# Chatbot report

# Abstract

The use of artificial intelligence has been increasing drastically in recent years and is often correlated with computational creativity. This is because arguably the system is using its own intelligence and creativity to produce some form of desired output. Notable inventions like open AI and artificial intelligence in games.

Computational creativity takes philosophy, art, cognitive psychology, and artificial intelligence into consideration. Discussions and studies around the topic of computational creativity raises the question about what makes a system computationally creative (Jordanus A 2012). And if so, how can we measure how computationally creative a system is? Despite whether a computer system can creative, there are various methods that are used to evaluate how creative a system. This paper proposes a project and applies one of these methods to measure the creativeness of the system in the project.

# Introduction

This paper will begin by explaining my project, a chatbot, and talking about the background on the topic of the chatbot. In this case, we will talk about deep learning in computer systems, mentioning the tools required to build the neural networks used in the chatbot. The chatbot consists of tw different types of neural networks: convolutional neural network (CNN) and Recurrent neural network (RNN). (Khosrowabadi R et all 2014).

Next, I showcase the development of my project, explaining how I designed and the results of the chatbot, elaborating on the use of Keras and Tensorflow to build the neural networks. And then how I utilised these neural networks to produce the desired output with Python. Following the design for each implementation, the results will then be shown and explained.

Afterwards, I will conduct an evaluation of the project that will include a method to measure the creativity of the system. The method used is Colton’s creative Tripod, that will enable us to measure based on three main aspects:

**Skill:** How well the system can interpret inputted information and make a reasonable response will be evaluated.

**Appreciation:** I will evaluate how well the chatbot can understand and take into consideration the context of the conversation when producing responses. For instance, if the user is behaving uninterestedly, we can evaluate how much the chatbot has acknowledged this as the situation in the context.

**Imagination:** I will measure how well the chatbot will have to use its creativity through neural networks to learn the training data. The chatbot will then have to use its creativity in how it uses the learnt dataset to engage in conversation with a user, which can also be measured.

Lastly, to finish off, this paper will conclude the discoveries of this paper.

# Background

To create this project and explain the design of the project, which we will talk about later in this paper, it is important to understand the background of the project.

Chatbots were first used in 1966, since then they are still being used today to perform numerous functions like providing customer support (Floatboat 2022). However, most chatbots used in situations like customer support typically interpret inputs and provide a response. When interpreting input, the chatbot could be computationally creative by attempting to interpret using artificial intelligence instead of simply pattern-matching via hardcoded inputs, although many may still use the latter. A good reason for interpreting inputs through artificial intelligence is so that the chatbot can understand the context of the message. If a user were to make a typo for an input that has not been hardcoded, the system would not be able to understand what the user is saying. Furthermore, the meaning of a user’s inputs can vary depending on the context of the conversation; hardcoded inputs cannot take context into consideration unless that has also been hardcoded. However, hardcoding context can be problematic and difficult since the context of a situation can vary in too many ways to reasonably consider when coding. Regarding my project, the chatbot will use artificial intelligence instead of hardcoding inputs, as it would not be computationally creative using the latter. Hardcoded inputs directly respond to instructions and do not derive their own solutions using creativity.

Neural networks have been used in machine learning to function as a brain for the machine; the idea is that they can be used to derive results based on the knowledge it has been trained with. These results can be things like predicting certain outcomes using information discovered from the data it was provided with and/or discovered. For instance, you could provide a neural network with a series of pictures of cars as training data, and then use it to detect vehicles in other images. Another example could be an algorithm that learns to control and drive a vehicle to reach a certain destination. For this project, a neural network will be required to interpret information and produce an output.

The chatbot uses Python as its source code to function. Python is a programming language that enables developers to write instructions for a computer to follow. With this, I can provide instructions like creating its own neural network, which it will utilise later to create its own solutions.

# Methodology, design, and results of the Chatbot:

The current prototype of the chatbot is a project made using python with two main functionalities, three classes, two neural networks, a JSON file, and series of text files.

## Text interpretation functionality

Its first functionality is ‘text interpretation’ functionality, which means that it interprets text inputs and delivers a response depending on what the chatbot thinks you are trying to say. It does this by matching your inputs to a list of pre-defined intents, and giving a response depending on which intent best matches your input.

