DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Funboy: Deducing, Assessing, Rendering, Validating and Attuning Humor in a Natural Language Dialogue for Human-Robot Interaction

Vagram Airiian

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Funboy: Deducing, Assessing, Rendering, Validating and Attuning Humor in a Natural Language Dialogue for Human-Robot Interaction

Funboy: Ableiten, Bewerten, Erzeugen, Validieren und Abstimmen von Humor in natürlichem Dialog für Mensch-Roboter-Interaktion

Author:	Vagram Airiian
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	Alona Kharchenko & Rafael Hostettler
Submission Date:	15 März 2020

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Garching bei München, 15 März 2020 Vagram Airiian

Acknowledgments

I would like to thank Alona Kharchenko, my good friend and main advisor, for her feedback, advice and all of the great projects we did together at Roboy. I also would like to thank Rafael Hostettler for the opportunity and the necessary resources for this project. Furthemore, I am grateful to Lora Koycheva who gave me support and very useful insight on the conducted experiment.
Moreover, I want to thank the Roboy team for all the friends and experiences I made throughout the years.

Abstract

More and more modern robots can perform various social functions such as maintaining a sustained conversation in natural languages. However, current Human-Robot Interaction experience in Natural Language Dialogues is still far from perfect. We assume that for better interaction, we need to endow our robots with higher-level functionality – such as humour generation capabilities. In this thesis, we decided to explore how people would react to robots that can produce humorous remarks in a sustained conversation. Given the current state-of-the-art in Language Models, we conditioned one of them using jokes collected over online sources and proposed a Natural Language Dialogue framework - DARVAH for Natural Language Generation tasks providing humorous responses in a conversation. We implemented the DARVAH framework in the Funboy module of the Ravestate Dialogue System. The module is capable of generating humorous responses, evaluating emotions of conversation partners and validating the generated humorous output accordingly. We used the implementation to carry out a proof-of-concept experiment that revealed how people react to a joking humanoid robot. The conducted experiment demonstrated great prospects for improving HRI experience via humour-enabled Dialogue Systems. However, further research and experimental evaluation are necessary to collect more data and unlock the full potential of the DARVAH framework.

Contents

1. Introduction

In Robotics, we could recently observe more and more robots being endowed with various behavioural functions not confined to strictly industrial configurations anymore. Now, robots can talk, recognise faces, hug people, and maintain a sustained conversation. We expect them to be personal helpers integrated into our society, meaning robots need to interact with people directly, communicate clearly, and reason thoroughly. However, even if these robots are sociable and likeable, many people remain wary of them. Our society is not yet ready to embrace robotic technologies in everyday life. Thus, researchers need to investigate the communication between robots and people to answer the question: "How can we help people become friends with robots?"
Unfortunately, today, people still consider Social Robots nothing more than toys – something akin to Aibo. This notion leads to at least two problems. Firstly, we do not expect Social Robots to appear self-contained and intelligent. When robots act independently to fulfil their social tasks, interlocutors often act unreasonably being astonished at such interactions due to its novelty. Secondly, we may still need humanoid robots for many human-centred tasks. Humanoid robots pose a whole new challenge to Human-Robot Interaction. They look similar to us, but they are strong and do not tire. By making general-tasks interactions easier for robots, we are making it harder for people to cooperate with these robots in a shared environment - humanoid robots may appear hostile to us.
These two problems usually arise from adopting a pure engineering approach. We have three laws of Robotics and safety rules; therefore, we can make safe robots. We have domain metrics and technological processes; therefore, we can make efficient robots. Nevertheless, this approach is still too mechanical to work well in the intricate world of humans. Our social fabric forms a convoluted and multifaceted system of many sometimes-illogical factors. To rely on someone and to trust them, we have to develop a complex set of psychological conditions based on the interactions we had in the past. A successful social exchange alone is not sufficient to fit into a human-centred environment. All conversation partners have to emulate at least the feeling of belonging in the group. What if we create a robot that emulates its belonging in a social circle? Theoretically, it is possible. We could draw inspiration from the Chinese Room Argument. Originally suggested against simplifications in AI definitions, the argument works well for imitating of intellectual capabilities. The concept of this thought experiment is rather simple. Imagine you are sitting in a room. Somebody slides a sheet of paper with a list of questions in Chinese through a slit on the wall. Unfortunately, you do not understand Chinese. However, in the room, there are manuals in English that explain how to convert the given set of Chinese characters into new characters corresponding to the answer.

You write a correct answer and pass it back through the slit, effectively faking your Chinese skills. Then the idea is simple: let's make robots demonstrate fake awareness of the group dynamics and the conversation. This approach seems plausible. Sometimes, even people imitate understanding of topics they do not know.

Unfortunately, we hit a deeper problem. In theory, the approach works. In practice, robots currently cannot maintain a sufficient context in a sustained conversation. This limitation is again a physical one. Today, we still cannot fit enough computational power into a human-sized robot. Even our mobile assistants cannot boast the ability to understand random questions, having access to data centres and the Internet. Are there other ways to tackle this situation?

To solve the problem, we need to make unorthodox choices. Some roboticists choose to pursue an approach focused on anatomical similarity. Therefore, they try to make a robot which will resemble a human body. They use artificial skin based on elastic materials and soft actuation based on tendons and micro-motors, which help to create more natural movements. Special vocoders help to create soothing voice registers.

Nonetheless, put together, it produces monsters. Small imperfections seem nightmarish to us. We call this phenomenon the uncanny valley. In aesthetics, the valley denotes a relationship between how much an object resembles a human being and our psychological response to it. Objects that match human physiology but are not humans cause us to feel repugnance. The solution is feasible only in the future, as we improve our embodiment technologies and increase acceptance of humanoid robots bringing us back to the square one.

In contrast, we can try another approach by choosing a principle that brings people together universally. For instance, if we think about music, it does feel like a universal phenomenon that works a bit on a subconscious level. Music makes us feel specific emotions associated only with the current auditory information. During verbal interaction, we can argue that humour may serve such a role. Humour allows us to let go and drop our defences.

Moreover, it results in collective laughter which is a way to defuse a social situation. The laughter initiates neurophysiological effects such as the release of endorphins which cause euphoric sensations, and reduction in the stress hormone, cortisol. Usually, in human interpersonal interaction, we find it easier to associate with people we can have fun with and a laugh together. Could it maybe work for robots too? The answer is: "not without a challenge".

Scientists have been trying to crack this enigma. Natural Language Processing incorporates the field of computational humour; in other words, teaching computers to produce humorous outputs in response to user inputs. The field has existed for several decades. However, there have not been many breakthroughs. The lack of success might be due to two reasons. The first one is a performer problem - until very recently, Natural Language Generation (NLG) technologies have not been advanced enough to produce seemingly random yet semantically relevant responses to verbal inputs. However, robots do not need to be strong-AI capable but need to possess a certain "Swiss-knife" conversational

toolkit good enough to just fool their conversational partners for awhile.
The second issue is a recipient problem, which is an issue even for human-generated humour. The reason is that we can never assume beforehand that our spectators or conversation partners will like a specific joke or a comedy style. It is the major complication which stands in the way of computational humour. To solve this problem, robots need to learn to approach people personally. The solution requires compiling interlocutors' profiles with specific styles of humour tailored to each particular person. To evaluate people's reactions to the humorous speech, our robots have to gauge the reaction perhaps via visual means, capturing the visual cues of the conversation partner. Thus, we need to personalise Human-Robot Interaction by creating interlocutor's preference profiles and gauging their preferences individually.
Unfortunately, computational humour is currently not very advanced. Therefore, this project is one of the first steps on the long path to make robots appear more self-contained and intelligent through humour generation capability.

1.1. Problem Statement

The thesis aims to explore how people react to jokes produced by a Neural Language Model conditioned on humour data and delivered by a robot in a verbal interaction - we want to see what is possible with the current level of the necessary technologies. To answer this question, we have to develop a Natural Language Dialogue System module that employs current humour generation and emotion recognition capabilities. Combined together, these two functions can form a feedback loop that not only allows to evaluate the reaction on an emotional level but also to adjust the behaviour of the system and examine how this reaction changes. Using the implementation of the proposed humour generation and evaluation framework, we will conduct the experiment and investigate the obtained results.

1.2. Thesis Structure

This section covers the structure of the thesis.
Chapter 1 introduces the idea of the thesis and the problem it is trying to address. We provide an overview of the theoretical background needed to understand the current state of the art in the field of computational humour and emotion recognition in Chapter 2. It is essential to observe to what extent the previous research in the field brings us closer to the solution. The following chapter provides information about the proposed system design and introduces the Deducing, Assessing, Rendering, Validating and Attuning Humour (DARVAH) framework, which is the cornerstone of the whole research project (Chapter 3). Specifically, the chapter presents key architecture components and their functions. The chapter also proposes how DARVAH should evaluate its interaction with the user.

We describe the technical details of the current system implementation in Chapter 4. There we provide the information about which software modules we built, how the NLG task works, and what tools we used to create Funboy - our implementation of DARVAH. The next chapter (Chapter 5) provides the experiment formulation and its results. There, we explore how people reacted to our robot using the Funboy module to produce humorous responses in the user study. The experiment description provides details regarding its conditions, setup and procedure. Given both the evaluation of the Automatic Emotion Recognition data as well as answers to the self-assessment questionnaires that the participants reported, we discuss the outcome of the user study. In Chapter 6, we discuss the current challenges that stand in the way of the Funboy project. The experiment revealed several limitations that affect the performance of Funboy and influence the users' perception - we describe the challenges that occurred and possible ways to tackle them in the future. In addition, we propose possible further research since there are a couple of ways to improve the functionality of Funboy in the future.
Chapter 7 concludes the current thesis.

2. Background

In this chapter, we explore the current state of the field of computational humour and the needed technologies to implement the system. To be able to conduct the necessary experiment on how people react to joking robots, we require an embodied humanoid robot capable of recording video and audio data. To sustain a conversation, the robot needs a Natural-Language-based Dialogue System (in our case, the language is English). We will need a solution to store the humour-related information about users persistently. This chapter also covers our approach to the emotion recognition task for the experiment.

2.1. What or who is Roboy?

Roboy 3.0 is the robot build by the team of the Roboy project in February 2020. It is the third generation of Roboy's robots. Initially, the project started as collaboration around Prof. Rolf Pfeifer at the University of Zurich in 2013 [1]. Currently, the project resides in Munich and Zurich.
Roboy is an Anthropomimetic Robot. Its musculoskeletal design uses elastic materials to mimic biological muscles and tendons, enhancing dexterity and adaptivity by using tendons routed along its joints and pulled by motors. Roboy 3.0 is in the top of its kind regarding the mechatronic design [2]. However, its social interaction capabilities could be better. Thus, we will use this opportunity to conduct the research to use the results for improving Roboy's verbal Human-Robot Interaction capabilities.

2.2. Dialogue System

To interact with robots, we use Dialogue Systems which provide knowledge extraction and processing capabilities for interaction in natural languages. A conventional Dialogue System can accept an input utterance, gauge its semantics, create a response utterance and provide an output. Most of these systems support interaction via spoken or written means, carried out directly or over social media messages. At Roboy, we developed the Ravestate framework to provide the Dialogue System functionality for our robots.

2.2.1. Ravestate

Ravestate is a framework that provides a real-time modular Dialogue System by incorporating event-based and reactive Software Engineering approaches (see Figure).

Essentially, the systems is a state machine which offers the following abstractions to implement a reactive conversation flow:

- context, defined as a dictionary which keeps track of the spikes in the system.
- properties, defined as context variables that hold necessary values for states to functions.
- signals, defined as variables that can be emitted, which causes a spike in the context.
- states, defined as conditions and an implementing function. The conditions represent disjunctive normal forms of boolean variables and are necessary for states to activate. When a state becomes active, it executes its function. If a state definition contains a signal signature, it can emit the associated signal.

The core functionality of Ravestate is implemented using these concepts in the following modules 2.1:

- ravestate is the Ravestate core module that contains the definitions for the state, the signal and the condition abstractions.
- ravestate_rawio provides raw input/output properties served by the IO modules.
- ravestate_ontology provides the interface to the built-in ontology.
- ravestate_interloc provides the interlocutors' properties, where present interlocutors' data resides.
- ravestate_idle provides bored and impatient signals for dialogue engagement.
- ravestate_verbaliser contains utilities for management of conversational utterances.
- ravestate_nlp provides Spacy-based NLP properties and signals.
- ravestate_emotion handles the robots' facial expressions and animations.
- ravestate_ros1 provides specific properties for ROS 1.0 communication.
- ravestate_ros2 provides specific properties for ROS 1.0 communication.

However, Ravestate lacks any humour related modules or functions [3]. Therefore, we needed to implement such a feature.

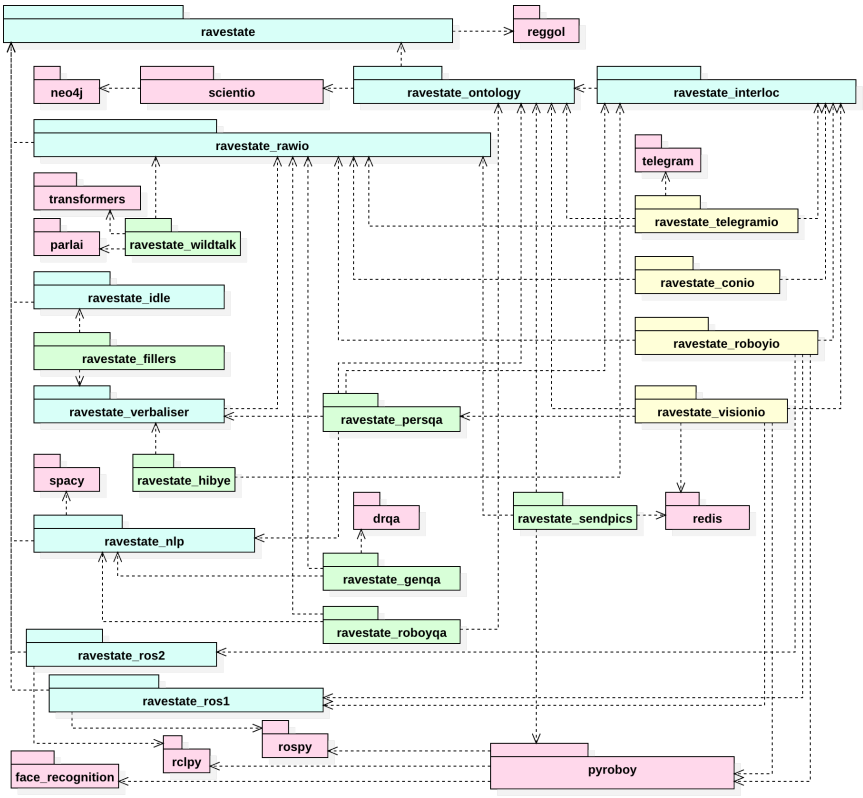

Figure 2.1.: The dependency diagram of Ravestate [3].

