Culturally Sensitive Social Robotics
for Africa

D4.3.2 Speech Event

Due date: **1/10/2024**
Submission Date: **20/02/2025**
Revision Date: **n/a**

Start date of project: **01/07/2023** Duration: **36 months**

Lead organisation for this deliverable: **Carnegie Mellon University Africa**

Responsible Person: **Clifford Onyonka** Revision: **1.0**

Project funded by the African Engineering and Technology Network (Afretec) Inclusive Digital Transformation Research Grant Programme		
Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants (including Afretec Administration)	
RE	Restricted to a group specified by the consortium (including Afretec Administration)	
CO	Confidential, only for members of the consortium (including Afretec Administration)	

Executive Summary

Deliverable D4.3.2 concerns the results of Task 4.3.2, a task whose objective was to train, test, and finally deploy a speech-to-text model on a ROS node that will enable speech utterances in Kinyarwanda and English languages captured by Pepper's microphones to be transcribed into written text.

This report details the output of each phase of the software development process used in the fulfillment of Deliverable D4.3.2. The requirements definition section specifies the functional requirements of the users of Speech Event, the module specification section outlines the functional characteristics of Speech Event, the interface design section outlines the specification of the inputs and outputs of Speech Event, the module design section describes the deep neural networks that perform speech recognition of Kinyarwanda and English utterances, the testing section shows the results and descriptions of unit and end-to-end tests of Speech Event, and the user manual section outlines how to build and run Speech Event.

Contents

1 Introduction **4**

2 Requirements Definition **5**

3 Module Specification **6**

4 Interface Design **7**
- 4.1 Source Code . . . 7
- 4.2 Configuration Files . . . 8
- 4.3 Topics Subscribing To . . . 10
- 4.4 Topics Publishing To . . . 10
- 4.5 Launch Files . . . 10

5 Module Design **11**
- 5.1 Model Architecture . . . 11
  - 5.1.1 Preprocessor . . . 11
  - 5.1.2 Encoder Architecture . . . 11
  - 5.1.3 Decoder/Predictor Architecture . . . 12
  - 5.1.4 Joint Network Architecture . . . 13
  - 5.1.5 Combined Model Architecture . . . 14

6 Testing Report **15**
- 6.1 Unit Testing . . . 15
- 6.2 End-to-end Testing . . . 18

7 User Manual **19**
- 7.1 Installation . . . 19
- 7.2 Usage . . . 19
- 7.3 Graphical User Interface . . . 20
- 7.4 Driver ROS Node . . . 20

References **21**

Principal Contributors **22**

Document History **23**

1 Introduction

Conversational systems are developed using the following main components: automatic speech recognition (speech-to-text), natural language understanding, dialogue manager, natural language generation, and text-to-speech. Speech-to-text converts an audio signal containing spoken speech utterances to written text transcriptions, natural language understanding analyses the transcribed text to extract meaning, the dialogue manager generates a response based on the inferred meaning of the text, natural language generation formulates text based on the response generated by the dialogue manager, and text-to-speech synthesises spoken speech utterances to be played to the other agent(s) in the conversation cycle [1]. Such a system is displayed in Fig 1.

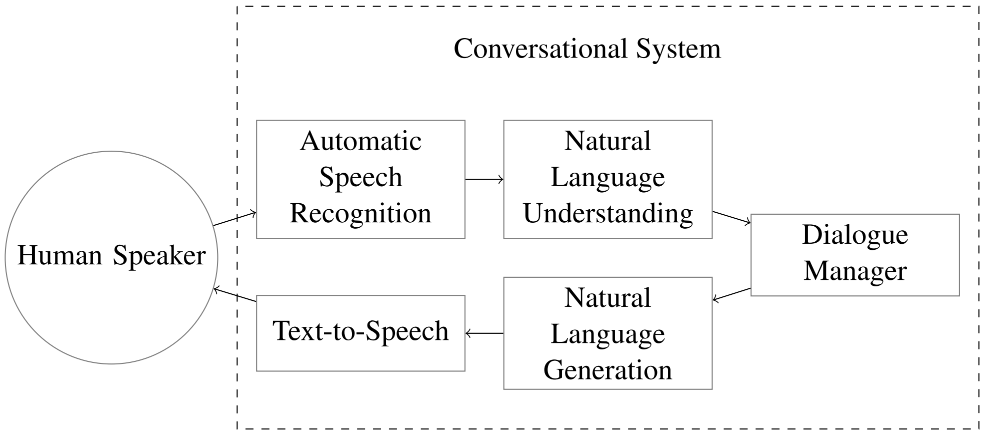

Conversational System
Human Speaker
Automatic Speech Recognition
Natural Language Understanding
Dialogue Manager
Natural Language Generation
Text-to-Speech

Figure 1: A diagrammatic representation of a conversational system [1]

The Speech Event module only handles one component of the conversational system described in the previous paragraph, that is, speech-to-text (which is referred to as automatic speech recognition in Fig 1). It acquires audio signals published on the `/soundDection/signal` ROS topic, and then passes these audio signals through a deep neural network that transcribes speech utterances contained within the audio to generate text strings representations, which are then published on the `/speechEvent/text` ROS topic. Kinyarwanda and English are the two languages that are transcribed by Speech Event, with the choice of language set via a configuration option in a configuration file.

2 Requirements Definition

A running Speech Event ROS node performs one main function - Kinyarwanda and English speech recognition. An audio signal is received via the `/soundDetection/signal` ROS topic, and Speech Event transcribes any speech utterances detected in the signal to either Kinyarwanda or English.