**intents.json**: This is a file that contains information required for **Training.py** to function. It contains all the possible intents as tags. Each intent has a pattern array that contains a list of strings that the program should associate with the corresponding intent. Each intent also has a response array, which is randomly selected if that tag is chosen. The contents of this file were created by me. However, the tutorial I followed shows you how to create it.

**Training.py:** This file is responsible for creating and training the neural network. It works by going throughthe **intents.json** file and loads all the data by storing them accordingly into arrays and saving the data as a pickle file. The data is then converted into an interpretable numerical format as the CNN is created and learns the information right afterwards. The summary and the test and accuracy results of the CNN are printed. The loading and preparation of data were done by following a tutorial; however, the command-line interface was created by me so that you don’t have to understand the code to use the class. I also created the entire CNN myself to get a better performing CNN.

**Chatbot.py (text interpretation)**: This file is the main file that allows the users to interact with the chatbot. It is responsible for loading and utilising both the CNN model for text interpretation and the RNN model for text generation, the latter of which we will talk about later in this report. The class creates a command-line interface that takes the user’s input and uses the CNN to predict what intent the input might belong to. The program will then print back a randomly generated response in the command-line interface. In this file regarding the text interpretation, the methods and some of the command-line interface were created by following a tutorial. I improved the command-line interface by also printing out the score for each tag, so you can see the neural network’s thought process on what that intent might be for every input. I also refactored the code so that it could be compatible with the text generation functionality.

## Results of text interpretation functionality

*Figure 1.1 Figure 1.2*

Text

Description automatically generated Text

Description automatically generated

*Green text is the user’s input. Grey text is the Chatbot’s output.*

In Figure 1.1 & 1.2 the chatbot takes the user’s input and tries to match that input to the pre-defined intent that best resembles that input using a neural network. The numbers above the grey text are the score values of each possible intent the input could be. *‘[4.0963716e-10 1.0000000e+00 3.0084238e-10 6.5964678e-10 1.2737666e-10 1.2563768e-10 1.5579874e-09].’* The value with the highest score is what the neural network believes is the most suitable intent for the input.

We can see that the text interpretation is imperfect; however, in *figure 1.2*, we can see that the chatbot failed to interpret “Howdy” and “Evening” as a greeting, even though those inputs are within the “greeting’s tag”.

## Text generation functionality

The second functionality of the chatbot is its text generation functionality. By typing special commands, the command-line interface will take you through an instructional process, allowing users to easily generate text through the chatbot command-line interface.

**TrainingText.py:** This file is responsible for allowing users to create RNN models trained on the desired transcript. The RNN model and the transcript will also be required to be used in **Chatbot.py** to generate text. The program works by creating a dictionary by assigning each character to an index value and another dictionary by assigning each index value to a character. Next, we create an array that will contain a sentence with 40 characters, and another array containing the next sentence (41st character) will also be created. After this data has been converted into two NumPy arrays for compatibility purposes, the RNN can then be created and saved for later. The code used to retrieve the data and the variables created were done by following a tutorial, whereas the RNN was built by myself.

**Chatbot.py (Text-Generation):** Regarding the text generation, this file allows the user to generate text using the RNN built from **TrainingText.py** and the text transcript of their choosing. There are two commands provided: ‘**/GenerateText**’, which the command-line interface will run through instructions to utilise pre-built trained RNN models and texts. The program also provides the ‘**/GenerateCustom**’ command, which enables users to generate their own text transcript, utilising their own RNN. For both commands, settings on how they should be generated are also provided through the command-line interface. For the RNN, we use a method from the official Keras tutorial, the Keras sampling function (Keras documentation 2022). With this method, you can also adjust the temperature; the higher the temperature, the more creative RNN is. By default, the chatbot will print out your text and gradually adjust the temperature. The generate text function directly takes the first 40 characters from the transcript to use as a base to generate further text. Afterwards, it will then take the next 40 characters in sentences, convert those characters into NumPy arrays for the RNN to predict, and convert those characters back into textual format by using its index to find its initial textual format. The code to generate the textual input was built following a tutorial, whereas the command-line interface and its functionality were built by me.

## Results of text generation functionality

*Figure 1.3*

A screenshot of a computer

Description automatically generated

Here *Figure 1.3* shows the results generated from using **SimpsonS1-S5.txt** transcript and the **textgenerator\_SimpsonV2**. Furthermore, the temperature of the generated text is directly above the text. The **SimpsonS1-S5.txt** transcript consists of the transcript for every Simpson episode from seasons 1 to 5. The **textgenerator\_SimpsonV2** was a RNN built based on the **SimpsonS1-S5.txt** transcript, interpreting from 2,000 characters to 200,000 characters and interpreting 65986 words, which took my home laptop 14 hours to do roughly.