2.3. Computational Humour

Humour is essential in interpersonal communication. It influences conversations drastically. Therefore, Computational Humour is considered an essential branch of Natural Language Processing and Computational Intelligence. Some publications report that including humour into embodied robots functionality improves HRI and makes it more pleasant for human users [4].
There has been ongoing research on humour applications in the Natural Language Generation tasks such as generating canned jokes. For example, punning riddles generation:

- What do you call a quirky quantifier? An odd number.

which was carried out as a symbolic description of humorous riddles and resulted in a "JAPE" project in 1994 [5].
Another well-known experiment is the "STANDUP" - a rule-based humour production system for stand-up comedy or a more modern alternative for "JAPE", developed in 2007. It used WordNet and was capable of generating such samples as:

- What do you call a cry that has pixels? A computer scream.

This experiment was the first attempt to create a joke generation system widely available to the end users [6].
However, even though implementing humour processing pipelines in Dialogue Systems is advantageous not only for joke generation on the go but also for other NLG problems, there was no significant breakthrough so far. Firstly, from the Natural Language Processing perspective, the intricate semantics of humour demands a lot of creativity and deep understanding of the context which may require the human mind to produce a joke. Secondly, humour is a multimodal interaction including but not limited to joke delivery, gestures, voice tonality, facial expressions.
Adapting to the current context might be a critical task and research problem for Social Robots. The NLG capabilities are one of the essential functions to interact with people and other Social Robots, which means that the robot has to produce response utterances online without preparing them beforehand. Besides language processing at runtime, the robot has to evaluate the non-verbal behaviour of its conversation partners to pick up subtle cues about the current context. All of the functionality mentioned above is also necessary for humour generation tasks.
Therefore, there are two important considerations to address:

- Lack of flexibility – we would like to be able to generate a wide range of different jokes using unsupervised approaches to create a Natural Language Generation system.
- Lack of feedback – we may never be sure if a generated joke is funny to a particular interlocutor. Thus, we need some approach to establish feedback for each conversation partner.

Regarding the feedback problem, there is a proposed approach from researchers at the University of Augsburg. Their idea is to combine the NLG and Reinforcement Learning (RL) so that a robot can adapt to the particular user's behaviour using external signals. To develop such capability, they propose to use multimodal Human-Robot Interaction in the context of dynamic joke generation as a way of learning how to improve the robot's perceived social intelligence. The modalities of interest are speech and video data to extract emotion-related patterns such as smiling or laughing. These modalities estimations are combined with the RL mechanism to generate rewards when the estimates are positive and punish the behaviour otherwise. This approach allows to automatically adjust the NLG parameter according to modality signals from the user [7] [8]. Figure 2.2 illustrates the concept of the proposed approach.

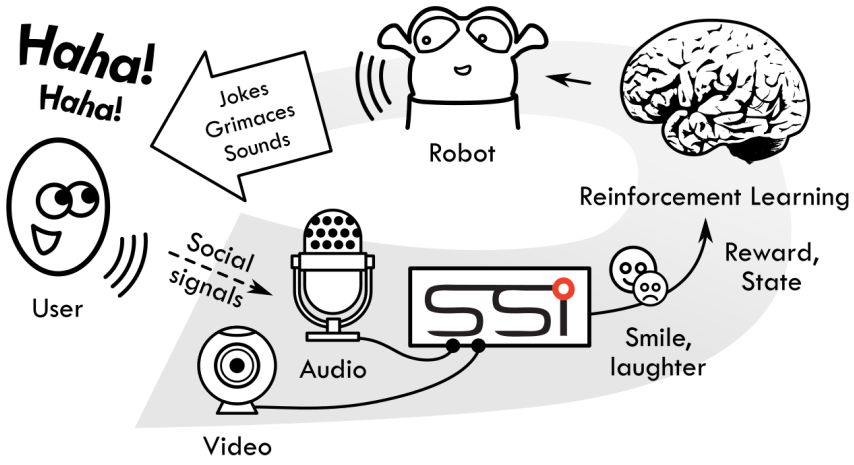

Figure 2.2.: Scenario of a robot learning to be funny from human social signals [8].

Following this approach, we would also like to take image and audio data into account in our system, considering that Roboy has all of the necessary hardware for this. However, we still need a Language Model for the NLG task.

2.4. Language Model

To successfully conduct the NLG tasks, we require a system that knows a probability distribution over words. Such systems are called Language Models. Natural Language Processing systems use these models to determine the necessary context to differentiate between words. Thus, given the particular word, we can predict the most probable words that could follow the initial one in the generated sentence. This capability is indispensable for humour generation tasks. Since recently, state-of-the-art Language Models have utilised Neural Networks for better performance. One of the latest such models is GPT-2 by OpenAI.

2.4.1. GPT-2

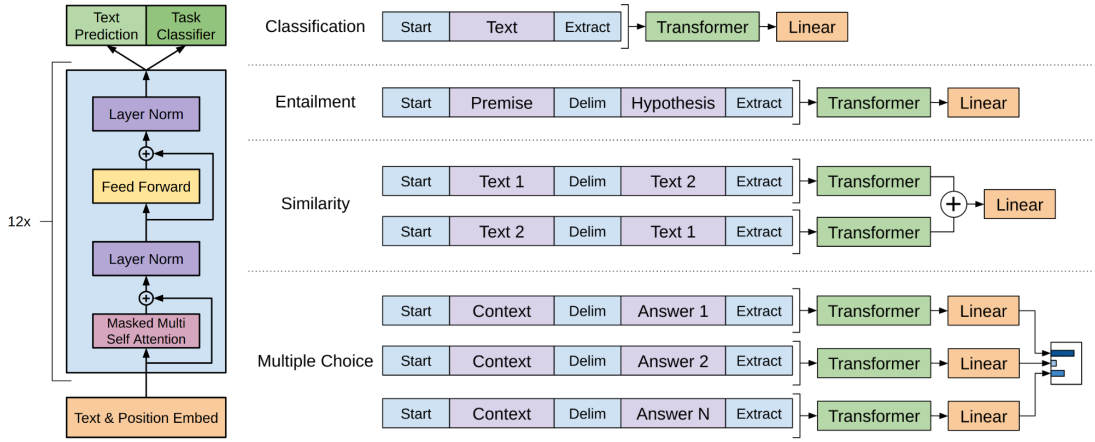

Figure 2.3.: GPT-2 Transformer architecture and training objectives [9].

OpenAI developed a new GPT-2 Language Model in 2019 based on the previous GPT model, with over 10-fold increase in parameters and used data. It employs a Deep Neural Network architecture to predict the next word (token) given the previous ones in the input text. The cornerstone of the architecture is the Transformer-layer (see Figure 2.3). Currently, the biggest available model contains 1.5-billion parameters. GPT-2 can learn Natural Languages tasks such as question answering, summarisation and comprehension within an unsupervised training approach while being superior to other Language Models (on 7 out of 8 tested datasets).
The GPT-2 prediction quality directly derives from the amount of training data. Therefore, the training process employed 40 GB of publicly available text from 8 million online sources which allow generating naturally sounding sentences due to the diversity of the dataset and the data domains. However, the creators reported that might take several prediction attempts until the model outputs a satisfactory sample given how broadly the topic was covered in the original data with up to 50% success rate. For our NLG task, humorous data naturally belongs to numerous domains. However, there are usually many samples that use similar lexical and vocabulary structure. A good example is the "chicken" riddle. Usually, it starts with: "Why did the chicken cross the road?" Then, follows the punchline – anything silly or absurd: from obvious "To get to the other side" to "She was afraid someone would Caesar!" Structurally, this type of jokes is very rigid. Therefore, we would like to see how well GPT-2 will perform on our data [10].

2.5. Automatic Emotion Recognition

The visual and audial cues can be crucial for humorous HRI applications similarly to how human psychology shapes the perception of verbal interaction. Given visual or

auditory stimuli, humans often gauge the seriousness of the situation based on facial expressions as well as timbre[1] and cadence[2] in social interaction. Therefore, we need a feedback mechanism to evaluate users' emotions.
To help Roboy to recognise and evaluate emotions of conversation partners, we have decided to adopt two following technologies to use in our implementation. The first one is the EmoPy package that offers facial emotion recognition using a frame-by-frame video stream. Another one is Speech Emotion Recognition (SER) - it provides emotion recognition result based on audio recordings. SER is a part of the bigger Multimodal Emotion Recognition project [11].

2.5.1. EmoPy

Developed at ThoughtWorks Arts, EmoPy is a Facial Expression Recognition (FER) package. The package offers various Neural Network architectures trained on facial expression data. The available pre-trained models use the Microsoft's FER+ dataset because it is the best currently available open-source dataset for FER [12].
FER+ is a dataset based on the previous Microsoft FER data. With the help of ten crowd-sourced taggers, creators of the dataset re-labelled all data points to improve the ground truth quality on each image. Similarly to original FER, FER+ offers still-image emotion data and labelling [13].
Using FER+ labels, EmoPy can recognise seven available emotions such as calm, anger, disgust, fear, happiness, sadness, and surprise. The best available EmoPy model uses a Convolutional Neural Networks (CNN) architecture (see Figure 2.4). The software implements the Neural Network modules via the Keras framework employing Tensorflow as the backend [14].
EmoPy is an open-source project available through PyPi package management system and as source code on GitHub [14].

2.5.2. Speech Emotion Recognition

The Speech Emotion Recognition is a technology that analyses emotions expressed in speech patterns given a wave recording of the person speaking. We used the SER package from the Multimodal Emotion Recognition project. The SER package employs Machine Learning models trained on the Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) dataset.
The dataset comprises 7356 data points. To create each sample, 24 (12 female, 12 male) professional voice actors participated in recording spoken utterances in a neutral North American accent. Each file consists of two matching utterances. The samples were recorded at two levels of emotional intensity, either normal or strong. The emotion labels comprise calm, happy, sad, angry, fearful, surprise, and disgust. To assign labels

[1] the character or quality of a musical sound or voice as distinct from its pitch and intensity
[2] a modulation or inflexion of the voice

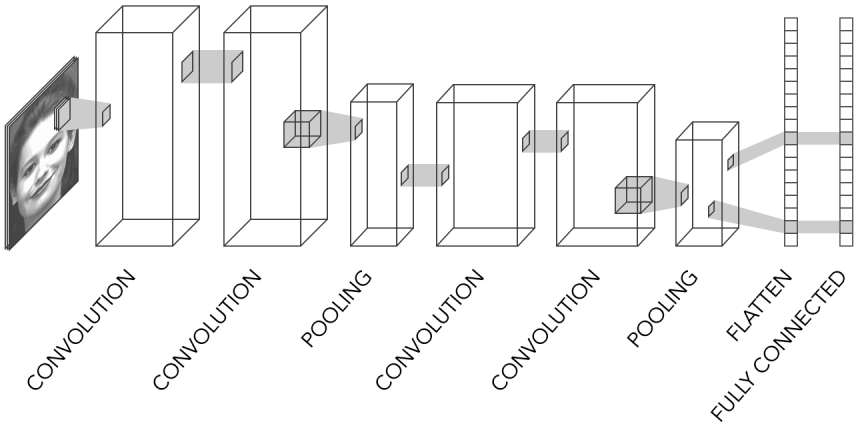

Figure 2.4.: EmoPy CNN architecture [14].

to each of the samples, 247 untrained researchers from North America provided ten ratings for each file with regard to its validity, intensity, and authenticity [15].

The Multimodal Emotion Recognition project is open-source software distributed under Apache License 2.0 and developed for the French employment agency Pole Emploi. The goal is to create a platform for job-seekers which will help the candidates by analysing their verbal and non-verbal behaviour. The SER offers Support Vector Machine and Time Distributed Convolutional Neural Network models. Given the higher accuracy provided by the CNN model, we will use it in our implementation. Figure 2.5 illustrates the architecture of the model.

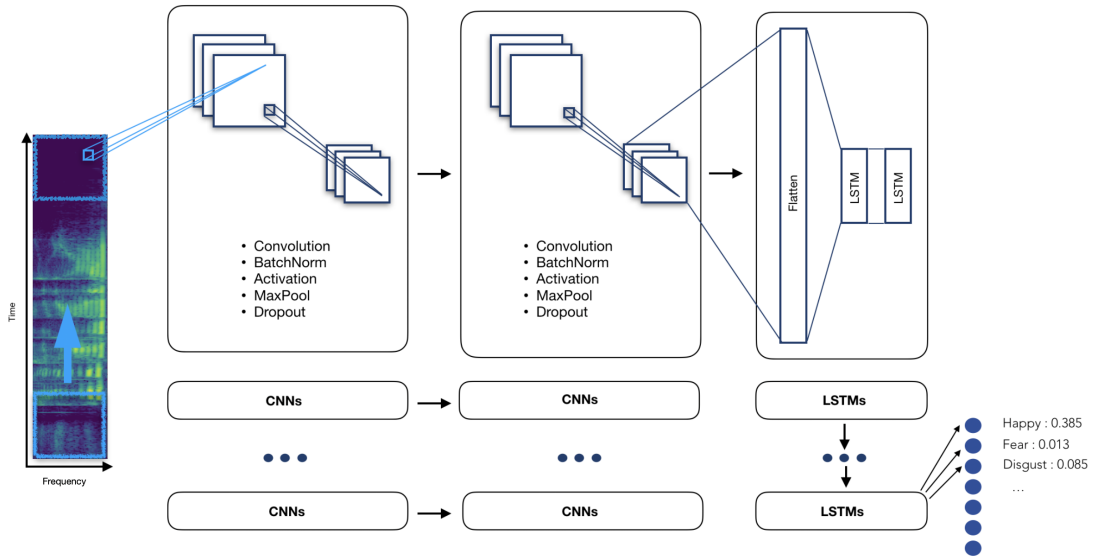

Figure 2.5.: SER Time Distributed CNN architecture. [github]

The idea behind the architecture it to employ a rolling window technique along the mel-spectrogram. Afterwards, each of the resulting window output constitutes an input to the Convolutional Neural Network layers. These layers consist of four Local Feature Learning Blocks (LFLBs). Then, the values are fed into the next neural network comprising 2-cells LSTM (Long Short Term Memory) model. At the end of the model pipeline, there is a softmax-activated Fully Connected (FC) layer which outputs the emotion labels for the speech samples [11].

3. System Design - DARVAH Proposal

In this chapter, we propose a dialogue framework for NLP applications implementing conversation flows, providing humorous responses via dialogue interfaces as specified in Chapter 4. The interfaces enable a dialogue system to communicate with the environment via software and hardware. The dialogue system comprises a feedback loop orchestrated as a set of states which allow Natural Language Generation and information extraction. Further, we elaborate on the proposed architecture.

3.1. DARVAH Framework

DARVAH system is a loop comprising five stages:

1. D stands for "deduce" and is a state that decides whether to generate either humorous or conventional response.
2. A_s stands for "assess" and is a state that selects an appropriate humour type according to available user data.
3. R stands for "render" and is a state that produces a humorous text for reply given the selected type.
4. V stands for "validate" and is a state that validates the generated response using emotion recognition feedback given positive or negative emotions experienced by the interlocutor.
5. A_t stands for "attune" and is a state that adjusts the humour types scores of the given interlocutor.

In this section, we clarify the roles and functioning of each module inside the dialogue system. Figure 3.1 shows a general structure of the proposed system.