In order to successfully perform this function, the following functional requirements need to be fulfilled by Speech Event:

1. Acquire an audio signal from the `/soundDetection/signal` ROS topic
2. Pass the audio signal through an automatic speech recognition (ASR) model to transcribe any speech utterances the audio signal contains to either Kinyarwanda or English text
3. Publish the transcribed text to the `/speechEvent/text` ROS topic

The exact language between Kinyarwanda and English which a running Speech Event ROS node will transcribe is decided based on a configuration option that is set in a configuration file that is loaded at the initialisation stage when the ROS node is started.

To supplement the core requirement that Speech Event performs (Kinyarwanda and English speech recognition), the text transcriptions that Speech Event will transcribe will be displayed on a graphical interface. The graphical interface will be used as a replacement for the terminal interface, especially in presentation settings where displaying text transcriptions on the terminal is impractical due to the small font size used by the terminal and the extra verbosity of the results displayed on the terminal.

A ROS node that emulates Sound Detection will also be developed, and its main purpose will be to showcase the working of Speech Event in isolation, separated from the rest of the CSSR4Africa system and from the Pepper robot. This ROS node will capture audio from the computer on which Speech Event will be running on, and publishing the captured audio to the same ROS topic that Sound Detection publishes to (`/soundDetection/signal`), therefore mimicking Sound Detection's operation of acquiring audio from Pepper and publishing it to the aforementioned ROS topic.

It is also worth noting that Sound Detection may fail, and since the Behaviour Controller does not directly interface with Sound Detection, the Speech Event ROS node bears the burden of signaling failures that may arise in Sound Detection to the Behaviour Controller. This is done by publishing the message 'Error: soundDetection is down' on the `/speechEvent/text` ROS topic.

3 Module Specification

A running Speech Event ROS node performs speech-to-text in real-time, acquiring an audio signal from the `/soundDetection/signal` ROS topic, generating a text description of the acquired audio, and then publishing transcribed text to the `/speechEvent/text` ROS topic as soon as speech utterances are detected in the audio signal. The data that is input to Speech Event, the transformation that this data undergoes, and the data that Speech Event outputs is described below:

1. An audio signal published by Sound Detection to the /soundDetection/signal ROS topic is acquired by Speech Event
2. The acquired audio signal is then passed through an ASR model that transcribes speech utterances present within the audio signal to text strings
3. The text strings output by the ASR model are then published to the `/speechEvent/text` ROS topic for the Behaviour Controller to use

This process, and the position of Speech Event within the CSSR4Africa system architecture, is captured in Fig 2.

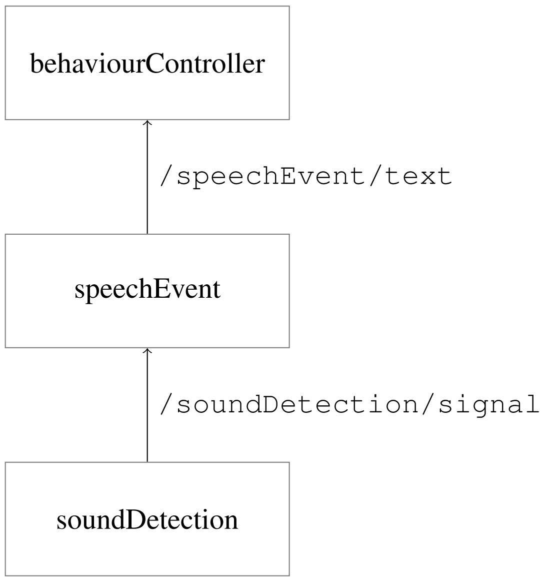


Figure 2: Speech Event within the CSSR4Africa architecture

4 Interface Design

4.1 Source Code

The file structure of Speech Event is as shown below:

```
speech_event/
├── config/
│   ├── speech_event_configuration.ini
│   ├── speech_event_driver.ini
│   └── speech_event_tests.ini
├── gui/
│   └── __main__.py
├── launch/
│   ├── speech_event_launch_robot.launch
│   └── speech_event_launch_test_harness.launch
├── src/
│   ├── speech_event/
│   │   ├── __init__.py
│   │   ├── speech_event_config_parser.py
│   │   └── speech_event_implementation.py
│   ├── speech_event_application.py
│   ├── speech_event_driver.py
│   └── speech_event_gui.py
├── tests/
│   ├── data/
│   │   ├── audio.wav
│   │   └── config.ini
│   ├── __init__.py
│   ├── test_config_parser.py
│   └── test_speech_event.py
├── .gitignore
├── CMakeLists.txt
├── package.xml
├── README.md
├── requirements.txt
└── setup.py
```

The `config/` directory houses configuration files that are used to configure different ROS nodes that are part of Speech Event:

1. `speech_event_configuration.ini`: configurations for the main Speech Event ROS topic
2. `speech_event_driver.ini`: configurations for a Speech Event driver ROS node that mimics Sound Detection, but captures audio from a personal computer's microphones rather

than from Pepper's microphones

3. `speech_event_tests.ini`: configurations for a ROS node that is used when running Speech Event unit and end-to-end tests

The `gui/` directory holds a python script that enables running as a python module the GUI interface that is used to monitor the `/speechEvent/text` ROS topic. When the current working directory is the Speech Event directory, the GUI interface can be run as a python module as follows: `python3 -m gui`.

The `launch/` directory holds ROS launch files that are used to bringing up either the Speech Event system or the testing system using a single terminal command:

1. `speech_event_launch_robot.launch`: used to bring up the whole Speech Event system using a single terminal command

2. `speech_event_launch_test_harness.launch`: used to bring up the whole testing system using a single terminal command

The `src/` directory holds all the python source code files that implement the Speech Event system together with the GUI application and driver application. Within the `src/` directory is the `speech_event/` subdirectory. This subdirectory is used by the `setup.py` script to configure Speech Event as a python package that can be easily installed by CATKIN when running the `catkin_make` command.

