.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Image-Extraction/src/combine\protect\penalty\z@_manual\protect\penalty\z@_automatic\protect\penalty\z@_examination.py`

What is done in the file:

- Based on the new images extracted after the first manual examination and the following second manual review, roles are either assigned or lobbies are excluded if the role remains undetectable.
- Finalizes the process of combining manual and automatic examination results of streamer lobbies where additional images were required for accurate role assignment.

Input:

- `ManualExamination_SecondStep.xlsx`: Contains manual examination results from the second round of analysis, specifying whether to assign a role or exclude the lobby for the streamer.
- `identified_roles.pkl`: A dictionary of identified roles for streamers from previous rounds of manual and automatic examination.

Output:

- `identified_roles.pkl`: The updated final dictionary of identified roles, reflecting the latest round of corrections based on both manual and automatic examinations.

4.5 Image_Extraction/src/evaluation.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Image-Extraction/src/evaluation.py`

What is done in the file:

- Performs an evaluation by randomly selecting a set of streamers and their corresponding lobbies for manual review of the roles identified by the automated pipeline.
- The script ensures that a selection of 10 streamer-lobby combinations is output for manual evaluation.

Input:

- `identified_roles.pkl`: A dictionary containing the previously identified roles for streamers across different lobbies.

Output:

- A printed list of 10 randomly selected streamer-lobby combinations for manual review.

5 Analysis

This is the part of the code in which I combine the extracted text data with the extracted player roles. In the thesis, this procedure is described in Chapter 5.4, namely Combination of Extracted Text Data With Extracted Player Roles. Additionally this is the part where I carry out the analysis of the final created data set, which is described in Chapter 6 of the thesis, namely Language Analysis of Players' Utterances. Here I have two basic analysis lines, classic Text Mining methods described in Chapter 6.1, and training a classifier to predict player roles based on utterances described in Chapter 6.2. The files involved in the three parts are presented separately in the following.

Order of execution for the included Python files:

1. `src/combine_role_with_transcription.py`
2. `src/describe_input_data.py`

1. `src/sentiment_analysis.py`
2. `src/pos_tag.py`
3. `src/word_frequency.py`
4. `src/lda_topic_analysis.py`

1. `src/bert_classifier.py`
2. `src/evaluate_bert_classifier_majority_vote.py`
3. `src/feature_importance.py`
4. `src/find_dead_utterances.py`
5. `src/evaluate_dead_utterances.py`

5.1 Analysis/src/combine_role_with_transcription.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/combine\protect\penalty\z@_role\protect\penalty\z@_with\protect\penalty\z@_transcription.py`

What is done in the file:

- Combines the identified roles of streamers with their corresponding speech transcriptions from the discussion rounds.
- Handles that role identification is on lobby level and speech extraction was on discussion round level.

- Deletes all transcribed utterances of lobbies for which role detection did not work.

Input:

- `../Image-Extraction/identified_roles.pkl`: A dictionary containing the roles of streamers across all lobbies, stored in the `Image-Extraction` directory.
- `../Speech-Extraction/PVAD_Application/data/final_transcriptions.pkl`: A dictionary containing the final transcriptions of discussion rounds assigned to each streamer, stored in the `Speech-Extraction/PVAD_Application/data` directory.

Output:

- `Analysis/data/transcriptions.pkl`: A dictionary where each transcription entry is associated with a role, saved in the `Analysis/data` directory.

5.2 Analysis/src/describe_input_data.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/describe\protect\penalty\z@_input\protect\penalty\z@_data.py`

What is done in the file:

- Analyzes and visualizes the input data, specifically the transcriptions of discussion rounds and their associated roles.
- Generates various plots to explore the frequency and distribution of different roles, as well as the number and length of utterances for each role.
- Performs the analysis for different precision levels, and for main and specific roles.
- Saves the analysis results as plots.

Input:

- `Analysis/data/transcriptions.pkl`: A dictionary containing the transcriptions of discussion rounds and the roles associated with them.