3.1.1. Deducing State D

To conduct a successful verbal exchange in a natural language, the Dialogue System requires capabilities to recognise whether the actual input is intelligible and entails an anticipated answer. The system makes a choice either to proceed with the ongoing conversation or to start a new one. In our case, this stage becomes the precursor to the humour generation process. Therefore, besides evaluating whether the input is relevant, the module has to answer a particular question: “Does the current state of the

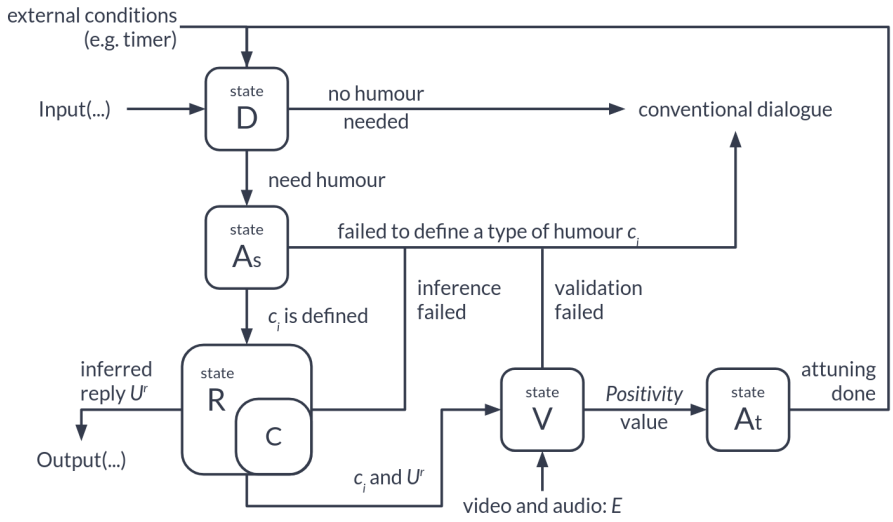

Figure 3.1.: DARVAH system with the five main stages

conversation prompt to pursue a humorous follow-up rather than a humourless one?" – i.e. to deduce if the conversation flow provides any immediate potential for a comical response (we denote this module by D which stands for "deducing/deciding"). It is important to note that we do not confine the module's to exclusively natural language utterances as it is beneficial to incorporate other variables, e.g. computer vision to capture the emotion cues expressed by a conversational partner or other internal signals of the Dialogue System.

To illustrate how the module makes a decision, we will consider the following situation. Let's assume two possible examples. Firstly, we could derive the condition to trigger the humour generation process via visual or audial cues prediction, such as negative emotion recognition result. This approach constitutes an active method of entering the DARVAH flow. In contrast, a passive approach presumes that users will be asking for a humorous response directly, for example:

U^i_{t-1}: Please, tell me a joke

U^r_t: Why are robots shy? Because they have hardware and software but no underwear.

where U^i_{t-1} is the utterance at the dialogue time step $t-1$ which is the verbal input we received from the interlocutor I_i. Considering the system produces a response at the current dialogue time step t, U^R_t is the utterance produced by the robot at the current step.

Another possible valid approach is to randomly proceed with the humour generation process, to introduce an element of spontaneity to the Natural Language conversation, which often occurs in verbal interaction between humans.

In case module D does not decide to proceed with a humorous follow-up, it makes the dialogue system transition to the conventional dialogue states until the next time the decision process triggers.

3.1.2. Assessing State A_s

If module D decides to proceed with a humorous follow-up, the system will activate the next module – A_s. This module assesses which type of humour to utilise based on available user data. The Dialogue System data structures contain information about the current and previous interlocutors and associations between them and the joke types (see Ch. x). To choose a joke type to generate a humorous response subsequently, A_s introduces and uses the information about the likeability of each joke type defined as affinity scores. This process is not trivial since it has to take into account further joke types score adjustments happening in the module A_t. To clarify the process, let's consider the following example of the possible affinity values provided in Table 3.1.

Affinity Scores							
...	c_1	c_2	...	c_j	...	c_{n-1}	c_n
I_{i-1}	5	5	...	8	...	5	0
I_i	5	10	...	15	...	20	5
I_{i+1}	10	20	...	5	...	0	0

Table 3.1.: Example affinity scores before the assessment

Where c_j is a joke category (or type), $j \in [1, n]$ and I_i is a current interlocutor i. The score scale is an integer scale with a step of 1 with each score $s_i j \in [0, 100]$. $S_i = \{s_1, s_2, ..., s_n\}$ is then a vector of affinity scores that belongs to the interlocutor I_i.
At the current step in module A_s, we require a heuristic to choose the next type of joke. Obviously, by choosing one with the maximum score, we are reinforcing only a single type of jokes until its complete saturation. If it is the case, we cannot select any other type, and the interlocutor will not be able to judge whether he or she prefers any other joke more. Therefore, the situation requires a more creative approach to resolve the saturation and flexible choice issues. To address the challenge, we proposed the following simple choosing strategy:

$$c_j = Assess(S_i)$$

where

$$Assess(S_i) = \begin{cases} choice(C_j^+), & \text{if } Validate(E_{t-2}^i[m]) \geq 1 \\ choice(C_j^-), & \text{if } Validate(E_{t-2}^i[m]) < 0 \end{cases}$$

where $Validate(E_{t-2}^i[m])$ is the validation result (defined further in Section 3.1.5) from the previous run of the DARVAH loop. Vector C_j^+ contains all types for which values from S_i are bigger or equal to $mean(S_i)$ and vector C_j^- contains all values all types for which values from S_i are less than $mean(S_i)$. $choice(C_j)$ is a random choice function that generates a sample from given array of number assuming any kind of a probability distribution over the values, for example, a uniform distribution.

Let's consider the current scores $S_i = \{5, 10, ..., 15, ..., 20, 5\}$ for interlocutor I_i. Assuming that

$$mean(S_i) = \frac{1}{N} \sum_{n=1}^{N} S_i n = 11$$

and $Validate(E^i_{t-2}[m]) = 1$,

$$Assess(S_i) = choice(c_j, c_{n-1}, ...)$$

If for some reason A_s fails to choose any type, it hands-over control to the conventional dialogue.

3.1.3. Rendering State R

After module A_s selected the joke type, module R assumes control and proceeds with the Natural Language Generation (NLG) task. This module is the principal stage of the humour-generation process and provides an interface to a Natural Language Generation sub-module – the Comedian. R can choose between proactive and reactive response rendering. We distinguish between these two kinds of humorous language generation due to the following reasoning.
The proactive response usually comprises a storytelling bit, when a group of people listen to the entertainer until he finishes. The performer narrates the bits of monologue, which are similar in rhythm, to build up the story. And at the end of the monologue, there is usually a culmination that takes an unexpected turn. In contrast, reactive response assumes active engagement in a conversation from both sides. The case is that when one conversation partner makes a remark, the other makes another one in response. Thus, this type of humour demonstrates more improvised features. For instance, stand-up is primarily a proactive type of comedy, while puns are often reactive. Therefore, to be able to choose either proactive or reactive response rendering, we propose a decision-making function $F_d(c_j, U^i_{t-1}) \in \{True, False\}$. A simple example of such a function could be:

$$F_d(c_j, U^i_{t-1}) = \begin{cases} True, & \text{if } \exists c_j \in \{c_1, ..., c_n\} : Similarity(c_j, Triple(U^i_{t-1})) \geq 1 \\ False, & \text{otherwise} \end{cases}$$

where $c_j = Assess(S_i)$ is one of the humour categories,

$$Triple(U) = \{Subject(U), Predicate(U), Object(U)\}$$

is a function which produces triple from sentence U and

$$Similarity(c, Triple(U)) = \sum_{k=1}^{3} sim(c, Triple(U)_k) \in [0, 3]$$

is a function that calculates a sum of semantic similarity $sim(x, y)$ between c and $Subject(U)$, $Predicate(U)$ and $Object(U)$.

A positive or negative decision will affect the rendering process in the following way. To obtain a response, we will pass only the selected type of humour if the decision was to use proactive humour ($F_d(c_j, U^i_{t-1}) = False$). Otherwise, we will also pass the interlocutor's input utterance U^i_{t-1}:

$$Render(c_j, U^i_{t-1}) = \begin{cases} Comedian(c_j, U^i_{t-1}), & \text{if } F_d(c_j, U^i_{t-1}) = True \\ Comedian(c_j), & \text{otherwise} \end{cases}$$

here, $Comedian : c_j, U^i \mapsto U^r$ is a variadic function that represents the NLG sub-module. The sub-module is a black-box abstraction over a humour-conditioned Language Model which creates humorous sentences and can represent any combination of rule-based production systems with DNN-based systems. After module R obtained the utterance U^r_t, the system outputs the response by the means of textual and/or speech interfaces.

3.1.4. Validating State V

To validate the emotional response, we propose module V which recognises emotion patterns expressed by the interlocutor, for example, visual and audial cues. The module receives a set of emotions scores vectors E^i_t for interlocutor I_i. These emotions can be separated into positive and negative ones. Therefore, we compare a sum of positive emotions scores against a sum of negative emotions scores for each vector E^i_t according to the following function:

$$Positivity(E^i_t) = \begin{cases} 1, & \text{if } \sum_{n=1}^{N} E^{i+}_{t,n} > \sum_{n=1}^{N} E^{i-}_{t,n} \\ 0, & \text{if } \sum_{n=1}^{N} E^{i+}_{t,n} = \sum_{n=1}^{N} E^{i-}_{t,n} \\ -1, & \text{if } \sum_{n=1}^{N} E^{i+}_{t,n} < \sum_{n=1}^{N} E^{i-}_{t,n} \end{cases}$$

where N is the number of recognised emotions.
Then, for E^i_t - an array of m vectors E_t:

$$Validate(E^i_t) = mode\{Positivity(E^i_{t,1}), Positivity(E^i_{t,2}), ..., Positivity(E^i_{t,m})\}$$

When the validation module obtains the result, it passes the value to module A_t.

3.1.5. Attuning State A_t

When the system receives the validation result in module A_t, it will add 1 point to the score if the interlocutor preferred the evaluated types of humour. For the types that interlocutor did not like, the module will subtract 1 point from the score. For example, if, for the selected type c_j, $Validate(E^i[m]$ equals to 1, A_t needs to increase the affinity score s_{ij} by 1 as per:

$$Attune(S_i|c_j) = \begin{cases} s_{ij} = s_{ij} + 1, & \text{if } Validate(E^i_t) = 1 \\ s_{ij} = s_{ij}, & \text{if } Validate(E^i_t) = 0 \\ s_{ij} = max(s_{ij} - 1, 0), & \text{if } Validate(E^i_t) = -1 \end{cases}$$

Besides the current score adjustments, A_t performs value decay over time according to a decay multiplier parameter applied to each value based on the timestamp of the previous update to introduce the functionality to slowly forget humour preferences. This capability is convenient if we expect our interlocutors to interact with the robot again after a long period of time because the decay function will first discard the scores with lower values allowing the most favourite joke categories to persist longer.
Given that the decay strategy can be arbitrary, we propose to define the parameter as a percentage of each score the system forgets over each preceding interval of time. Then, we calculate the number of intervals since the previous update of the scores and raise the decay multiplier to the power of the amount of the interval. Afterwards, updated affinity scores equal to each of them multiplied by the resulting value and rounded:

$$S_i^{'} = round((1-p)^{\frac{t_now - t_prev}{T}} S_i)$$

where p is the decay parameter, T is the period in seconds, t_now is the current time, and t_prev is the time of the last update. Rounding function is the round half away from zero function as following:

$$round(x) = sgn(x)\lfloor |x| + 0.5 \rfloor = -sgn(x)\lceil |x| - 0.5 \rceil$$

For example, let's set $p = 0.1$ which corresponds to 10% decay and $T = 86400s$ which equals to 1 full day. Given that the last update for interlocutor I_{i-1} happened two days ago, the decayed scores will be:

$$S_i^{'} = round(((1-0.1)^{\frac{2*86400}{86400}}(5,5,...,5,...,5,0)) = round(0.81(5,5,...,10,...,5,0)) \\ = (4,4,...,8,...,4,0)$$

Given values in Table 3.1, we provide the final scores adjustment result in Table 3.2.

Affinity Scores							
...	c_1	c_2	...	c_j	...	c_{n-1}	c_n
I_{i-1}	4	4	...	8	...	4	0
I_i	5	10	...	16	...	20	5
I_{i+1}	10	20	...	5	...	0	0

Table 3.2.: Example affinity scores after the adjustment process

After A_t finishes the score attuning process, state D becomes active again.

3.2. The Comedian

In this section, we would like to clarify the difference between proactive and reactive humorous Natural Language Generation tasks. Since we are employing a Language

Model based on GPT-2 Deep Neural Network model, the following example illustrates the usage of this particular.
Because GPT-2 can accept from one to technically unlimited amount of tokens as an initial prompt to generate further tokens, it inherently represents a variadic function. This feature allows us to use the model as the Comedian submodule without any unique intermediate interfaces or combinations of separate Language Models for either proactive or reactive response generation tasks.
Let's consider the following toy problem. Given the model $\mathcal{M}$ implementing the GPT-2 architecture and the dataset $\mathcal{D}$ containing:

- a single class: $c_j =< |robot| >$
- a single sentence: $U =$"Why are robots shy? Because they have hardware and software but no underwear."

which form a single data point:

- <|robot|>Why are robots shy? Because they have hardware and software but no underwear.<|endoftext|>

we had trained $\mathcal{M}$ on the dataset until it overfitted. Now, we would like to test how it performs in the NLG task.
Case 1: Proactive response generation.
If we have set the input utterance to an empty value $U^i_{t-1} =$"" and run the response generation, we will obtain the following result:

1. chosen class: $c_j =< |robot| >$
2. input utterance: $U^i_{t-1} =$""
3. decision-making function: $F_d(c_j, U^i_{t-1}) = False$
4. rendering function: $Render(c_j, U^i_{t-1}) = Comedian(c_j) = \mathcal{M}(c_j)$
5. response utterance: $U^r_t = \mathcal{M}(< |robot| >) =$"Why are robots shy? Because they have hardware and software but no underwear."

Case 2: Reactive response generation.
If we have set the input utterance to a specific value $U^i_{t-1} =$"Why are robots shy?" and run the response generation, we would obtain the following result:

1. chosen class: $c_j =< |robot| >$
2. input utterance: $U^i_{t-1} =$"Why are robots shy?"
3. decision-making function: $F_d(c_j, U^i_{t-1}) = True$
4. rendering function: $Render(c_j, U^i_{t-1}) = Comedian(c_j, U^i_{t-1}) = \mathcal{M}(c_j, U^i_{t-1})$

5. response utterance: $U_t^r = \mathcal{M}(< |robot| >$, "Why are robots shy?") ="Because they have hardware and software but no underwear."

The main difference is that the model will incorporate the reactive input utterance (which can also be pre-processed into more sophisticated tokens before use) as a stronger prior for the generated tokens. This approach could be useful, for example, for a question-answering task such as riddle jokes. This style of humour represents riddles that ask a question as a premise for a humorous punch line for an answer.