The `tests/` directory contains unit tests and end-to-end tests for Speech Event. Unit tests are used to test individual components of Speech Event in isolation, while end-to-end tests are used to test all components of Speech Event working together in unison to obtain an audio signal from the `/soundDetection/signal`, transcribe it, and then publish the generated text to the ROS topic `/speechEvent/text`. In essence, end-to-end tests in this case are integration tests that test all the components of Speech Event working together. The `data/` subdirectory in the `tests/` directory holds data that is used as test fixtures for the unit tests and end-to-end tests.

The file `.gitignore` lists directories and files that are to be excluded from git, `CMakeLists.txt` and `package.xml` are required by CATKIN to denote Speech Event as a ROS package together with all packages that Speech Event depends on, `README.md` contains documentation of how to use Speech Event, and `requirements.txt` contains a list of versioned Python packages that Speech Event depends on.

4.2 Configuration Files

A running Speech Event ROS node requires to be configured prior to starting it, a task that is accomplished via a configuration file that is stored as plain text. The configuration parameters stored in a configuration file as displayed in Table 1.

Key	Value	Description
`language`	`kinyarwanda` or `english`	The language of speech utterances to be ingested by ASR models
`verbose_mode`	`true` or `false`	Whether to print informational messages to the terminal
`rw_model_path`	`<string>`	Path to the Kinyarwanda ASR model
`en_model_path`	`<string>`	Path to the English ASR model
`audio_storage_dir`	`<string>`	Path to directory that will store audio captured from `/soundDetection/signal` ROS topic
`cuda`	`true` or `false`	Whether to use GPU or CPU for running model inference (True means use GPU, False means use CPU)
`sample_rate`	`<integer>`	The sample rate of audio signals captured from the `/soundDetection/signal` ROS topic

Table 1: Speech Event configuration file options

A running driver ROS node also requires to be configured prior to starting it. Its configuration parameters are stored in a configuration file as displayed in Table 2. (A driver ROS node mimics Sound Detection, capturing audio from a personal computer's microphone instead of from Pepper, and publishing the audio signal to `/soundDetection/signal` topic in the same way as Sound Detection does).

Key	Value	Description
`channels`	`<integer>`	Number of output channels for the audio captured from the personal computer's microphones
`chunk_size`	`<integer>`	Number of audio samples to be read per iteration from the personal computer's microphones
`sample_rate`	`<integer>`	Sample rate to use when capturing audio
`speech_amplitude_threshold`	`<float>`	Threshold for segmenting silence and non-silence regions in the captured audio (non-silence regions are assumed to contain speech utterances)
`utterance_time_buffer`	`<integer>`	Number of padding samples around non-silence audio regions

Table 2: Driver configuration file options

4.3 Topics Subscribing To

A running Speech Event ROS node acquires audio signals from the `/soundDetection/signal` ROS topic. Table 3 shows this fact, while also mentioning the ROS node that publishes these audio signals.

Topic	Node	Platform
`/soundDetection/signal`	`soundDetection`	`Physical robot`

Table 3: Topics subscribing to

4.4 Topics Publishing To

The Speech Event ROS node publishes transcribed text to the `/speechEvent/text` ROS topic, publishing each time a transcription process completes. The published text transcriptions are of the `string` data type. Table 4 shows the ROS topic that Speech Event publishes to.

Topic	Node	Platform
`/speechEvent/text`	`behaviourController`	`Physical robot`

Table 4: Topics publishing to

4.5 Launch Files

Two launch files are defined in Speech Event's source code:

1. `speech_event_launch_robot.launch`: used to bring up the whole Speech Event system when performing speech transcription
2. `speech_event_launch_test_harness.launch`: used to bring up the whole testing system for Speech Event when running Speech Event tests

5 Module Design

5.1 Model Architecture

Both the Kinyarwanda and English speech recognition processes performed by Speech Event rely on deep learning models. Both of these deep learning models are based on the conformer transducer architecture, and the specific models that are used by Speech Event were acquired from NVIDIA's Nemo catalog of models.

The conformer transducer architecture is made up of a conformer encoder and a transducer decoder. The conformer encoder combines transformers and Convolution Neural Networks (CNNs) in an attempt to utilise the strengths of both, with transformers excelling at capturing global dependencies of an audio signal and CNNs excelling at capturing local dependencies of an audio signal [2]. The transducer decoder, on the other hand, makes use of Recurrent Neural Networks (RNNs) to form an autoregressive model whose next token prediction is conditioned on the previously predicted tokens.

5.1.1 Preprocessor

When an audio signal is received, it is first passed through a preprocessor block that extracts filterbank features from the audio signal. For this process, an audio signal is first converted to a spectrogram by a Short Time Fourier Transform (STFT), and then a series of filters are applied to the resultant spectrogram to obtain the filterbank features that each represent the magnitude of the energy within a distinct frequency band. This preprocessor block is visualised in Fig 3.

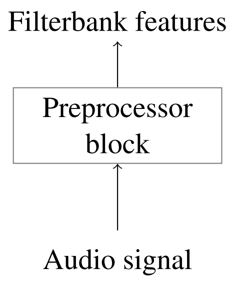


Figure 3: Audio to filterbank features preprocessor

5.1.2 Encoder Architecture

The filterbank features obtained from the preprocessor block are then passed to the encoder part of the model. The encoder transforms acoustic features in the original audio signal captured within the filterbank features into latent features that better represent the speech in the original audio signal.