Output:

- `../final_plots/frequency_individual_roles.png`: A plot showing the frequency of individual roles.
- `../final_plots/frequency_general_roles.png`: A plot showing the frequency of general roles (Crewmate, Neutral, Impostor).

- ../final_plots/avg_number_utterances_low_acc.png: A plot showing the average number of utterances for each role (low accuracy level).
- ../final_plots/avg_number_utterances_mid_acc.png: A plot showing the average number of utterances for each role (mid accuracy level).
- ../final_plots/avg_number_utterances_high_acc.png: A plot showing the average number of utterances for each role (high accuracy level).
- ../final_plots/avg_length_utterances_low_acc.png: A plot showing the average length of utterances for each role (low accuracy level).
- ../final_plots/avg_length_utterances_mid_acc.png: A plot showing the average length of utterances for each role (mid accuracy level).
- ../final_plots/avg_length_utterances_high_acc.png: A plot showing the average length of utterances for each role (high accuracy level).

5.3 Analysis/src/sentiment_analysis.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/sentiment\protect\penalty\z@_analysis.py

What is done in the file:

- Performs sentiment analysis on the extracted utterances of discussion rounds using a pre-trained model from Hugging Face's Transformers library, the twitter_roberta_base_sentiment_latest model.
- Categorizes utterances into three sentiment labels: positive, neutral, or negative.
- Visualizes the results in bar charts showing the ratio of sentiment labels by role (Crewmate, Impostor, Neutral) and the mean sentiment score for each role.
- Saves the generated plots.

Input:

- Analysis/data/transcriptions.pkl: Dictionary containing the final extracted utterances of discussion rounds and the roles associated with them.

Output:

- ../final_plots/sentiment_ratio_by_role_lighter.png: A bar chart visualizing the ratio of sentiment labels (positive, neutral, negative) for each role (Crewmates, Neutrals, Impostors).
- ../final_plots/mean_sentiment_scores_by_role.png: A bar chart showing the mean sentiment score for each role.

5.4 Analysis/src/pos_tag.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/pos\protect\penalty\z@_tag.py

What is done in the file:

- Performs part-of-speech (POS) tagging on the extracted utterances of all discussion rounds based on the roles assigned to the speakers, using the NLTK library.
- Calculates the frequency of POS tags for each role category (Crewmate, Neutral, Impostor) and visualizes the results.
- Computes the differences in POS tag distributions between different roles.
- Saves final plots for analysis.
- Conducts similar analysis for specific streamers like 'ozzaworld' and 'zeroyalviking'.

Input:

- `data/transcriptions.pkl`: A dictionary containing the transcriptions of discussion rounds and the roles associated with each utterance.

Output:

- `../final_plots/pos_frequencies_crewmate.png`: Bar plot showing POS tag frequencies for Crewmates.
- `../final_plots/pos_frequencies_neutral.png`: Bar plot showing POS tag frequencies for Neutrals.
- `../final_plots/pos_frequencies_impostor.png`: Bar plot showing POS tag frequencies for Impostors.
- `../final_plots/diff_crewmate_neutral.png`: Bar plot showing differences in POS tag ratios between Crewmates and Neutrals.
- `../final_plots/diff_crewmate_impostor.png`: Bar plot showing differences in POS tag ratios between Crewmates and Impostors.
- `../final_plots/diff_neutral_impostor.png`: Bar plot showing differences in POS tag ratios between Neutrals and Impostors.
- `../final_plots/pos_frequencies_crewmate_ozzaworld.png`: Bar plot showing POS tag frequencies for Crewmates in 'ozzaworld'.
- `../final_plots/pos_frequencies_neutral_ozzaworld.png`: Bar plot showing POS tag frequencies for Neutrals in 'ozzaworld'.