3.3. DARVAH Ontology

In this section, we would like to mention our proposal to implement an ontology for the DARVAH framework.
Again, let's consider our example from Section 3.1. If we define a graph

$$G = (\{I, C\}, \{\{I_1, c_1\}, \{I_1, c_2\}, ..., \{I_i, c_j\}, ...\})$$

where $\{I, C\}$ is a set of vertices, I is a set of all interlocutors and C is a set of all humour categories such that they form the nodes of the graph G , then we can assign a connection from each interlocutor to each type of humour $\{I_i, c_j\}$. The scores s_{ij} will define the weight of the connection. Therefore, for interlocutor I_i, S_i will denote all weights of all connections to all humour categories in the graph (see Figure 3.2). Using this approach, the $Assess(S_i)$ function simply returns the humour node c_j with the corresponding connection weight r_{ij} (as per affinity scores in Table 3.1):

$$s_{ij} = r_{ij}(I_i \rightarrow c_j) = 15$$

while the $Attune(S_i|c_j)$ function changes the weight of the connection (as per affinity scores in Table 3.2):

$$r_{ij}(I_i \rightarrow c_j) = s_{ij} = 16$$

This architecture is very convenient to use in a graph-based database and easily supports adding new types and interlocutors. Since it is possible support Neo4j database, thanks to this design decision, we can integrate the DARVAH ontology directly into Ravestate ontology.

3.4. Emotion Recognition Subsystem

In this section, we discuss an approach to collecting and using emotion recognition data. Since it is unclear at which moment of system time we need to validate the emotion recognition data, we propose to continuously evaluate the input sources such as camera and microphone streams and access the recognition results as soon as we need them for the validation step. Therefore, we imagine the task of emotion recognition as a series of discrete steps where FER and SER run as two parallel processes (see Figure 3.3).

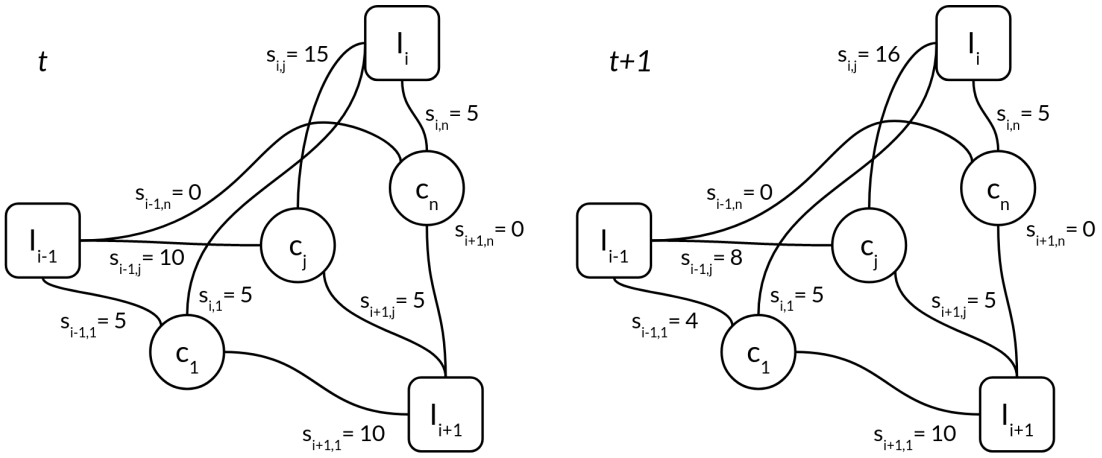

Figure 3.2.: The ontology graph for DARVAH before and after attuning the scores.

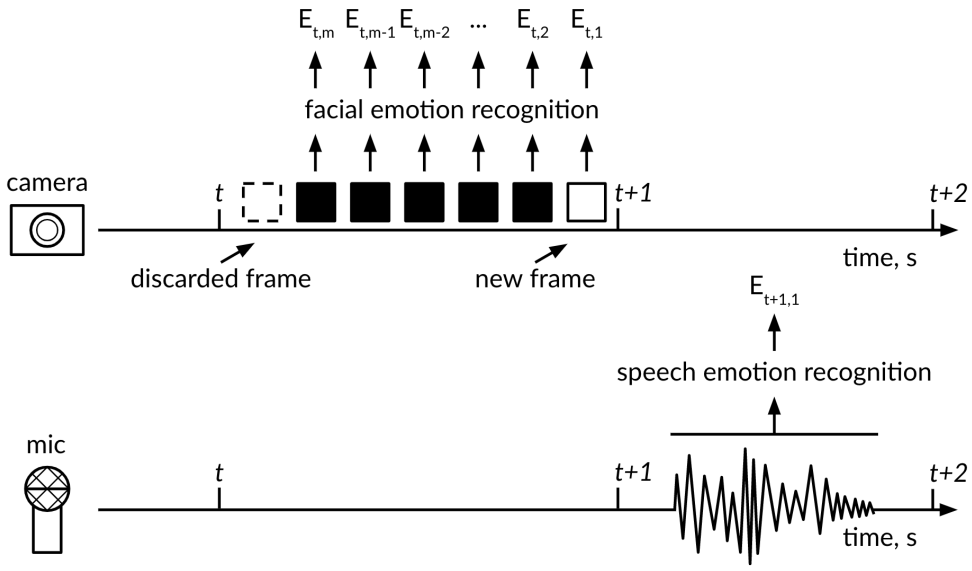

Figure 3.3.: The emotion recognition timeline for DARVAH.

3.4.1. Facial Emotion Recognition

For the FER task, we can enable our Facial Emotion Recognition to capture web camera video stream frame-wise and classify facial emotions based on each frame. Thus, the DARVAH framework can access the latest m results from the recognition subsystem at any moment of system time. At the time step t, when state R has produced the response utterance U_t^r, the validation state will receive:

$$E_t^i = \{E_{t,1}^i, E_{t,2}^i, ..., E_{t,m-1}^i, E_{t,m}^i\}$$

where $m = f \times w$, f is the frequency of sampling and w is the window size. For the standard sampling $f = 30Hz$ (frames per second in the stream) rate and a window size $w = 10s$ (which is a sufficient size for most of the response utterances U_t^r to be perceived by the interlocutor I_i), E_t^i will contain m = 30 Hz * 10 s = 300 emotion score vectors.
As soon as the FER module receives a new sample, it discards the oldest one according to first-in, first-out (FIFO) queue principle, such that the last added element is always the most recent recognition result. On the DARVAH side, we can access all of the queue elements as an ordered list.

3.4.2. Speech Emotion Recognition

To successfully perform the SER task, we need to record the following spoken utterance U_{t+1}^i by the interlocutor. As soon as the robot is done vocalising the response U_t^r, we can start the recording and finalise it as soon as the interlocutor stopped speaking. After the SER module obtains the recording, it recognises the emotions in speech and outputs a single vector of the emotion scores which we denote by E_{t+1}^i given that the following utterance by the interlocutor I_i occurs in the next time step after the response U_t^r. However, this circumstance means that state V can only utilise the SER result when DARVAH produces the next response at time step $t + 2$ because the previous execution of the DARVAH loop has already completed. To resolve this discrepancy, we propose the following emotion combination rule:

$$E_{t+2}^{i'} = \sum_{k=1}^{m} \alpha E_{t+2,k}^i + \beta E_{t+1}^i$$

where $\alpha \in [0, 1]$ and $\beta \in [0, 1]$ are the weights for vectors $E_{t+2,k}^i$ and E_{t+1}^i respectively, such that $\alpha + \beta = 1$. At the next DARVAH time step $t + 2$, this rule allows the previous Speech Emotion Recognition result from interlocutor's input U_{t+1}^i to condition the next emotion evaluation $Validate(E_{t+2}^i)$.

4. Funboy, the DARVAH Implementation

In this chapter, we will provide the implementation details. Funboy is the module that offers the DARVAH loop functionality inside of the Ravestate Dialogue System. This approach allowed us to seamlessly integrate the current dialogue functionality with the new capabilities which we developed to conduct the experiment (see Chapter 5). Figure 4.1 illustrates the full system overview.
There are several components in the system:

- Funboy is a Ravestate module that implements the main loop of the DARVAH framework.
- GPT-2 TLH is a GPT-2-based DNN Language Model conditioned on humour data via Transfer Learning.
- Funboyn4j is a Docker container that provides persistence for users' humour profiles via Neo4j graph database instance running inside of the container.
- Rosemo is an emotion recognition software that contains EmoPy Facial Emotion Recognition and Speech Emotion Recognition (SER) software modules to support processing and evaluating emotions. We deploy Rosemo in a Docker container as well.
- Roboy Sonosco for speech synthesis and roboy_speech_recognition for speech recognition capabilities. Both packages use Google Speech Services to provide functionality.
- The interfaces between the parts use ROS 1.0 and web-sockets for communication.

The system employs the following important dependencies used in development and deployment and is guaranteed to work with the stated versions of the software distributions:

- Python 3.6 is the target Python version[1] for Ravestate
- Docker 19.03 to deploy the containers
- Rosemo with EmoPy (version from 23.01.2020 on GitHub) and SER (version from 28.06.2019 on GitHub)

[1] by default we assume this version anywhere we mention Python

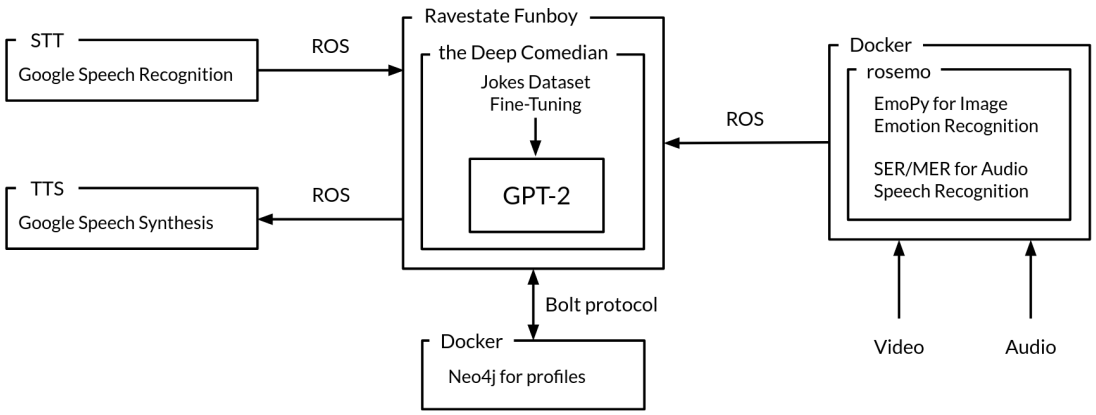

Figure 4.1.: The system implementing Funboy - DARVAH for Roboy.

- ROS 1.0 Kinetic or Melodic for ROS-based interfaces on Ubuntu 16.04 or 18.04 respectively
- roboy_communication package containing all ROS messages and nodes for Roboy
- Tensorflow 1.15.2 for EmoPy and SER to run the models on CPU
- Tensorflow-gpu 1.15.2 to run GPT-2 TLH on GPU
- CUDA 10.0 and CuDNN 7.6.4 for GPU computing capabilities
- ScientIO 0.9.0 to interface from Ravestate to Neo4j
- neo4j-driver 1.7.6 to provide a Neo4j database driver for ScientIO
- Neo4j 3.5 for compatibility with neo4j-driver
- PyAudio 0.2.11 for audio processing capabilities

It is important to note that even if the full implementation does not need an embodied robot to work, it is not fully independent from Roboy as it heavily uses software distributed by the Roboy project. However, all Roboy-developed software adheres to open-source principles and follows permissive licensing practices.

4.1. Ravestate Funboy Module

Funboy is a proof-of-concept implementation of the DARVAH framework which we developed as a Ravestate module. Ravestate is not only the default Dialogue System for Roboy it also offers the system of abstractions very convenient for implementing the DARVAH loop in code (see Section 2.2 in Chapter 2).

We define all of the main Funboy module structures in the init file. It comprises six Ravestate states. The first state *start*() initialises the following Funboy Ravestate properties:

- prop_start_time = rs.Property(name="start_time"), the property defines a timestamp used to postpone start-up of the DARVAH deducing state *D*.
- prop_comedian = rs.Property(name="comedian"), the property defines the ComedianStrategy object.
- prop_emotion = rs.Property(name="emotion"), the property defines the EmotionStrategy object.

The state activates on the condition rs.sig_startup which occurs when the Ravestate starts up.
The following state *deduce*() implements DARVAH's *D* functionality. It activates every time the Ravestate input triples change in the ravestate_nlp module: nlp.prop_triples.changed(). This state reads prop_start_time property and checks if it satisfies the δ time constraint - if the difference between the current system time and the property is bigger than the constraint, it starts the DARVAH loop. Otherwise, other Ravestate dialogue states take precedence. If the DARVAH deducing process is successful, the state emits sig_tell_joke signal.
The third state is *assess*(). Its activation condition is those mentioned above sig_tell_joke Ravestate signal. The state reads the interloc.prop_all property which contains all the interlocutor's data in Ravestate. This property is necessary to access the affinity scores for joke types. When *assess*() selects the joke category, it writes the chosen category into prop_joke_category property.
The fourth *render*() state activates given two conditions: the sig_tell_joke and the prop_joke_category.changed() signals. Thus, after *deduce*() emits the signal, this state still waits for the *assess*() state to write into the prop_joke_category property. When both signals are present, the state accesses the ComedianStrategy object from the prop_comedian property (see Subsection 4.1.1). If necessary, it configures the strategy and calls the *render*() method. Based on the current utterance and chosen joke category, the state either skips the utterance to perform proactive humour generation or passes the utterance to perform reactive humour generation. Afterwards, the state emits the sig_joke_told signal.
The next state is *validate*() which activates given the sig_joke_told signal. This state reads the EmotionStrategy object prop_emotion property (see Subsection 4.3). If necessary, it configures the strategy and calls its *get_positivity*() method. Then, *validate*() evaluates the received values and writes the evaluation result into prop_emotion_val.
The last one is the *attune*() state. It activates when two conditions are present: the sig_tell_joke and the prop_emotion_val.changed() signals. The state reads the prop_joke_category and the interloc.prop_all properties to access the affinity scores of the current interlocutor. Based on the validation result, it changes the score of the joke

type accordingly and writes back into interloc.prop_all. When the writing process is done, *attune*() emits sig_joke_finish to restart the Funboy's DARVAH loop.

4.1.1. ComedianStrategy

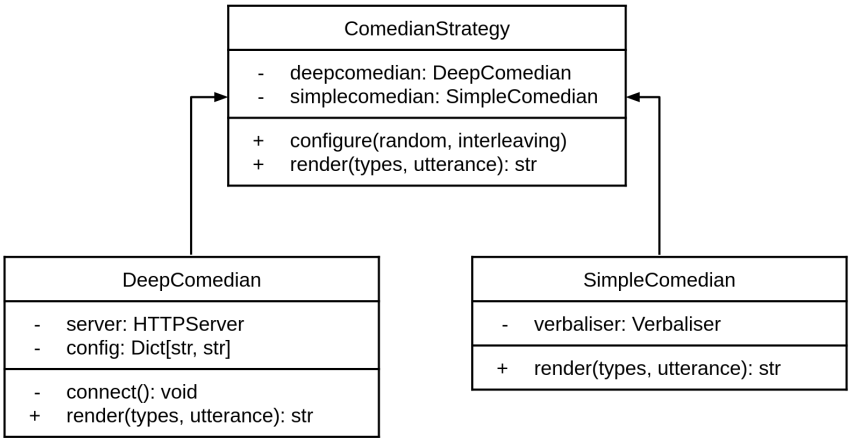

Figure 4.2.: The UML diagram for ComedianStartegy.