A conformer encoder is shown in Fig 4. The encoder used in Speech Event has 17 conformer blocks.

The filterbank features are first passed through a SpecAugment block. SpecAugment is a data augmentation method that is suitable for end-to-end speech recognition models such as the Conformer Tranducer model used by Speech Event. "The augmentation policy consists of warping the features, masking blocks of frequency channels, and masking blocks of time steps" [3].

The features are then processed by a convolution subsampling block, dropout applied, and then passed through the series of 17 conformer blocks in the encoder.

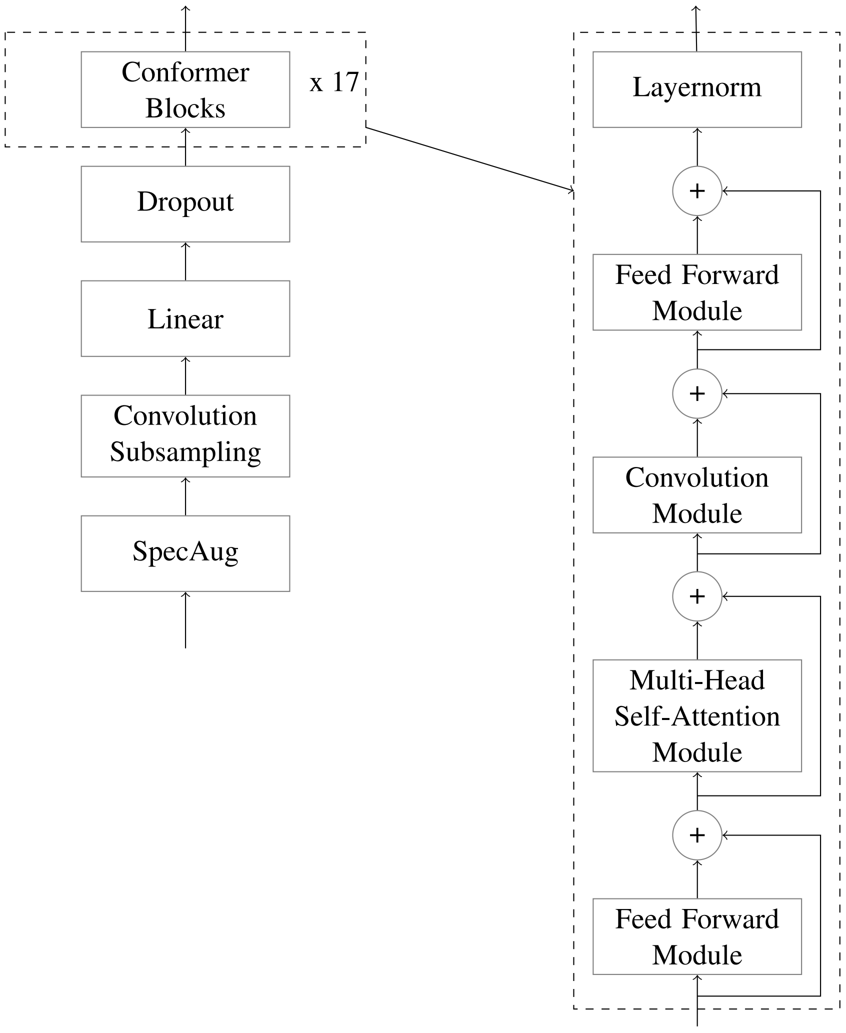


Figure 4: Conformer encoder architecture [2]

Each confomer block consists of a multi-head self-attention module and a convolution module collectively sandwiched between two feedforward modules [2]. The self-attention module is more adept at capturing global feature dependencies while the convolution module is more adept at capturing local feature dependencies, thereby capturing the semantics of speech utterances much better than models that only specialise in capturing only either of the global or local feature dependencies.

5.1.3 Decoder/Predictor Architecture

Keeping in line with Gulati et al., a single LSTM layer is used in the decoder [2]. An embedding layer converts input tokens into vectorised representations that are then passed to the LSTM layer. Additionally, drop out is applied both in the LSTM layer and on the output of the LSTM layer.

The decoder architecture used in Speech Event is shown in Fig 5.

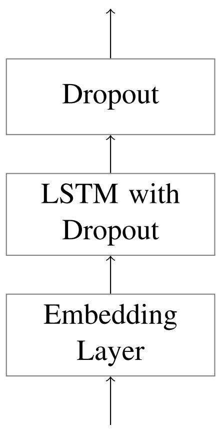


Figure 5: Transducer predictor architecture

5.1.4 Joint Network Architecture

The joint network combines the output of the encoder and the output of the decoder in order to predict the text in the speech utterances contained in the original sound signal passed to the encoder. The combined outputs are then passed through a ReLU activation, dropout applied, and lastly passed through a linear layer.

The joint network architecture used in Speech Event is shown in Fig 6.