- `../final_plots/pos_frequencies_impostor_ozzaworld.png`: Bar plot showing POS tag frequencies for Impostors in ‘ozzaworld‘.
- `../final_plots/diff_crewmate_neutral_ozzaworld.png`: Bar plot showing differences in POS tag ratios between Crewmates and Neutrals in ‘ozzaworld‘.
- `../final_plots/diff_crewmate_impostor_ozzaworld.png`: Bar plot showing differences in POS tag ratios between Crewmates and Impostors in ‘ozzaworld‘.
- `../final_plots/diff_neutral_impostor_ozzaworld.png`: Bar plot showing differences in POS tag ratios between Neutrals and Impostors in ‘ozzaworld‘.
- `../final_plots/pos_frequencies_crewmate_zeroyalviking.png`: Bar plot showing POS tag frequencies for Crewmates in ‘zeroyalviking‘.
- `../final_plots/pos_frequencies_neutral_zeroyalviking.png`: Bar plot showing POS tag frequencies for Neutrals in ‘zeroyalviking‘.
- `../final_plots/pos_frequencies_impostor_zeroyalviking.png`: Bar plot showing POS tag frequencies for Impostors in ‘zeroyalviking‘.
- `../final_plots/diff_crewmate_neutral_zeroyalviking.png`: Bar plot showing differences in POS tag ratios between Crewmates and Neutrals in ‘zeroyalviking‘.
- `../final_plots/diff_crewmate_impostor_zeroyalviking.png`: Bar plot showing differences in POS tag ratios between Crewmates and Impostors in ‘zeroyalviking‘.
- `../final_plots/diff_neutral_impostor_zeroyalviking.png`: Bar plot showing differences in POS tag ratios between Neutrals and Impostors in ‘zeroyalviking‘.

5.5 Analysis/src/word_frequency.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/word\protect\penalty\z@_frequency.py`

What is done in the file:

- Performs word frequency and TF-IDF analysis on the extracted utterances of discussion rounds.
- Generates visualizations of the most frequent words and words with the highest TF-IDF scores.

- Focuses on streamer 'ozzaworld' and examines differences between word frequencies in Impostor and Crewmate utterances, using TF and TF-IDF scores.
- Saves the generated plots.

Input:

- `data/transcriptions.pkl`: A dictionary containing transcriptions of discussion rounds, with each utterance labeled according to its speaker's role.
- `data/word_frequency/stop_words_english.txt`: A custom stopword list to be excluded during word frequency analysis.

Output:

- `data/word_frequency/top_20_words_overall.png`: Plot showing the top 20 most frequent words across all utterances.
- `data/word_frequency/wordcloud_overall.png`: Word cloud visualization based on overall word frequency.
- `data/word_frequency/tf_idf_overall.pkl`: A pickle file containing the calculated TF-IDF scores for each word in the corpus.
- `data/word_frequency/wordcloud_cfa_overall.png`: Word cloud visualization based on the top 100 words by TF-IDF scores.
- `data/word_frequency/ozzaworld_top_20_words_CREWMATE.png`: Plot showing the top 20 most frequent words when 'ozzaworld' is playing as Crewmate.
- `data/word_frequency/ozzaworld_top_20_words_IMPOSTOR.png`: Plot showing the top 20 most frequent words when 'ozzaworld' is playing as Impostor.
- `data/word_frequency/ozzaworld_lowest_20_word_frequencies.png`: Plot showing the 20 words with the largest negative difference in frequency between 'ozzaworld' as Crewmate and as Impostor.
- `data/word_frequency/ozzaworld_highest_20_word_frequencies.png`: Plot showing the 20 words with the largest positive difference in frequency between 'ozzaworld' as Crewmate and as Impostor.
- `data/word_frequency/ozzaworld_lowest_20_tfidf_words.png`: Plot showing the 20 words with the lowest TF-IDF score differences between 'ozzaworld' as Crewmate and as Impostor.
- `data/word_frequency/ozzaworld_highest_20_tfidf_words.png`: Plot showing the 20 words with the highest TF-IDF score differences between 'ozzaworld' as Crewmate and as Impostor.