ComedianStrategy is a class that implements the strategy pattern for the joke generation process (see the listing for this class in Appendix A). When initialised, it creates both DeepComedian and SimpleComedian objects (see Figure 4.2 and Section 4.2). The constructor accepts one parameter, which is the threshold value for the Random strategy's random number generator (default value is 0.5).
The class features a simple interface that offers two methods:

- configure(self, random: bool = False, interleaving: bool = False) - this method defines the current strategy. There are three available strategies:
  1. Interleaving strategy - this strategy chooses either SimpleComedian for even calls or DeepComedian alternately for odd calls.
  2. Random strategy - this strategy randomly selects either DeepComedian or SimpleComedian according to the defined probability threshold.
  3. Default strategy - this strategy selects only DeepComedian.

- render(self, type, utterance: str) -> str - this method provides the interface to the Comedian render() method with the same signature. When called, it calls this method of either DeepComedian or SimpleComedian and returns the generated string.

4.2. DeepComedian and SimpleComedian

Includes the DeepComedian and the SimpleComedian as implementations of the DARVAH's Comedian sub-component.

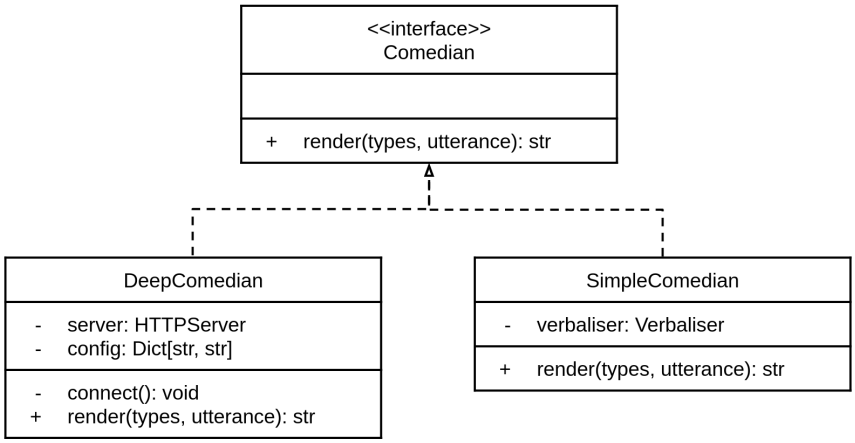

Figure 4.3.: The UML diagram for Comedian classes.

4.2.1. Language Model

As a baseline Language Model for the rendering state *R*, we have selected the 744-million-parameters GPT-2-L model [16]. The model is the second biggest one of all currently available GPT-2 models, which is beneficial for solving Natural Language Generation problems because more parameters yield better quality. At the same time, the biggest model, GPT-2-XL, offers only a minor increase in NLG quality while being too large to run locally on our hardware [10].
To test how well the GPT-2-L model generates text, we provided an initial prompt for completion denoted by IP.

- IP: Why are robots shy?

The model generated following completions (here, we selected shorter clauses from all generated samples):

- A1: The answer is likely a combination of genetics and programming.
- A2: They are not shy because they are not human.
- A3: Robots don't like getting too close to people or touching people.
- A4: It's hard to say for sure.
- A5: Because it takes a lot to get one's attention.

- A6: It's not what they're afraid of but what you're afraid of.

All clauses form comprehensible phrases (excluded ones as well). However, their humorous effect is low when compared to a human-generated answer HA:

- HA: Because they have hardware and software but no underwear.

To overcome this drawback, we decided to modify the model via transfer learning.

4.2.2. Transfer Learning

Transfer learning is a concept which allows applying previously learned knowledge from one domain or problem to another domain or problem. The process is similar to how humans can leverage their mental models based on ideas which they learned before [17]. However, unlike humans, Roboy cannot do it automatically and requires us to adjust the model.
We could instead train the GPT-2-L from scratch using our dataset of jokes. Usually, in our case, it is impractical because the GPT-2-L features a substantial size. It is a Deep Model comprising 774 million parameters trained on a dataset of 8 million webpages (40 GB of plain text) [10]. Thus, the model's size poses a challenge to the data collection and makes it infeasible to train by us because of the extreme computational requirements. However, transfer learning helps us to solve this issue by utilising already trained parameters' values.
Transfer learning alleviates Deep Learning issues in three ways: lower data requirements, smaller memory and processing requirements, shorter training time. For example, training a full GPT-2-L might take several weeks while the transfer learning allows having a new model in under a day depending on the size of the dataset [17].
To adapt the model for transfer learning, we could follow either of two approaches. The first one is to keep the pre-trained model architecture and weights. This way, we need to add several new layers at the output of the original model, or we can use the model output as input to a new smaller model. This approach is suitable if the target problem stays the same.
Another method is to modify the internal architecture of the original. The approach is useful if we want to adapt to a different target problem in the same domain. In this case, we could have used the pre-trained model to initialise a new model, including problem-specific changes such as adding skip or residual connections or using bottleneck modules between the layers of the original model.
However, in our case, the optimal choice was to adhere to the former approach due to two reasons. Firstly, the domain stays similar because the GPT-2-L provides the English language model while we would like to have the English humour (language) model, which narrows down the original domain to more specific textual representations. Secondly, the size of the GPT-2-L architecture makes it unfeasible to pursue the second approach due to time and computational constraints. Finally, the target task remains the same - generation of natural language sentences.

To perform transfer learning using GPT-2-L, we update Transformer layers (see Figure 2.3) with the encoded joke data using original GPT-2 model and encoder from its GitHub repository [16].

4.2.3. Roboy Jokes Dataset

To assemble the dataset, we decided to employ online data sources since there is not much ready-to-use joke data available publicly. For most of our data, we used the Pushift project [18] to access cached data from Reddit forums (subreddits). The collected data consists of text posts from the following subreddits from the original date of the forum creation until 01.01.2020:

- "jokes",
- "darkjokes",
- "dadjokes",
- "cleanjokes",
- "oneliners",
- "badjokes",
- "dirtyjokes",
- "DarkJokeCentral"

After we downloaded the data, we stripped it from all functional characters, removed empty and deleted samples, removed the jokes with offensive words or nonsensical data and chose their class labels. We also added data from the publicly available joke-dataset [19] containing 208000 jokes from www.reddit.com, www.stupidstuff.org, www.wocka.com. All of the collected raw data contained over 1350000 data points. The resulting clean dataset contains over 800000 jokes.
Each line in the dataset is a separate joke. We annotated each line with an <|endofline|> label at the end of the line, to delimit where the previous joke ends and a new one starts. At the beginning of each line, we appended multiple class labels according to the following idea.
First, we split the jokes into groups according to their sizes. The introduced size classes are:

- "short" for lines of length smaller or equal to 200 characters.
- "medium" for lines of length bigger than 200 and smaller or equal to 400 characters.
- "long" for lines of length bigger than 400 and smaller or equal to 800 characters.
- "story" for lines of length bigger than 800 and smaller or equal to 1600 characters.

The choices of the lengths for the classes were based on the time it takes to pronounce the spoken utterances.
Besides the size classes, we split the dataset into 20 content classes based on token frequency. For example, we defined class c_1=<|c_1|> through the token "chicken" and assigned the class to all samples for which the frequency of the token "chicken" was higher than other tokens' frequencies. The whole list of the class token contains 20 words: "chicken", "momma", "trump", "pet", "army", "police", "religion", "bar", "clown", "german", "date", "queer", "boss", "doctor", "british", "cookie", "family", "friend", "dadjokes", "other". All classes that did not have a suitable token were assigned the token "other" for c_{20}=<|c_20|>. However, during the initial joke generation testing, we realised that this approach introduced two disadvantages. First, many generated jokes were very close to the original data, which indicated that that the model was overfitting. Although, in our case, overfitting is not an entirely negative phenomenon since it is, for example, similar in nature to people reading and retelling jokes from elsewhere to their friends. However, we still wanted the model to produce more creative results.
The second disadvantage is that the set of classes was not extendable after the training process was finished. Usually, this is not a problem. For example, with the size classes, they can change only globally on the whole dataset. Thus we will need to retrain the model. However, let's consider the following situation.
We would like to generate a reactive response and we receive the following input utterance:

- U^i_{t-1} ="Why are robots shy?"

For example, we selected l_1=<|l_1|> size class. However, it is unclear which class we have to select from the content classes because, for the utterance U^i_{t-1}:

$$Subject(U^i_{t-1}) = "robots"$$

which is not in the list of class tokens, thus the only viable choice would be to use token "other" and the associated class c_{20}=<|c_20|>. We tried generating a response and obtained the following result:

- Input prompt: "<|l_1|> <|c_20|>Why are robots shy?"
- U^r_t ="They have lower case letters than police."

which is semantically ambiguous.
Therefore we decided to switch to plain-text class labels. We reasoned that if the model can connect the label with the words in the joke string, we might be able to add more classes at runtime if needed. We replaced all labels according to their class token as in:

- l_1=<|l_1|> $\rightarrow$ l_1=<|short|>
- c_1=<|c_1|> $\rightarrow$ c_1=<|chicken|>

Afterwards, we retrained our model and tested the same example with the new class labelling schema. At runtime, we generated a new class label c_{21}=<|$Subject(U^i_{t-1})$|>=<|robots|> and tried generating a reactive response. We obtained the following result:

- Input prompt: "<|short|> <|robots|>Why are robots shy?"
- U^r_t ="The only job robot does well is driving a bus."

which greatly improved semantically.
The change also slightly reduced overfitting of smaller jokes due to less restrictive labelling since the class labels are not unique strings.

4.2.4. Training Process and Resulting Model

We used download_model.py script from OpenAI GPT-2 GitHub repository [16] to obtain their original pre-trained GPT-2 774M model. Although OpenAI originally trained the model using Tensorflow 1.12.0, it is fully compatible with Tensorflow 1.15.2 which we used for training. The model's checkpoint is over 3 GB in size and uses a Tensforflow 1.0 checkpoint format:

- model.ckpt.data-00000-of-00001 - contains all variables' values
- model.ckpt.index - describes where tensors' values are stored in the data file
- model.ckpt.meta - contains model's graph structure

The GPT-2 loading also makes use of auxiliary files such as: hparams.json (contains hyper-parameters), encoder.json (contains mapping numbers for words and symbols), and vocab.bpe (contains byte-pair encodings).
Before training, we had to pre-process our dataset of jokes using the encode.py script. The resulting npz file contains 26536988 encoded tokens.
For the Transfer Learning task, we fine-tuned the model on our pre-processed dataset by updating only the Transformer layers (see Figure 2.3 in Chapter 2) employing the gradient checkpointing feature to reduce video memory usage. To be able to use several GPUs, we used a modified GPT-2-simple [20] library which allows easy distributed retraining of GPT-2 models on multiple GPUs.
For training, we configured our Google Cloud instance with:

- 10 virtual CPUs
- 64 GB of RAM capacity
- 64 GB of SSD capacity
- 2 x NVIDIA Tesla V100 with 16 GB of VRAM capacity each

Using this instance, we fine-tuned our final GPT-2 TLH model for 100 000 iterations on two V100 GPUs simultaneously with the following parameters:

- batch_size=1 - use one sample per batch
- learning_rate=0.0001 - default learning rate for Adam optimiser
- accumulate_gradients=5 - accumulate 5 latest gradients
- multi_gpu=True - use distributed training
- use_memory_saving_gradients=True - save gradients to reduce video memory usage
- only_train_transformer_layers=True - fine-tuning the Transformer layers
- optimizer='adam' - using Adam, an adaptive learning rate optimisation

This configuration utilised 18 GB of VRAM and would exceed the available memory otherwise. The training process took around 150 hours. We used the resulting fine-tuned GPT-2 TLH model in the DeepComedian implementation.

4.2.5. DeepComedian

DeepComedian is an implementation of the Comedian interface (see Figure 4.3) which offers the single method:

- render(self, type: List, utterance: str = None) -> str, which accepts a list of joke types and an utterance (in case of reactive responses) and returns a string generated by our Language Model.

When initialised, DeepComedian tries to connect to the GPT-2 TLH server. If the server does not exist, DeepComedian starts a new instance and connects to it. The server implements simple HTTPServer and BaseHTTPRequestHandler from the built-in Python http library.
Inside BaseHTTPRequestHandler, we call GPT2TLHBackend object that returns a completion from the GPT-2 TLH model based on passed types and utterance. When the server starts, GPT2TLHBackend pre-loads the GPT-2 TLH model in a TensorFlow InteractiveSession as shown in Listing 4.4. Until the server shuts down, the InteractiveSession allows to keep the model in memory and access it when needed to lower the processing time.
When the response is ready, we randomly attach a "Ha-Ha!" token at the end of the string, so that the Speech-To-Text service can generate a respective "laughter" sound in the wave sample.
Besides the server capabilities, we also provide a simple tool offering a bash command-line interface to render jokes manually which is based on a similar script from OpenAI (available in GPT-2 GitHub repository) [16].
Furthermore, we developed a Telegram bot that generates jokes for users and allows them to rate each generated joke. The bot uses python-telegram-bot 12.4.2 library to implement Telegram functionality and communicates to the DeepComedian server via a web-socket to create jokes.

```
sess = tf.InteractiveSession()
context = tf.placeholder(tf.int32, [batch_size, None])
np.random.seed(seed)
tf.set_random_seed(seed)
output = sample_sequence(
    hparams=hparams,
    length=length,
    context=context,
    batch_size=batch_size,
    temperature=temperature, top_k=top_k, top_p=top_p
)

saver = tf.train.Saver()
ckpt = tf.train.latest_checkpoint(os.path.join(self.path,
self.model_name))
saver.restore(sess, ckpt)
```

Figure 4.4.: Loading the pre-trained GPT-2 TLH model in TensorFlow InteractiveSession.

4.2.6. SimpleComedian

SimpleComedian is an implementation of the Comedian interface which offers the single method:

- render(self, type: List, utterance: str = None) -> str, which accepts a list of joke types and an utterance (in case of reactive responses) and returns a string from the pre-generated list of jokes.

When initialised, SimpleComedian loads a YAML file containing pre-generated jokes, split into groups according to the jokes' categories, into the Ravestate Verbaliser module. Then, when ComedianStrategy calls its render() method, SimpleComedian passes the joke type into the Verbaliser which returns a random joke with this type.