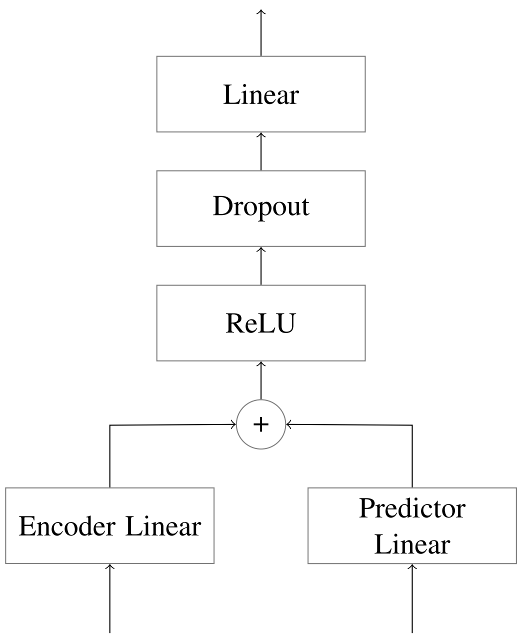


Figure 6: Joint architecture

5.1.5 Combined Model Architecture

The four different sections of the full conformer transducer architecture discussed above are combined together to form the whole network that performs end-to-end automatic speech recognition. This full model's architecture is summaried in Fig 7.

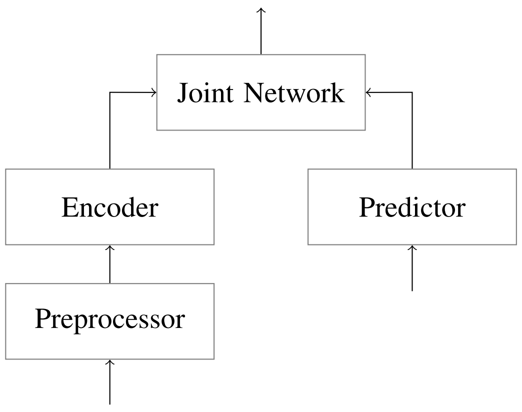


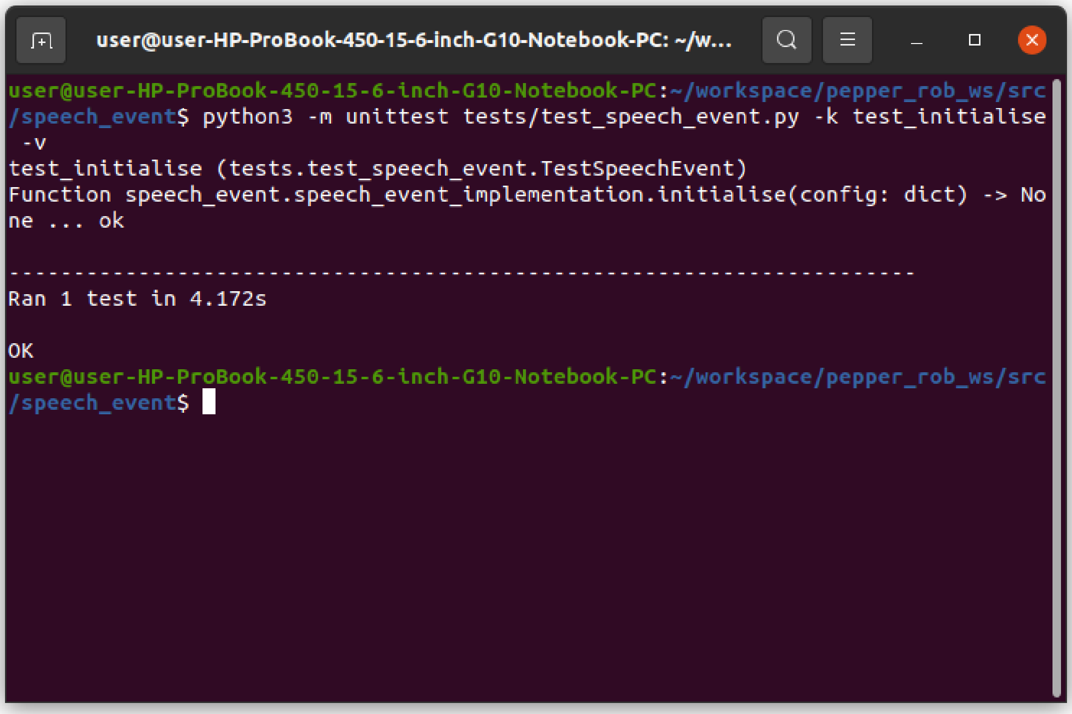
Figure 7: Combined model architecture

6 Testing Report

The `tests/` directory contains unit tests and end-to-end tests. The unit tests test in isolation functions that implement Speech Event, while the end-to-end tests test the whole Speech Event as a whole (from when audio signals are captured from the `/soundDetection/signal` ROS topic to when transcribed text is published on the `/speechEvent/text` ROS topic). The unittest python testing framework is used for these tests, meaning that to run the tests the command `python3 -m unittest` is used (the `-v` flag is used when increased verbosity is required). The unittest package is part of the standard python library, and therefore no extra packages need to be installed when running tests. The end-to-end tests provided for Speech Event require a ROS master node to be running, and therefore `roscore` needs to be run in a separate terminal before running tests.