5.6 Analysis/src/lda_topic_analysis.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/lda\protect\penalty\z@_topic\protect\penalty\z@_analysis.py

What is done in the file:

- Performs topic modeling on transcriptions of discussion rounds using Latent Dirichlet Allocation (LDA).
- Generates visualizations of topic distances, word clouds for each topic, and calculates the coherence score.
- Saves resulting plots.

Input:

- `data/transcriptions.pkl`: Dictionary containing transcriptions of discussion rounds, where each utterance is labeled according to its speaker's role.

Output:

- `topic_distances.png`: A plot showing the distances between the three topics identified by the LDA model.
- `wordcloud_topic_1.png`, `wordcloud_topic_2.png`, `wordcloud_topic_3.png`: Word cloud visualizations showing the top 10 words for each topic.

5.7 Analysis/src/bert_classifier.py

File: https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/bert\protect\penalty\z@_classifier.py

What is done in the file:

- Fine-tunes a BERT-based model to classify extracted utterances from discussions into two categories: `CREWMATE` and `IMPOSTOR`.
- Performs cross-validation with oversampling of the minority class (Impostor) to handle data imbalance.
- Trains the model and evaluates its performance for each fold of the cross-validation.
- Saves evaluation results, confusion matrices, and training logs.

Input:

- `data/transcriptions.pkl`: A dictionary containing transcriptions of discussion rounds with labels and precision modes (low, mid, high).

Output:

- `data/bert_classifier/[accuracy_mode]/test_trainer_fold_{fold}`: The best model and tokenizer for each fold.
- `data/bert_classifier/[accuracy_mode]/evaluation_results_fold_{fold}.txt`: Evaluation results for each fold, including metrics such as accuracy.
- `data/bert_classifier/[accuracy_mode]/confusion_matrix_fold_{fold}.png`: Confusion matrix plots showing the performance of the model on classifying `CREWMATE` and `IMPOSTOR`.
- `data/bert_classifier/[accuracy_mode]/trainlog.txt`: Log file containing information about the training process, including the epoch results for each fold.

5.8 Analysis/src/evaluate_bert_classifier_majority_vote.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/evaluate\protect\penalty\z@_bert\protect\penalty\z@_classifier\protect\penalty\z@_majority\protect\penalty\z@_vote.py`

What is done in the file:

- Evaluates the BERT classifier's performance by applying a majority voting mechanism to classify all utterances of a discussion round.
- Evaluates discussions with a minimum of three utterances.
- Performs evaluations for all five classifiers trained using 5-fold cross-validation.
- Saves the results, confusion matrices, and evaluation logs.

Input:

- `data/transcriptions.pkl`: A dictionary containing transcriptions of discussion rounds with their respective accuracy modes (low, mid, high).
- `data/bert_classifier/[accuracy_mode]/test_indices_fold_{fold}.json`: The test indices for each fold.
- `data/bert_classifier/[accuracy_mode]/best_model_fold_{fold}`: The best model and tokenizer for each fold, used for predicting discussion rounds.

Output:

- `data/majority_vote/confusion_matrix_majority_vote_{accuracy_mode}_fold_{fold}.png`: Confusion matrix plots showing the performance of the majority vote predictions for each fold and accuracy mode.
- `data/majority_vote/filtered_confusion_matrix_majority_vote_{accuracy_mode}_fold_{fold}.png`: Confusion matrix plots for discussion rounds with at least 3 utterances.
- `data/majority_vote/results_majority_vote.txt`: Log file containing evaluation metrics such as accuracy, confusion matrices, and filtered evaluation metrics.

5.9 Analysis/src/feature_importance.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/feature\protect\penalty\z@_importance.py`

What is done in the file:

- Uses Integrated Gradients to compute and visualize feature importance (token attributions) for utterances classified as `CREWMATE` or `IMPOSTOR`.
- Analyzes the average attributions of tokens over all utterances to identify the most influential tokens for each class.
- Visualizes the attributions for selected utterances.
- Analyzes question mark token attributions.
- Saves the final token attributions and plots.