4.3. EmotionStrategy

EmotionStrategy is a class that implements the strategy pattern for the emotion evaluation process (see Figure 4.5). When initialised, it creates both VideoEmotion and AudioEmotion objects (see Section 3.4 in Chapter 3). The constructor accepts two parameters α and β which are the weights parameters for FER and SER respectively (see Section 3.4 in Chapter 3).
The class features a simple interface that offers two methods:

- configure(self, video: bool = True, audio: bool = True) - this method defines the current strategy. There are three available strategies:

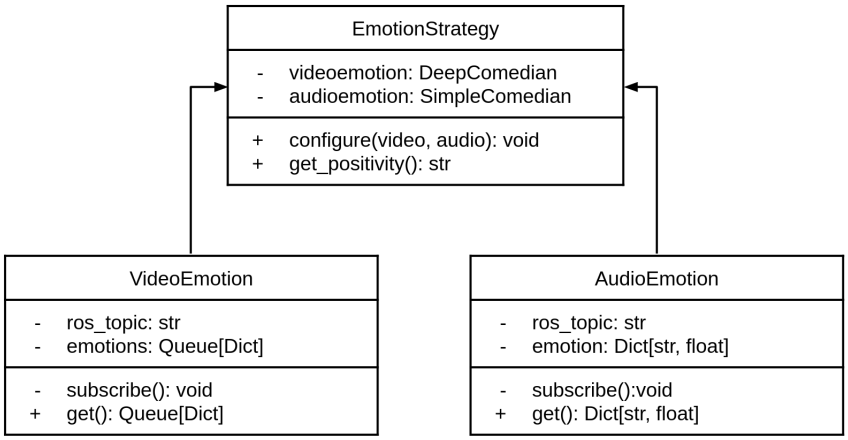

Figure 4.5.: The UML diagram for EmotionStrategy.

1. VideoAudio - when both video and audio parameters are True, this strategy calculates the weighted sum of FER emotion vectors and SER emotion vectors
2. VideoOnly strategy - this strategy chooses only VideoEmotion results and discards AudioEmotion results
3. AudioOnly - this strategy chooses AudioEmotion results and discards VideoEmotion results

- get_positivity() -> int, which returns the emotion positivity score as per Section 3.1.4 in Chapter 3.

AudioEmotion contains the single Python dictionary [String, Float] field containing seven entries representing a mapping from 7 Rosemo emotions to their emotion recognition scores and a getter method for the data. When EmotionStrategy initialises the AudioEmotion object, the object subscribes to the Rosemo /rosemo_ser_pub topic.
VideoEmotion comprises the a custom queue implementation field that contains $m = f \times w$ dictionaries [String, Float] for each FER emotion vector containing 7 entries representing a mapping from 7 Rosemo emotions to their emotion recognition scores. It also offers a getter method for the data. When EmotionStrategy initialises VideoEmotion object, the object subscribes to Rosemo /rosemo_fer_pub topic.

4.4. Rosemo: EmoPy + SER

4.4.1. ROS Interfaces

Using rospy, we implemented the following ROS topics to communicate from Rosemo to Funboy:

- /rosemo_fer_pub - the topic for Facial Emotion Recognition publisher using EmoPy
- /rosemo_ser_pub - the topic for Speech Emotion Recognition publisher using SER

We also implemented the following ROS message:

- string[] labels
- float64[] scores

where the string array contains labels of the recognised emotions, and the float array contains emotion recognition scores for each label.

4.4.2. Rosemo

Rosemo is capable of recognising seven basic emotions' labels: calm, anger, happiness, surprise, disgust, fear, sadness - both in speech recordings and in facial image data.
The software module consists of two parts. The first one is video.py script that regularly collects image frames from the camera and processes them. First, it crops the face of the user in the frame using OpenCV. Then it converts the frame into the grey-scale colour range also using OpenCV. After the conversion, it passes the image to EmoPy FERModel classifier. The obtained result is published via ROS to /rosemo_fer_pub topic.
The second module is speech.py. When it detects that the user is talking, it records their speech in wave 8 bit, 16kHz format. Then it passes the recording to the SER CNN-LSTM model. The obtained result is published via ROS to /rosemo_ser_pub topic.

4.5. Interlocutor Profiles

To adapt the interlocutors' data for the DARVAH humour affinity scores, we needed to make minor changes to the Ravestate ontology. Therefore, we used the default ontology and made two changes. Firstly, we added new AFFINITY relationship to !OType Person, which is the type used for interlocutors and other known people. Then, we added a new !OType - JokeType:

- entity: JokeType
- properties: [name, timestamp]
- relationships: [EQUALS]

4.6. Speech Processing

In the Ravestate Funboy implementation, we based our speech processing capabilities on Roboy Soncreo and roboy_speech_recognition software. Both of these packages are own

Roboy distributions developed by us utilising pyAudio library for sound processing capabilities. In this project, we used the versions based on Google Cloud Text-to-Speech and Cloud Speech-to-Text services. These modules are accessible from Ravestate via ROS using the following interfaces.
ROS topic for roboy_speech_recognition which captures the speech via the microphone stream and publishes a string of text recognised from the stream:

- /roboy/cognition/speech/recognition
- message type: RecognizedSpeech
  - int16 source
  - string text
  - float64 start_timestamp

ROS service for Roboy Soncreo - creates and plays a wave file from a passed string of text:

- /roboy/cognition/speech/synthesis/talk
- service type: Talk
  - input: string text
  - output: bool success

5. Experiments and Results

In this chapter, we will describe the conducted experiment. The goal was to observe human reactions through image and audio recordings of the participants in conditions simulating exposure to different styles of humour delivered by a humanoid robot Roboy 3.0.

5.1. Funboy Experiment

To perform the experiment, we needed to expose the participants to a spoken Natural Language Dialogue with Roboy. The Roboy remarks could contain either conventional Ravestate dialogue utterances or humorous statements derived from the DeepComedian's GPT-2 TLH model. We did not use any pre-generated humorous expressions due to two reasons. Firstly, it would introduce a noticeable variance in delay between the input U_{t-1}^{i} and output U_{t}^{i} depending on whether the output was pre-generated or not. Secondly, if the response U_{t}^{i} was pre-generated, we would not be able to incorporate reactive responses into the dialogue flow. Therefore, the system produced all humorous responses at runtime via the DeepComedian submodule.

5.1.1. Joke Categories

For the experiment, we conditioned the model using the following types of humour:

- Size types: <|short|> and <|medium|>. Due to the speech synthesis computational cost, we decided to limit the size of the generated text. Otherwise, when the system produced a wave file containing synthesised speech, the cumulative delay would exceed 5 seconds, which we considered as a critical cut-off for the verbal exchange.
- Content types:
  - <|chicken|> - c_1: all samples for which the frequency of token "chicken" is higher than other tokens
  - <|momma|> - c_2: all samples for which the frequency of tokens "yo, momma", "your, momma" or "yo, mama" is higher than other tokens
  - <|trump|> - c_3: all samples for which the frequency of token "trump" is higher than other tokens
  - <|pet|> - c_4: all samples for which the frequency of tokens "pet", "cat" or "dog" is higher than other tokens

- <|army|> - c_5: all samples for which the frequency of tokens "army", "military" or "soldier" is higher than other tokens
- <|police|> - c_6: all samples for which the frequency of tokens "police" or "cop" is higher than other tokens
- <|religion|> - c_7: all samples for which the frequency of tokens "religion", "christian", "muslim", "jesus", "god", "buddha" is higher than other tokens
- <|bar|> - c_8: all samples for which the frequency of tokens "bar", "barman" or "bartender" is higher than other tokens
- <|clown|> - c_9: all samples for which the frequency of tokens "clown" or "circus" is higher than other tokens
- <|german|> - c_10: all samples for which the frequency of token "german" is higher than other tokens
- <|boss|> - c_11: all samples for which the frequency of token "boss" is higher than other tokens
- <|doctor|> - c_12: all samples for which the frequency of tokens "doctor" or "nurse" is higher than other tokens
- <|british|> - c_13: all samples for which the frequency of tokens "english", "british" or "brexit" is higher than other tokens
- <|cookie|> - c_14: all samples for which the frequency of tokens "cookie", "biscuit" or "shortbread" is higher than other tokens
- <|family|> - c_15: all samples for which the frequency of tokens "mom", "mum", "mama", "mother", "brother", "sister", "father", "grandfather", "granny", "grandpa", "grandmother", "grandma", "parent" or "family" is higher than other tokens
- <|friend|> - c_16: all samples for which the frequency of token "friend" is higher than other tokens
- <|dadjokes|> - c_17: all samples that belong to the subset of jokes derived from "r/dadjokes" subreddit but do not belong to other classes
- <|other|> - c_18: the rest of the samples

5.1.2. System Configuration

In the experiment, we used Roboy 3.0 as the joking humanoid robot.
Hardware configuration setup comprised the following equipment:

- Workstation running all local software:
  - CPU: Ryzen 3900X
  - GPU: Nvidia RTX 2080Ti

  - RAM: 32 GB
- Logitech HD Pro Webcam C920 camera to record images for VER and speech for SER
- Standalone microphone to record speech for ASR
- Loudspeaker for sound output
- Projector for Roboy face projection

Software setup comprised the following modules and packages:

- Software modules and packages:
  - GPT-2 TLH based on GPT-2-L and trained for 100 000 iterations using the defined joke categories (see Section 5.1.1)
  - Ravestate 0.7.0 for Dialogue System capability including:
    * Ravestate Funboy as the DARVAH framework implementation
    * Ravestate Wildtalk (version from 24.09.2019) running ConvAI GPT model
  - Docker running the following containers:
    * Funboyn4j with Neo4j instance for the DARVAH ontology
    * Rosemo with EmoPy and SER
  - Roboy face Android application to provide face projection and facial emotion animations
  - Roboy Socnreo for speech synthesis
  - roboy_speech_recognition (Google) for speech recognition

5.1.3. Experiment Setup

We conducted the experiment as a one-to-one conversation between a participant and the robot. The researcher explained the goal and the flow of the interaction to the participant before the start. The researcher was always present in the vicinity and controlled the beginning and the end of each experiment. Although we did not wholly restrict access to the room, we isolated the part of the room where the experiment took place from other people. A table separated Roboy from the participant who sat in the chair across. We recorded each participant from a frontal view by a camera and microphone placed on Roboy while they were communicating. This was necessary to capture natural visual and audial cues as an involuntary response to humorous input. Although we did not specifically inform the participants about the conversation structure or that it would contain any comedic elements, all of the participants had an idea about the humorous component of the experiment. There was no time limit for conversations. Thus, each participant could choose when to finish the interaction themselves. We structured the flow in each conversation in the following way:

1. The greeting – in this part, after the participant greeted Roboy, Roboy greeted the participant.

2. The conventional dialogue – in this part, Roboy asked or answered simple questions.

3. The humour-enabled dialogue – in this part, Roboy started to use Funboy's D state to decide whether to tell jokes or use conventional responses. The part came into effect after 90 seconds since the start.

4. The farewell - in this part, the participant finished the conversation by saying farewells to Roboy.

We prepared this conversation structure to pursue two objectives. The first one was to observe how emotions and behaviour of the participants change when the robot starts joking. Thus, we did not start the Funboy's DARVAH loop immediately after the whole system start-up but introduced an 80 seconds delay so that the participants had a chance to experience a pure conventional Ravestate dialogue.
During each conversation, we collected the image and audio streams and processed them at near real-time via rosemo models to obtain the classification of emotions expressed by the participant.
We collected the visual and audial information, under the assumption that they may contain behavioural patterns indicating responses to a humorous input which may lead to understanding if people are more willing to engage in an interaction with Roboy, given the conditions of the experiment.
The system continuously acquired latest w =10 seconds of recognised emotions from the image stream. We chose this size of the window via testing the setup before the experiment. Given that we configured rosemo.image to use a frequency of 2 fps[1], the V state received 20 latest samples each time the validation process occurred. We calculated the positivity value based on these samples to validate the emotion data for attuning the affinity scores.
Although we recorded the speech as well, we encountered a certain problem trying to use the obtained emotion samples. Unfortunately, due to the delays in response and processing tasks on Roboy's side and the necessity to record the whole user input to recognise the emotions, we could not use the speech emotions predictions for calculating the positivity value considering the time elapsed from the moment we collected the data until we obtained a speech emotion recognition result. Therefore, in this experiment we had to select VideoOnly strategy in EmotionStrategy class to set parameter $\beta = 0$ for the SER.
After the interactive part of the experiment, we asked each subject to answer a set of questions designed to gauge their involvement in the process.

[1]frames per second

5.1.4. User Study Questions

The questionnaire consists of 6 parts. The first one is a set of four questions on how anthropomorphic Roboy seemed to the participants. Each question offered a modified Likert scale from -2 to 2 and asked to rate the participant's impression of the robot according to the following scales:

- Q 1.1. From Fake: -2 to Natural: 2.
- Q 1.2. From Machine-like: -2 to Human-like: 2.
- Q 1.3. From Unconscious: -2 to Conscious: 2.
- Q 1.4. From Artificial: -2 to Life-like: 2.

The second part offered five questions on how likeable Roboy appeared. Each question provided a modified Likert scale from -2 to 2 and asked to rate the participant's impression of Roboy according to the following scales:

- Q 2.1. From Dislike: -2 to Like: 2.
- Q 2.2. From Unfriendly: -2 to Friendly: 2.
- Q 2.3. From Unkind: -2 to Kind: 2.
- Q 2.4. From Unpleasant: -2 to Pleasant: 2.
- Q 2.5. From Awful: -2 to Nice: 2.

The following set of questions consisted of seven fully formulated statements about Roboy and asked the respondents to rate their agreement on the standard Likert scale from 1 to 5 ("strongly disagree" to "strongly agree"):

- Q 3.1. I enjoyed the robot talking to me.
- Q 3.2. I find the robot fascinating.
- Q 3.3. Talking with the robot was easy.
- Q 3.4. I consider the robot a pleasant conversational partner.
- Q 3.5. I find the robot pleasant to interact with.
- Q 3.6. I feel the robot understands me.
- Q 3.7. I think the robot is funny.

Questions Q4 and Q5 offered the participants to write their feedback in the free form regarding what they liked and didn't like about Roboy.
The last question consisted of three parts Q6.1, Q6.2 and Q6.3, which asked to briefly describe the respondents' emotions in the beginning, the middle and the end of the conversation with Roboy. This is the most essential part of the questionnaire because it directly asks to self-evaluate the person's emotions which are a necessary component of the experiment.

5.2. Results of the Main Experiment

In this section, we will discuss the results obtained from the Main Experiment. There were 13 participants - all of them are students in STEM fields at the Technical University of Munich. Most of the respondents were familiar with the Roboy project. Further, this section covers the performance of Funboy as well as the observations about the experiment. Regarding the materials collected by Rosemo, we will concentrate on the image data. In the second subsection, we explore the self-reported assessment from the User Study questionnaire and scrutinise the reports.

5.2.1. Funboy Results

We evaluated the performance of the Funboy system using the hardware mentioned above setup:

- Speech synthesis took on average around 1 second using 600 Mbps Internet connection with about 9 ms ping to Google servers.
- EmoPy can recognise emotions on a single sample extracted from a Full HD frame (1920 x 1080 pixels) in 200 milliseconds. This means that to stay within near real-time constraints, we should require the sampling frequency $f \leq 5Hz$. In the experiment, we set the frequency to 2 Hz to have enough leverage on speed.
- For the SER, it takes around 800 milliseconds to recognise a wave sample of 20 seconds (the time scales linearly).
- Joke generation was the most resource taxing task. The <|short|> joke type takes 3 seconds on average to generate a sample while the <|medium|> type jokes were taking over 5 seconds.