6.1 Unit Testing

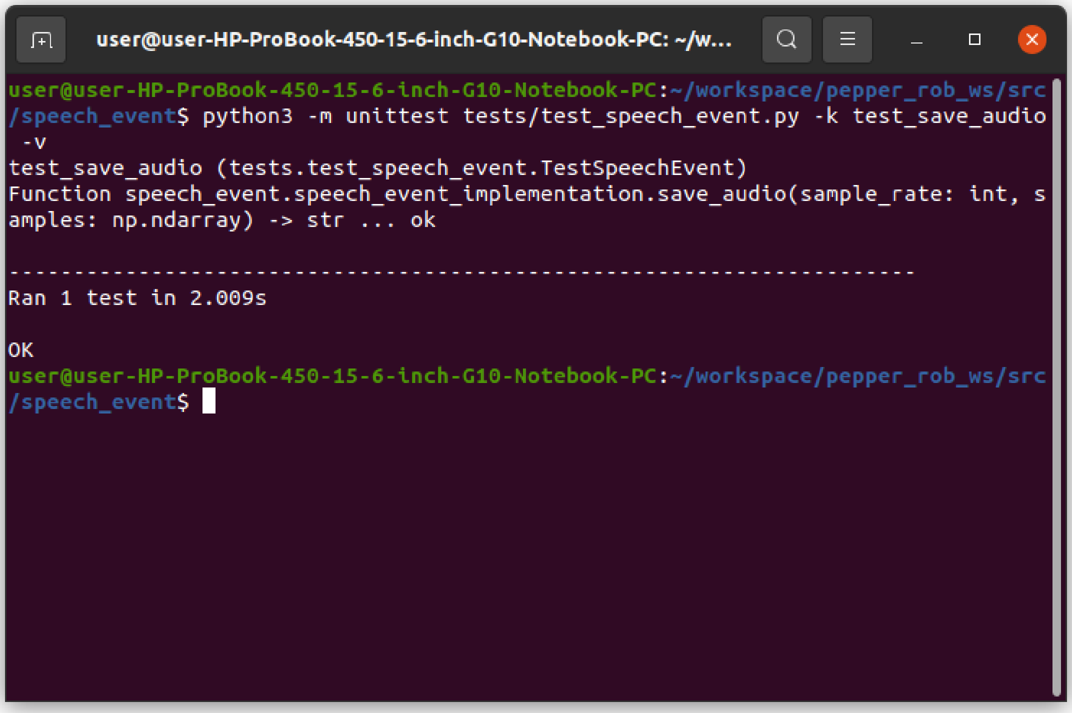
Four unit tests are provided in Speech Event. The first unit test tests the initialisation function (the function that, at the start of a Speech Event ROS node, reads a configuration file and sets up Speech Event to operate with required settings such as the language to be transcribed). Fig 8 shows the results of running this test.



```
user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC:~/workspace/pepper_rob_ws/src
/speech_event$ python3 -m unittest tests/test_speech_event.py -k test_initialise
 -v
test_initialise (tests.test_speech_event.TestSpeechEvent)
Function speech_event.speech_event_implementation.initialise(config: dict) -> No
ne ... ok

----------------------------------------------------------------------
Ran 1 test in 4.172s

OK
user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC:~/workspace/pepper_rob_ws/src
/speech_event$
```

Figure 8: Test report for Speech Event's initialisation function

The second unit test tests the save audio function (this function saves an audio signal to storage as a .wav file, and this wav file is then passed to the ASR model that transcribes speech in the next phase of the speech-to-text process). Fig 9 shows the results of running this test.

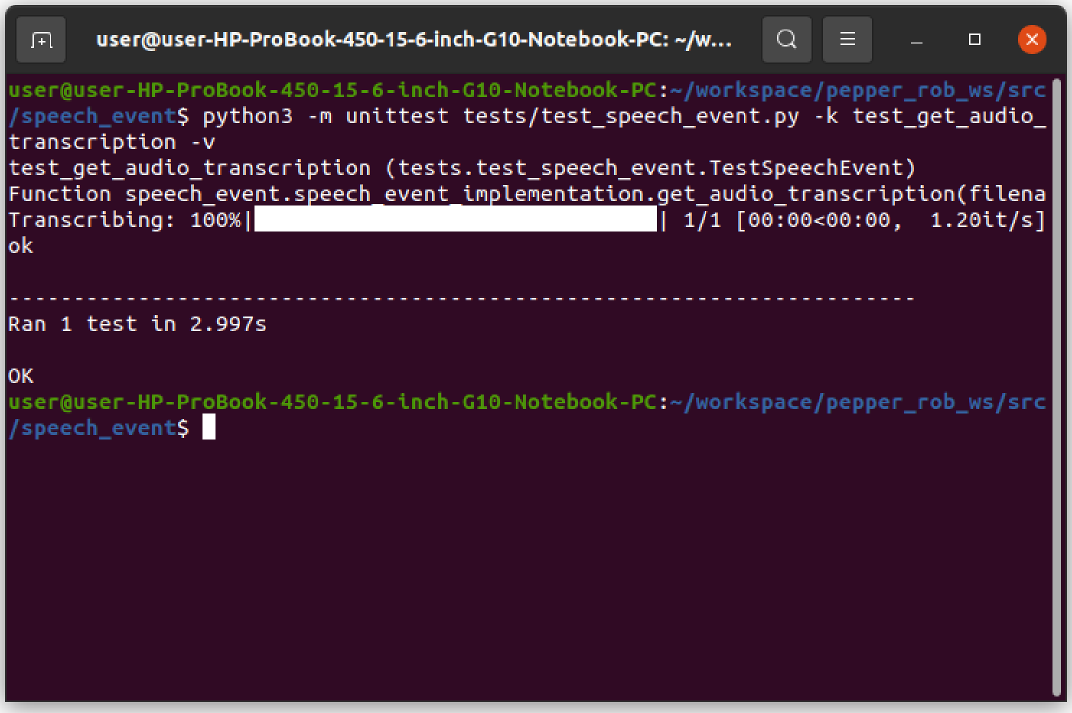


user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC: ~/w...

```
user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC:~/workspace/pepper_rob_ws/src
/speech_event$ python3 -m unittest tests/test_speech_event.py -k test_save_audio
 -v
test_save_audio (tests.test_speech_event.TestSpeechEvent)
Function speech_event.speech_event_implementation.save_audio(sample_rate: int, s
amples: np.ndarray) -> str ... ok

----------------------------------------------------------------------
Ran 1 test in 2.009s

OK
user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC:~/workspace/pepper_rob_ws/src
/speech_event$ 
```