Input:

- `data/transcriptions.pkl`: Dictionary containing the transcriptions of discussion rounds, categorized by their precision levels.
- `data/bert_classifier/low_acc/best_model_fold_1`: The pre-trained BERT model for the `low_acc` mode from fold 1, used to calculate token attributions.

Output:

- `data/feature_importance_consistent_plots/average_token_attributions_crewmate.pkl`: A dictionary containing the average attributions of tokens for the `CREWMATE` class.
- `data/feature_importance_consistent_plots/average_token_attributions_impostor.pkl`: A dictionary containing the average attributions of tokens for the `IMPOSTOR` class.

- `data/feature_importance_consistent_plots/top_tokens_crewmate_by_average_attribution.png`: A bar plot showing the top 20 tokens by average attribution for the `CREWMATE` class.
- `data/feature_importance_consistent_plots/bottom_tokens_crewmate_by_average_attribution.png`: A bar plot showing the bottom 20 tokens by average attribution for the `CREWMATE` class.
- `data/feature_importance_consistent_plots/top_tokens_impostor_by_average_attribution.png`: A bar plot showing the top 20 tokens by average attribution for the `IMPOSTOR` class.
- `data/feature_importance_consistent_plots/bottom_tokens_impostor_by_average_attribution.png`: A bar plot showing the bottom 20 tokens by average attribution for the `IMPOSTOR` class.
- `data/feature_importance_consistent_plots/crewmate_attribution_{i}.png`: Visualized token attributions for 50 random `CREWMATE` utterances.
- `data/feature_importance_consistent_plots/impostor_attribution_{i}.png`: Visualized token attributions for 50 random `IMPOSTOR` utterances.

5.10 Analysis/src/find_dead_utterances.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/find\protect\penalty\z@_dead\protect\penalty\z@_utterances.py`

What is done in the file:

- Identifies discussion rounds where the probability that a player who made an utterance is already dead is maximized.
- Selects 30 discussion rounds with the highest probability of having a dead streamer for manual examination.
- Extracts the respective segments from corresponding video files using `ffmpeg`.
- Saves extracted video segments for manual review.

Input:

- `data/transcriptions.pkl`: A dictionary containing the extracted utterances of discussion rounds categorized by precision modes.
- `data/bert_classifier/low_acc/test_indices_fold_1.json`: A file containing indices of the test utterances for fold 1 in the low accuracy mode.

- `../Lobby-Synchronization/data/final_synchronization_output/final_lobby_times.pkl`: A file containing the start and end times of lobbies, used to extract relevant video segments.
- `../../pop520978/data/`: Directory containing the video files from which segments will be extracted.

Output:

- `data/test_discussion_rounds_low_acc_fold_1.txt`: A text file containing unique discussion rounds from the low precision mode, fold 1.
- `data/evaluate_dead_videos/{disc_round}.mkv`: Extracted video segments for each selected discussion round.

5.11 Analysis/src/evaluate_dead_utterances.py

File: `https://gitlab.inf.uni-konstanz.de/tim.kleinlein/among-us-analysis/-/blob/main/Analysis/src/evaluate\protect\penalty\z@_dead\protect\penalty\z@_utterances.py`

What is done in the file:

- Evaluates the utterances from identified discussion rounds where players were confirmed dead.
- Classifies these utterances using the BERT classifier and evaluates the model's accuracy.
- Generates a confusion matrix for the classification performance on `CREWMATE` vs `IMPOSTOR` utterances.
- Saves evaluation results and the confusion matrix.

.

Input:

- `data/transcriptions.pkl`: A dictionary containing the transcriptions of discussion rounds categorized by accuracy modes.
- `data/bert_classifier/low_acc/best_model_fold_1`: The pre-trained BERT model fine-tuned for sequence classification from low precision mode and fold 1.

Output:

- `data/evaluate_dead_videos/evaluation_results.txt`: A text file containing the evaluation results, including accuracy metrics.
- `data/evaluate_dead_videos/confusion_matrix.png`: A confusion matrix visualizing the classification results for CREWMATE vs IMPOSTOR utterances.