Based on the data we recorded during the experiment, we could calculate the amount of time people spent talking with Roboy until the farewells. The mean conversation duration is $\mu = 853$ seconds (while the standard deviation is $\sigma = 431$). Even the shortest conversation of 347 seconds was well over the delay value of the DARVAH loop start-up, which was 80 seconds. Therefore, we consider the conversation engagement to be satisfactory because most of the participants spent enough time to familiarise themselves with Funboy.
However, longer engagement time does not necessarily mean that the participants found conversing with Roboy rewarding. Although all of the participants were enthusiastic at the start of the conversation, only four of them were satisfied by the end of the experiment.
9 out of 13 participants experienced troubles with the delay of the joke generation task between the input and output of the Ravestate Dialogue System - some of the respondents became too impatient and would repeat their remark several times.
All participants visually responded better to jokes that closely resembled canned comedy such as:

- Why did the chicken cross the road?
- How many X does it take to screw in a light bulb?
- Yo momma is so X, that she Y

Completely original jokes seemed to cause more confusion due to either being too noisy or puzzling to be perceived well.
Several participants asked the robot to repeat the previous joke or general statement during the conversation, which currently is impossible. This functionality might be useful even outside of the Funboy's DARVAH loop and should be implemented system-wide in Ravestate.
Overall, all participants wished for Roboy to be more expressive regarding the humour delivery to better distinguish between conventional conversation and humorous responses:

- Use more facial expressions
- Use verbal confirmations in the humorous responses, such as starting the response with, for example:
  - "I know a joke..."
  - "Let me tell you a joke..."
  - "Here is a joke..."
- Introduce head movements into the delivery process

However, when asked whether they would prefer canned laughter over the current laugh token approach, most of the participant agreed that the current way of adding laughter to the response seems better.
Furthermore, it is interesting to compare two of the most different results concerning the calculated affinity scores in Table 5.1 (see scores for all participants in Appendix B). User 0304J142142B spent 17 minutes 48 seconds, and User 0304L200631S spent 31 minutes 13 seconds talking to Roboy. The first user heard 33 jokes, of which the only reaction to 7 was evaluated positively. Thus, the success rate is 0,21. For the second user, the number of generated jokes was 70, and all of them induced positive evaluations, which gives us the success rate of 1. These results demonstrate the difference in how individual respondents emotionally reacted to the generated humorous responses.

Final Affinity Scores																		
User	Scores $\{c_1{:}c_{18}\}$																	
0304J142142B	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	1	1
0304L200631S	0	2	1	4	4	4	5	5	4	6	4	4	7	5	2	1	3	9

Table 5.1.: Final affinity scores for two participants after the experiment.

Regarding the EmoPy results, unfortunately, we encountered a substantial problem with Automatic Emotion Recognition results. Over the course of the whole experiment, the recognition results were continuously biased towards "calm" emotion (see Appendix C). Although we initially ignored neutral emotions such as calm and focused on validation using only binary "negative versus positive" emotions combinations approach, we cannot accept the current Facial Emotion Recognition result as reliable due to the demonstrated bias. There has to be another solution to tackle this problem in the future to be able to continue the research further.
We tried to identify the cause for such behaviour and arrived at the following possible reasons after examining our saved image data samples:

- EmoPy's FERModel might be sensitive to uneven lighting
- the FERModel's performance may substantially degrade when encountering shadows on faces
- the FER+ dataset used to train FERModel contains exaggerated facial expression samples which may skew the recognition results (see Figure 5.1)

These hypotheses need to be evaluated in future research.
We could not use the SER data evaluation in validation process due to the constraints above (see Subsection 5.1.3). However, we recorded the audio streams and tested the recognition result. Unfortunately, the SER model also demonstrated a bias towards "neutral" emotion, although, the prediction was less continuously biased. The results for the two samples are given in Table 5.2.
We tried to examine the reasons for such behaviour and formulated the following hypotheses:

- the SER model might be sensitive to background noise
- the model was trained on native speakers data which might distort the recognition for speakers who exhibit accents

These additional hypotheses also need to be evaluated in future research.

5.2.2. User Study Results

We gathered a succinct summary of the user feedback based on their answers to questions Q4 and Q5:

- The participants liked the following features about Roboy: facial animations, voice, asking personal questions, some jokes were funny, understandable sentences, looks very cute, humour generation capability, diverse humour, entertaining conversation, mimics, dark humour.

SER Evaluation		
Emotion	Disgust	Happy
Angry	14	0
Disgust	42	0
Fear	0	0
Happy	0	0
Neutral	14	100
Sad	28	0
Surprise	0	0

Table 5.2.: Examples of correct and incorrect recognition results by the SER.

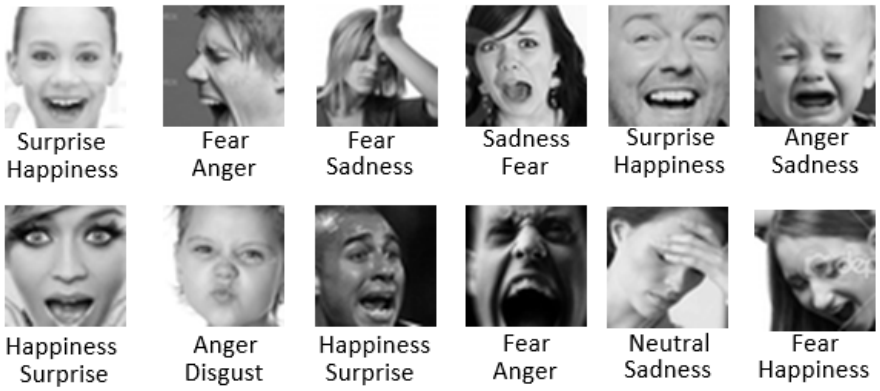

Figure 5.1.: Examples of images from FER/FER+ dataset [13].

- The participants didn't like these characteristics of Roboy: hard to distinguish between processing and listening, impossible to follow up on jokes, cannot repeat the previous response, sometimes the following response collides with the previous response, unfriendly, incoherent answers, delay in response, doesn't always understand the user's input, forgetful, interrupts the user, doesn't keep the context of the previous conversation.

Clearly, users had troubles with Roboy being incapable of following the conversation precisely. First of all, the Ravestate dialogue is missing the semantic context of the previously exchanged utterances in the conversation. Besides the contextual issues, the speech recognition service was not always properly recognising users' inputs, for example, interpreting "Hi, Roboy!" as "Jairo boy" or "Iowa". Additionally, the whole dialogue flow is still not good enough at being semantically precise regarding joke generation.
However, many participants also enjoyed that the robot tried to joke, although not always successfully. Also, the respondents reported that the facial animations were amusing. Several persons mentioned diverse humour and conversation flow as an advantage. Some of the users warned that several generated jokes might sound rude and offensive.
Furthermore, we calculated mean μ and standard deviation σ values for the questions from part Q1 (see Table 5.3), Q2 (see Table 5.4) and Q3 (see Table 5.5). Observing these values, we derived the following conclusions. Firstly, the results of the first set of questions Q1.1 - Q1.4 almost negate each other regarding the evaluation of anthropomorphism by the participants. Every question's mean value is close to 0, which corresponds to being neutral on the modified Likert scale.
Concerning the next set of questions on likeability, we obtained slightly different results. First of all, the respondents were less in agreement about their scoring, given bigger standard deviation σ value. The mean values indicate a slightly positive leaning in the ratings with the values being between 0 and 1, which corresponds to being neutral and agreeing with statements on the modified Likert scale. The highest mean belongs to the question on "liking/disliking" Roboy. The lowest mean belongs to the question on Roboy's politeness.
The fully formulated statements about Roboy demonstrated more interesting results. Generally, the participants' ratings agreed the most on this part of the questionnaire with the standard deviation values between 0.96 and 1.08. The mean values for questions Q3.1 ("I enjoyed talking to the robot") and Q3.5 ("I find the robot pleasant to interact with") are 3.62 and 3.46 respectively which is in between "neutral" and "agree". Questions Q3.3 ("Talking with the robot was easy"), Q3.4 ("I consider the robot a pleasant conversational partner") and Q3.6 ("I feel the robot understands me") were rated below "neutral", therefore, most of the respondents leaned to disagree with the statements. However, people agreed on Roboy being fascinating and funny according to values $\mu = 4.08$ and $\mu = 4.00$ for questions Q3.2 ("I find the robot fascinating") and Q3.7 ("I think the robot is funny") respectively.

Anthropomorphism		
Q	Mean μ	Std. Deviation σ
1.1	0.08	1.04
1.2	-0.08	1.32
1.3	0.15	0.99
1.4	-0.38	1.04

Table 5.3.: Means and standard deviations of answers to the question on Anthropomorphism.

Likeability		
Q	Mean μ	Std. Deviation σ
2.1	0.85	1.14
2.2	0.38	1.45
2.3	0.31	1.25
2.4	0.23	1.36
2.5	0.54	1.13

Table 5.4.: Means and standard deviations of answers to the question on Likeability.

Acceptance		
Q	Mean μ	Std. Deviation σ
3.1	3.62	0.96
3.2	4.08	0.86
3.3	2.92	0.76
3.4	2.77	0.93
3.5	3.46	0.97
3.6	2.38	1.04
3.7	4.00	1.08

Table 5.5.: Means and standard deviations of answers to the question on Acceptance.

Summary of Emotions			
User	Q 6.1	Q 6.2	Q 6.3
0228L160431P	curious, excited	excited, frustrated	disappointed
0228Y165728S	interested	confused	frustrated
0303D134007S	excited	confused, laughing	exhausted
0303D161701L	happy, curious	happy, confused	happy, surprised
0303J155544M	regular	amused	amused
0303M151400P	curious	chill, happy	a bit frustrated
0303M170134N	nice, curious	confused	a bit tired
0303P142946H	interested	confused, happy	disappointed
0303P201302N	excited	interested	bored
0303V145445S	curious, excited	a bit uncomfortable, excited	a bit annoyed
0304J142142B	excited, happy	impressed, interested	satisfied
0304L200631S	curious, excited	curious, fun	curious, fun
0304M152010H	curious, excited	frustrated, amused	frustrated

Table 5.6.: Summary of the self-reported emotions.

The last part of the questionnaire was devoted to emotion self-assessment. We compiled a summary of the reported answers in Table 5.6. Given the results, we arrived to the following conclusions:

- Overall, 4 out of 13 participants reported feeling positive emotions over the whole experiment.
- 12 out of 13 participants felt positive at the beginning of the conversation and mostly split between being either curios or excited.
- 7 participants who were excited about the experiment kept feeling positive by the middle of the conversation.
- 5 out of 13 participant felt confusion in the middle of the conversation, which might have occurred due to assertive DARVAH behaviour.
- Most of the participants tried to make sense of noisy non-jokes which was mentally taxing and resulted in 9 participants experiencing negative emotions by the end of the conversation.

Based on the User Study self-assessment, we can consider the experiment to be moderately successful. However, we discovered many drawbacks that need to be fixed before conducting further research. Mainly, the DARVAH's decision state implementation in Funboy needs to be less aggressive regarding the positive decision-making, and the processing delays need to be lower.

6. Challenges and Future

6.1. Current Challenges

In this section, we will discuss the challenges that we encountered while implementing the project and conducting the experiment. These following challenges mainly arose from how used data looked like and as well as the GPT-2 TLH model's performance.

6.1.1. Joke Types Identification

The first problem we encountered with the dataset was the researcher bias. Since we used plain-text-based class tokens such as <|chicken|> or <|cookie|>, the choice of exact tokens suffers from the bias exhibited by the researchers. This situation means that if the tokens were chosen purely according to the desire of the responsible regardless, for example, the token frequency distributions in the dataset, we could end up with a very weak prior for the classes. At the same time, a potentially infinite amount of semantically-dependent combinations between tokens doesn't allow us to solve this problem by a purely analytical approach.
Furthermore, tokens sub-string matching works only on a subset of jokes. For example, it works very well on such riddling jokes as:

- "Why did the chicken cross the road?"

because all of these jokes have the same subject and same structure. At the same time, the quality of generating other structurally invariant jokes such as:

- "I like my X as I like my Y"

suffers because these jokes can feature different objects and might get split into several classes. These erroneous splits can prevent the Language Model from properly learning the generation task.
However, we propose to solve this problem by allowing many people to vote for the class labels simultaneously. In other words, the solution could be to use an online crowd-sourcing of the label data. To enable such opportunity, we can set up a web-page which offers users to read utterances one by one sampled from our data. Then, the users can label the joke samples according to the semantic information extracted from them. For sampling the data points, we propose to use a bootstrapping approach on the original dataset.

6.1.2. Dataset Cleanliness

During the data pre-processing step, we identified another problem. Unfortunately, online data had many impurities, such as:

- disfluencies, when the line of text is interrupted by extra spaces or other unrelated characters;
- noise, when online users posted incomprehensible data; non-jokes, simply humourless texts unwanted in our data;
- damaged data when the data was not fully present after fetching;
- empty samples, when the fetched data turned out to be empty.

For this problem, we also offer to use online crowd-sourcing similar to the previous proposal. This time, the users will have first to decide if the sample belongs to jokes or non-jokes. The non-joke data is discarded after the critical number of hits. If the joke samples contain impurities, the users can edit them and save the updated variant.

6.1.3. Language Model Performance

We discovered four noticeable problems regarding the Language Model performance. The first issue was that the model learned offensive jokes better and faster than non-offensive ones. The main hypothesis is that the offensive jokes provide a stronger signal in data due to more specific tokens. Moreover, the tokens may exhibit higher co-occurrence values in offensive jokes. This effect was observable on the jokes of <|momma|> type such as:

- "Yo, momma is so X, she Y"

The next issue happened due to the dataset not being fully clean. The data impurities caused the model to learn noisy tokens which were often present across various classes such as URL word-pieces, subreddit identifiers, usernames, unique characters. We can solve this problem only by gradually making our data cleaner (see Subsection 6.1.2). The generation results sometimes contained so much nonsensical phrasing and disfluencies that the interlocutors were not capable of understanding the responses.
The last complication was not a problem of the Language Model in essence. However, due to the substantial delay between the input and the generated response, the participants of the experiment could not fully engage in the verbal interaction. The system experienced critical delays (five seconds or more) already from the response length of 50 tokens which is not enough even for the smallest length type <|short|>.

6.2. Future Research

In this section, we will offer a possible future approach which may help to advance the current implementation of Funboy further. We will focus on two main topics: emotion recognition and humour generation.