Figure 9: Test report for Speech Event's save audio function

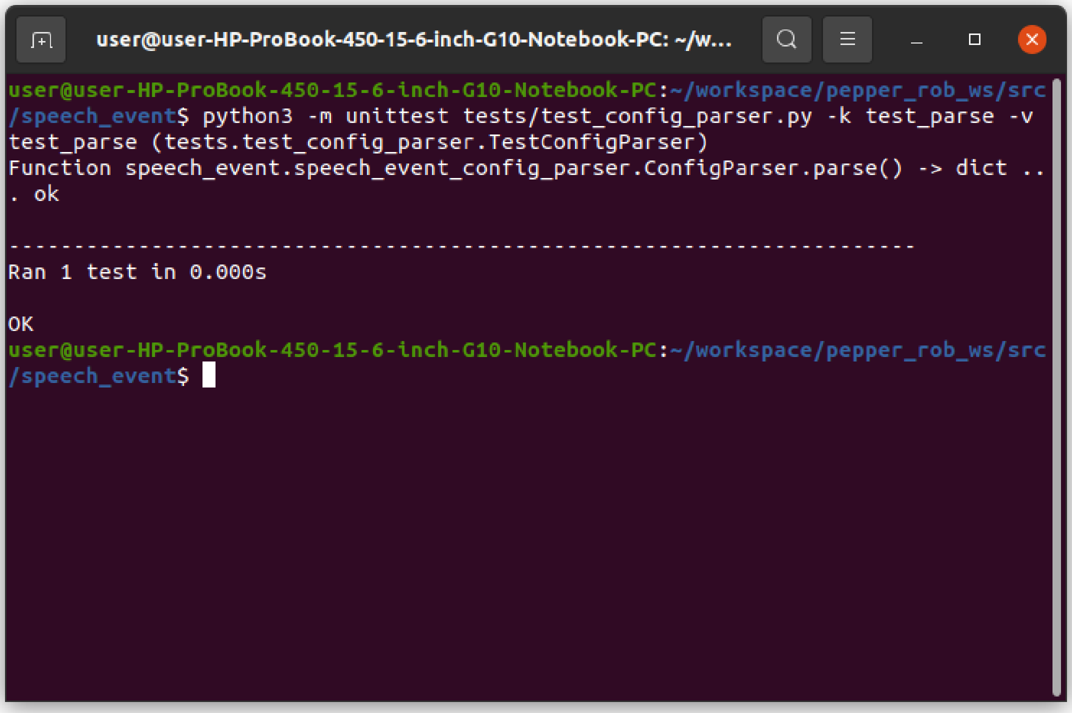
The third unit test tests the get audio transcription function (this function uses an ASR deep learning model to transcribe an audio signal, generating a text string representation of any speech utterances contained within the audio signal). Fig 10 shows the results of running this test.


user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC: ~/w...

```
user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC:~/workspace/pepper_rob_ws/src
/speech_event$ python3 -m unittest tests/test_speech_event.py -k test_get_audio_
transcription -v
test_get_audio_transcription (tests.test_speech_event.TestSpeechEvent)
Function speech_event.speech_event_implementation.get_audio_transcription(filena
Transcribing: 100%|██████████████████████████████████| 1/1 [00:00<00:00,  1.20it/s]
ok

----------------------------------------------------------------------
Ran 1 test in 2.997s

OK
user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC:~/workspace/pepper_rob_ws/src
/speech_event$ 
```

Figure 10: Test report for Speech Event's get audio transcription function

The last unit test tests the parse function (this function parses an ini configuration file and generates its python dictionary representation, and this function is re-used everywhere an ini configuration file needs to be read and parsed - the ini specification used by CSSR4Africa does not follow the standardised ini file specification, and therefore cannot be parsed by standard ini libraries provided by the standard python library, which necessitates creation of a custom ini file parser). Fig 11 shows the results of running this test.