6.2.1. Visual Cues Detection Model

To recognise facial emotions more decidedly, Roboy can utilise some visual cues detection model which has to be more specific and sensitive than conventional emotion classification model, because we are interested in asymmetric quantitative prediction in a humour perception domain. For example, our current emotion classifier can infer whether a person is calm, angry or happy.
To train such cues detection model, we will need to collect more image data on how interlocutors interact with Roboy. In the following step, we could compare the timestamps of the recordings with the timestamps of the humorous and non-humorous responses. In this way, we can pinpoint the intervals coinciding with a specific humorous situation. Afterwards, we will manually determine the data points containing useful information. These frames will act as data points for our visual cues dataset, under the assumption that they may provide behavioural patterns indicating a reaction to humorous input. We could use this dataset to apply Transfer Learning technique to achieve better recognition accuracy already. It is important to note that due to very narrow domain (interacting with only one particular robot) the resulting model will be biased towards Roboy interacting with other people and might show worse results when used with other robots.
Furthermore, for the model, we could introduce four possible classification scenarios while interacting with an interlocutor: the interlocutor is amused given humorous interaction $VC(A|H)$; the interlocutor is amused given no humorous interaction $VC(A|NH)$; the interlocutor is not amused given humorous input $VC(NA|H)$; the interlocutor is not amused given lack of humorous interaction $VC(NA|NH)$.
The first situation denotes the most crucial evaluation for social robots since it allows gauging interlocutors' reactions to the specific style of humour being employed at the moment. However, at the same time, it poses a challenge to the classification task, since it requires not only a qualitative result but also a quantitative assessment according to a yet-to-be-defined scale of amusement (positive reward values). The resulting score is then added to the specific style of humour value in the interlocutor's profile. The second scenario is almost equally important because it provides a regularisation estimation considering the following problem: a person who is currently engaging in a conversation with a robot might be affected by the novelty of such interaction, which might lead to a sustained state of amusement regardless of humorous undertakings. Thus, the model will have to subtract $VC(A|NH)$ from $VC(A|H)$, effectively regularising the prediction. In the third case, the robot assumes that the person is not interested in this type of joke. Thus, it has to assign a negative reward to this style of humour. However, the

negative reward is a constant negative value, since the model cannot predict whether the interlocutor did not like the joke, or the delivery was substandard. Therefore, it might penalise the joke too harshly in case of a quantitative evaluation. The latter situation is trivial; it does not provide any information gain since it is the default state in any arbitrary verbal interaction.

6.2.2. Better Language Model

While testing of the implementation and during the experiment, we have observed many "almost funny" generated responses. It is evident that the GPT-2 TLH model grasped the concept behind some joke types. However, there was not enough information to learn about how successfully the model interpreted the joke's category. To improve this situation, we propose to utilise Reinforcement Learning on top of the current Language Model.
Reinforcement Learning is a learning technique that concentrates on learning the actions needed to maximise the reward instead of finding the association between two sets of data. In this way, RL is different from both supervised and unsupervised approaches. A learning agent has to try different variations of actions to improve the amount of reward gained on the next action [21]. Therefore, we find this idea appealing to adjust the weights of our Language Model.
However, we would like to combine the Reinforcement Learning process with our developed telegram bot (see Chapter 4) so that many more people can rate the jokes and reward or punish the model respectively directly utilising the human reasoning in the task. This approach also will not require to interact with the whole experiment setup simplifying the task.

7. Conclusion

The aim of the thesis was to explore how people react to jokes produced by a Neural Language Model conditioned on humour data and delivered by a robot in verbal interaction. We had to develop the DARVAH loop and implement it in the Funboy module for Ravestate, our Dialogue System, to be able to produce humorous utterances when needed. We achieved the humour generation capabilities by utilising Transfer Learning on the state-of-the-art GPT-2-L Language Model to condition it on the humour data using over 600 000 jokes that we collected using online sources, mainly Reddit forums. Besides the humour generation capabilities, it was necessary to implement the emotion recognition functionality for our robot. To accomplish this task, we employed EmoPy and Speech Emotion Recognition (SER) software packages which we had to adapt to our system architecture such as ROS 1.0 interfaces, Docker deployment and background processing. We combined these to parts together to form a feedback loop for the DARVAH generation, evaluation and adjustment. To store the information about interlocutors' affinity scores, we used ScientIO combined with Neo4j.
Using the implemented modules, we conducted the experiment to observe the participants behaviour. The respondents have also answered questionnaires to provide their self-assessment on the experiment. We obtained and investigated both automatic emotion recognition and self-assessment results to answer how people react to joking robots. The main conclusion is that the people were mainly confused and frustrated due to several reasons:

- Considerable delays between the input and the output because of the high computation cost of running GPT-2 models.
- Disfluencies, lexical noise and non-jokes in the humour generation results. Semantically unrelated responses both from Ravestate Funboy and Wildtalk modules.
- The Ravestate Wildtalk module that acted as a non-humorous alternative provides only a small range of answers and is not deterministic.
- Imperfect Automatic Speech Recognition which suffers from accents and background noise.

However, the participants also found the whole situation, when the robot tries to joke, funny and endorsed the idea.
To simplify current and future development, we divided the system into separate parts integrated via ROS and web-socket interfaces. The modules of the system can act independently:

- Funboy is a Ravestate module that implements the main loop of the DARVAH framework and supports the humour generation and evaluation capabilities. The module comprises functionally independent states that support common signals-and-conditions interface. The internal implementation of each state is subject to any possible improvements in the future:
  - Deduce state - decides if the current conversation requires a humorous response.
  - Assess state - retrieves interlocutor's data and selects humour types based on their affinity scores.
  - Render state - using the chosen humour types, generates a proactive or reactive humorous response.
  - Validate state - receives and evaluates emotion scores vectors from Rosemo.
  - Attune state - based on the emotions evaluation, adjusts the affinity scores for the humour types.
- GPT-2 TLH - is a GPT-2-based DNN Language Model conditioned on humour data that generates jokes based on passed joke types. It is also deployable as a standalone server.
- Funboyn4j - is a Docker container that contains Neo4j graph database storing the affinity scores of interlocutors and other Ravestate data.
- Rosemo is an emotion recognition module that contains two packages for image and audio emotion recognition. The module supports ROS 1.0 interfaces and Docker deployment.

We have also developed an additional software package that creates a Telegram bot that we can use in the future to allow people to score jokes generated by GPT-2 TLH. In the future, this information can help us to not only create a more advanced approach to the assessment and the attuning processes in the DARVAH framework but also to create a more advanced Humour Language Model on top of the existing one.
Although the humour generation and emotion recognition tasks did not fully demonstrate their potential at the current state of both technologies, they successfully showed their prospects as proof of concept. The experiment and the obtained results (in Chapter 5) demonstrated that it is possible to develop a Dialogue System that can tackle humour generation and adaptation problems at a satisfactory level given the state-of-the-art in Natural Language Processing.
Therefore, we see potential in developing the project further by addressing the challenges and solving the problems stated in Chapter 6 as well as taking the research to the next level considering our proposals. The future advances in humour applications in Dialogue Systems may uncover great opportunities to improve verbal Human-Robot Interaction.

A. ComedianStrategy

```
class ComedianStrategy:
    def __init__(self, threshold = 0.5):
        self.deep_comedian = DeepComedian()
        self.simple_comedian = SimpleComedian()
        self.count = 0
        self.threshold = threshold

    def configure(self, random: bool = False, interleaving: bool = False):
        self.random = random
        self.interleaving = interleaving
        if self.interleaving:
            self.count += 1
            if self.count % 2 == 0:
                self.comedian = self.simple_comedian
            else:
                self.comedian = self.deep_comedian
        elif self.random:
            if self.threshold < rand.random():
                self.comedian = self.simple_comedian
            else:
                self.comedian = self.deep_comedian
        else:
            self.comedian = self.deep_comedian

    def render(self, type: List, utterance: str) -> str:
        if self.comedian is not None:
            return self.comedian.render(type, utterance)
        else:
            logger.error("No comedian specified!")
            return ""
```

Figure A.1.: The main methods of the ComedianStrategy class.

B. Final Affinity Scores Result

Final Affinity Scores																		
User	Scores $\{c_1{:}c_{18}\}$																	
0228L160431P	4	0	3	5	0	3	2	1	1	1	0	3	0	0	0	0	0	0
0228Y165728S	3	0	1	0	0	1	3	0	0	2	0	1	0	0	0	0	0	0
0303D134007S	0	2	2	1	0	1	1	1	2	0	0	1	2	0	2	1	0	2
0303D161701L	0	4	0	1	5	0	3	0	8	2	0	4	0	0	2	0	6	1
0303J155544M	0	3	0	2	0	0	0	3	0	0	0	0	0	0	3	0	1	0
0303M151400P	0	1	0	0	3	0	0	1	2	1	0	0	0	3	1	1	2	1
0303M170134N	0	1	1	0	1	2	1	2	2	3	1	1	2	1	1	2	0	3
0303P142946H	3	3	0	2	5	3	0	2	4	0	1	1	4	0	4	1	3	0
0303P201302N	0	3	2	3	1	0	3	3	2	1	5	4	2	0	2	1	1	3
0303V145445S	1	1	1	2	2	0	1	0	0	0	0	1	0	0	0	0	3	2
0304J142142B	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	1	1
0304L200631S	0	2	1	4	4	4	5	5	4	6	4	4	7	5	2	1	3	9
0304M152010H	3	2	0	1	4	1	4	2	0	6	2	2	0	1	2	1	1	1

Table B.1.: Final affinity scores in Funboy for all participants in after the conducted experiment.

C. EmoPy Misclassification Example

	t_{end-4}	$t_{end-3.5}$	t_{end-3}	$t_{end-2.5}$	t_{end-2}
Calm	**0.9649**	**0.9546**	**0.9454**	**0.9260**	**0.9721**
Anger	0.0008	0.0010	0.0010	0.0014	0.0006
Happiness	0.0228	0.0287	0.0345	0.0427	0.0223
Surprise	0.0003	0.0005	0.0006	0.0008	0.0001
Disgust	0.0099	0.0135	0.0168	0.0256	0.0044
Fear	5.1391e-05	8.2555e-05	0.0001	0.0001	4.4135e-05
Sadness	0.0010	0.0013	0.0013	0.0031	0.0002

Figure C.1.: Example of the VideoEmotion frame with misclassified EmoPy result.

List of Figures

List of Tables

Acronyms

ASR Automatic Speech Recognition. 41

DARVAH Deducing, Assessing, Rendering, Validating and Attuning Humour. 3, 4, 15, 24, 41

DNN Deep Neural Network. 10, 19, 21, 25, 56

FER Facial Emotion Recognition. 22, 24, 25, 35–37, 46

NLG Natural Language Generation. 2, 4, 8–10, 18–21, 29

RL Reinforcement Learning. 9

SER Speech Emotion Recognition. 11, 12, 22, 24–26, 35–37, 41, 42, 44, 55, 63

VER Video Emotion Recognition. 41

Bibliography

[1] R. Pfeifer, H. Marques and F. Iida. *Soft Robotics: The Next Generation of Intelligent Machines*. 2013.

[2] S. Trendel, Y. Chan, A. Kharchenko, R. Hostettler, A. Knoll and D. Lau. 'CARDSFlow: An End-to-End Open-Source Physics Environment for the Design, Simulation and Control of Musculoskeletal Robots'. In: Nov. 2018, pp. 245–250. DOI: 10.1109/HUMANOIDS.2018.8624940.

[3] Roboy. *Ravestate*. GitHub Repository. 2019. URL: https://roboy.github.io/ravestate/ (visited on 12/03/2020).

[4] A. Nijholt. 'Conversational Agents and the Construction of Humorous Acts'. Undefined. In: *Conversational Informatics: An Engineering Approach*. Ed. by T. Nishida. P2773. 10.1002/9780470512470.ch2. United States: Wiley, Nov. 2007, pp. 21–47. ISBN: 978-0-470-02699-1. DOI: 10.1002/9780470512470.ch2.

[5] K. Binsted and G. Ritchie. *A symbolic description of punning riddles and its computer implementation*. 1994. arXiv: cmp-lg/9406021 [cmp-lg].

[6] G. Ritchie, R. Manurung, H. Pain, A. Waller, R. Black and D. O'Mara. 'A practical application of computational humour'. English. In: *Proceedings of the Fourth International Joint Conference on Computational Creativity (Goldsmith's, London)*. Ed. by A. Cardoso and G. Wiggins. 2007, pp. 91–98.

[7] H. Ritschel and E. André. 'Shaping a social robot's humor with Natural Language Generation and socially-aware reinforcement learning'. In: *Proceedings of the Workshop on NLG for Human–Robot Interaction*. Tilburg, The Netherlands: Association for Computational Linguistics, Nov. 2018, pp. 12–16. DOI: 10.18653/v1/W18-6903. URL: https://www.aclweb.org/anthology/W18-6903.

[8] K. Weber, H. Ritschel, I. Aslan, F. Lingenfelser and E. André. 'How to Shape the Humor of a Robot - Social Behavior Adaptation Based on Reinforcement Learning'. In: *Proceedings of the 20th ACM International Conference on Multimodal Interaction*. ICMI '18. Boulder, CO, USA: Association for Computing Machinery, 2018, pp. 154–162. ISBN: 9781450356923. DOI: 10.1145/3242969.3242976. URL: https://doi.org/10.1145/3242969.3242976.

[9] A. Radford. 'Improving Language Understanding by Generative Pre-Training'. In: 2018.

[10] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei and I. Sutskever. 'Language Models are Unsupervised Multitask Learners'. In: (2019).

[11] M. Fabien. *Multimodal-Emotion-Recognition*. GitHub Repository. 2019. URL: https://github.com/maelfabien/Multimodal-Emotion-Recognition (visited on 12/03/2020).

[12] A. Perez. *EmoPy: a machine learning toolkit for emotional expression*. ThoughtWorks. 2018. URL: https://www.thoughtworks.com/insights/blog/emopy-machine-learning-toolkit-emotional-expression (visited on 12/03/2020).

[13] E. Barsoum, C. Zhang, C. Canton Ferrer and Z. Zhang. 'Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution'. In: *ACM International Conference on Multimodal Interaction (ICMI)*. 2016.

[14] T. Arts. *EmoPy*. GitHub Repository. 2018. URL: https://github.com/thoughtworksarts/EmoPy (visited on 12/03/2020).

[15] S. R. Livingstone and F. A. Russo. *The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)*. Version 1.0.0. Funding Information Natural Sciences and Engineering Research Council of Canada: 2012-341583 Hear the world research chair in music and emotional speech from Phonak. Zenodo, Apr. 2018. DOI: 10.5281/zenodo.1188976. URL: https://doi.org/10.5281/zenodo.1188976.

[16] OpenAI. *gpt-2*. GitHub Repository. 2019. URL: https://github.com/openai/gpt-2 (visited on 12/03/2020).

[17] L. Torrey and J. Shavlik. *Transfer Learning*. University of Wisconsin. 2009. URL: http://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf (visited on 12/03/2020).

[18] J. Baumgartner, S. Zannettou, B. Keegan, M. Squire and J. Blackburn. *The Pushshift Reddit Dataset*. 2020. arXiv: 2001.08435 [cs.SI].

[19] T. Pungas. *A dataset of English plaintext jokes.* GitHub Repository. 2017. URL: https://github.com/taivop/joke-dataset (visited on 12/03/2020).

[20] M. Woolf. *gpt-2-simple*. GitHub Repository. 2019. URL: https://github.com/minimaxir/gpt-2-simple (visited on 12/03/2020).

[21] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. 2018. URL: http://incompleteideas.net/book/RLbook2018.pdf (visited on 12/03/2020).