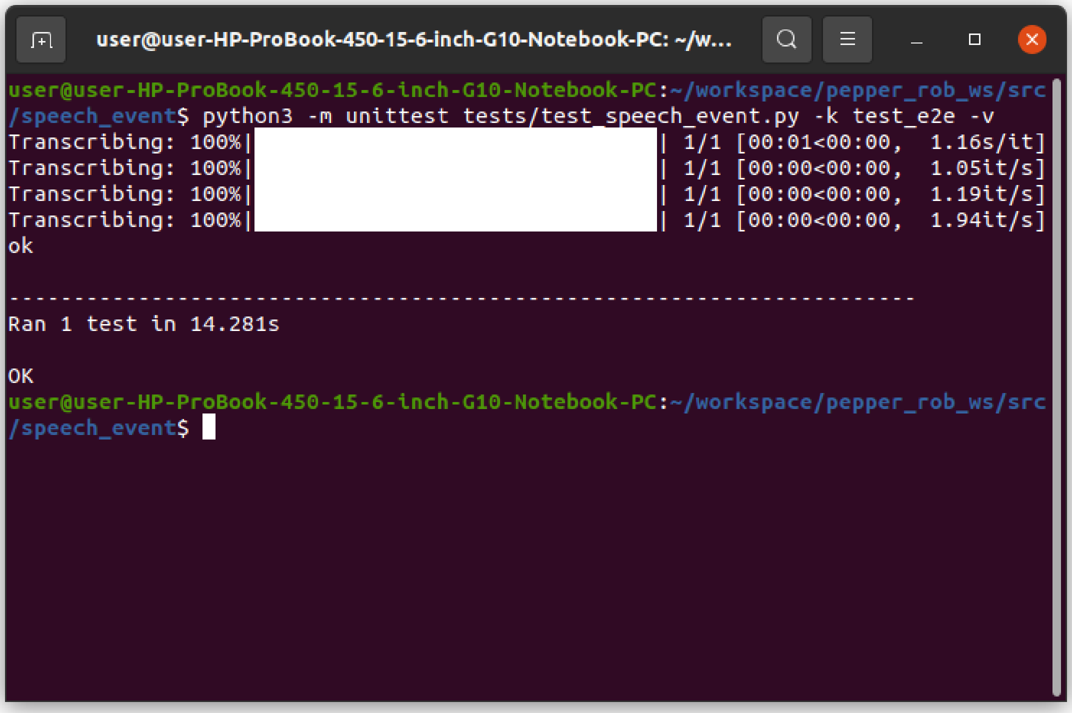

user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC: ~/w...

```
user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC:~/workspace/pepper_rob_ws/src
/speech_event$ python3 -m unittest tests/test_config_parser.py -k test_parse -v
test_parse (tests.test_config_parser.TestConfigParser)
Function speech_event.speech_event_config_parser.ConfigParser.parse() -> dict ..
. ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC:~/workspace/pepper_rob_ws/src
/speech_event$ 
```

Figure 11: Test report for Speech Event's parse function

6.2 End-to-end Testing

One end-to-end test is provided in Speech Event. A running ROS master node is needed to run this test because this test necessitates creation of a full-fledged ROS node when it is run, and therefore `roscore` needs to be run on a separate terminal before running this test. It tests the whole Speech Event ROS node as a unit, from the capture of audio signals from the `/soundDetection/signal` ROS topic to the final step of publishing transcribed text to the `/speechEvent/text` ROS topic. Fig 12 shows the results of running this test.



```
user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC:~/workspace/pepper_rob_ws/src
/speech_event$ python3 -m unittest tests/test_speech_event.py -k test_e2e -v
Transcribing: 100%|██████████████████████████████| 1/1 [00:01<00:00,  1.16s/it]
Transcribing: 100%|██████████████████████████████| 1/1 [00:00<00:00,  1.05it/s]
Transcribing: 100%|██████████████████████████████| 1/1 [00:00<00:00,  1.19it/s]
Transcribing: 100%|██████████████████████████████| 1/1 [00:00<00:00,  1.94it/s]
ok

----------------------------------------------------------------------
Ran 1 test in 14.281s

OK
user@user-HP-ProBook-450-15-6-inch-G10-Notebook-PC:~/workspace/pepper_rob_ws/src
/speech_event$ 
```

Figure 12: Test report for Speech Event's parse function

7 User Manual

7.1 Installation

The Speech Event package needs to be installed before it can be used. To install Speech Event, clone the Speech Event repository to the CSSR4Africa workspace, and then proceed with the following steps:

1. Install required packages
   (a) Install required Ubuntu packages: `sudo apt-get install cython3 ffmpeg gfortran libopenblas-dev libopenblas64-dev patchelf pkg-config python3-testresources python3-typing-extensions sox`
   (b) Install required python packages: `pip3 install -r requirements.txt` (the `requirements.txt` file is located in the Speech Event root directory)
2. Install Speech Event: `roscd && cd ../ && catkin_make`

7.2 Usage

To run a Speech Event ROS node, a Sound Detection ROS node needs to also be running (or a Speech Event driver node, which is described in a subsequent subsection).

1. Run a Speech Event ROS node: `rosrun speech_event speech_event_application.py -c CONFIG` (where CONFIG is the path to a configuration ini file; an editable configuration file is provided in the `config/` directory)

2. View text transcriptions on the terminal (optional): `rostopic echo /speechEvent/text`

7.3 Graphical User Interface

A graphical interface is provided to display the text transcriptions that are being published on the `/speechEvent/text` ROS topic on a more user friendly interface compared to the terminal interface. To view text transcriptions on this graphical interface:

1. Install Tk: `sudo apt-get install python3-tk`

2. Run the GUI application: `rosrun speech_event speech_event_gui.py` (or `python3 -m gui` if the current working directory is set to the Speech Event root directory)

7.4 Driver ROS Node

A driver ROS node that mimics the Sound Detection ROS node is also provided. It capturing audio from a personal computer's microphone instead of from Pepper, and publishes the audio signal on the `/soundDetection/signal` ROS topic in the same way that the Sound Detection ROS node does. To bring up this driver ROS node:

1. Install required Ubuntu packages: `sudo apt-get install libasound-dev portaudio19-dev libportaudio2 libportaudiocpp0 libav-tools`

2. Run a driver ROS node: `rosrun speech_event speech_event_driver.py -c CONFIG` (where CONFIG is the path to a configuration ini file; an editable configuration file is provided in the `config/` directory)

References

[1] Cristina Romero-González, Jesus Martínez-Gómez, and Ismael García-Varea. Spoken language understanding for social robotics. In *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 152–157, April 2020.

[2] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented Transformer for Speech Recognition. In *Interspeech 2020*, pages 5036–5040. ISCA, October 2020.

[3] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In *Interspeech 2019*, pages 2613–2617. ISCA, September 2019.

Principal Contributors

The main authors of this deliverable are as follows (in alphabetical order).

Clifford Onyonka, Carnegie Mellon University Africa.
David Vernon, Carnegie Mellon University Africa.
Richard Muhirwa, Carnegie Mellon University Africa.

Document History

Version 1.0
First draft.
Clifford Onyonka.
20 February 2025.