## Testing text-interpretation

When testing the text interpreter, there were three main things I took into consideration when measuring and determining its performance.

The first thing was its ability to interpret single words such as **“Hello”, “Farewell”,** etc. Which, unfortunately, the text interpreter often struggled with. Commonly, when greeting the chatbot, the text interpreter would interpret this as the user saying goodbye and the same vice versa. Adjusting the neural network sometimes fixes this, but so far, I have found the chatbot struggles with this. *Figure 1.4* some of the results.

The second thing I tested was its ability to interpret multiple word sentence inputs such as **“I am doing alright”, “I’m sad”,** etc. With this, the text interpreter was always accurate in matching the input with the correct intent tag. A hypothesis for why this was the case is that there was more information available for the CNN meaning better accuracy. *Figure 1.5* some of the results.

The third thing I tested was its ability to interpret typos and inputs similar but not quite the same as the hardcoded pattern inputs. For example, I tested words like ***“Utnil enxt teim”,*** which is a typo that is supposed to be Until next time. I also tested ***“Tell me, what is it that you are doing”,*** which is similar but not the same as one of the patterns ***"Tell me what you are doing".*** The interpreter was quite good at recognising typos and matching them to the most appropriate input. *Figure 1.6* some of the results.

*Figure 1.4 Figure 1.5*

A screenshot of a computer program

Description automatically generated with medium confidence A screenshot of a computer program

Description automatically generated with medium confidence

*Figure 1.6*

A picture containing text, screenshot, font

Description automatically generated

## Evaluation of text-interpretation using Colton’s creative Tripod

**Skills**: The programs ability to interpret information is imperfect but still reasonably okay to use. There is still room for improvement, and with some additional work to interpret single-word inputs, which will be necessary for a better result. The chatbot is at least consistent with interpreting inputs with typos, inputs that deviate from the standard patterns, and multi-word sentence inputs. I believe I can improve this by increasing the number of intent tags and patterns the CNN must work with, as there is not much data for the CNN to train with. Ultimately, this could potentially be the main reason why the chatbot behaves like this.

**Appreciation:** With the current prototype, the system randomly chooses a text as a response instead of generating one. Therefore, the system is showing little creativity here at all. However, with the functionality of the text generation, the functionality to generate responses to the user’s input already exists. I can implement this into a future prototype to be able to generate responses based on intents by feeding the RNN transcripts of texts that suit the corresponding intent.

**Imagination:** I believe text interpretation is extremely creative, at least for the functionality it currently performs. The CNN functions as a brain for the program, forcing the computer to be creative in matching user’s inputs to the correct intent. Also, it can interpret data decently well in most aspects; all that is needed is just to be able to creatively output data.

## Testing the text generation

I tested the text generation by making the program print the results at different temperatures. I tested the program with different transcripts alongside their corresponding RNN that was built specifically for the following transcripts: Game of Thrones Season 1, Simpsons Seasons 1 to 5, Shakespeare, and SpongeBob seasons 1 and 2 (Transcripts: Game of Thrones 2020) (The Simpsons episode scripts) (SpongeBob SquarePants transcripts: Season 1) (SpongeBob SquarePants transcripts: Season 1) (Shakespeare dataset 2022).

*Figure 1.7*shows the SpongeBob, whilst *Figure 1.8 Shows the Simpsons.*

Figure 1.7

A picture containing text, screenshot, font

Description automatically generated

*Figure 1.8*

A screenshot of a computer

Description automatically generated with medium confidence

It’s worth knowing that the RNN used for the Simpsons, ‘**textgenerator\_SimpsonV2,**’contains 200 epochs, whereas SpongeBob’s RNN, ‘**textgeneratorV2\_SpongeBob.model,’** contains only 80 epochs. Epochs represent the quantity of time cycles for the neural network. Furthermore, the number of characters used for training the SpongeBob model was 99,000, whereas the Simpsons used 198,000 characters. Typically, with neural networks, more training-data usually means better performances. In my opinion and experience, the Simpson text generation typically generates much clearer text than the SpongeBob, although that is down to your interpretation.

## Evaluation of text-generation using Colton’s creative Tripod

**Skills:** When comparing the styles, the text generation performs well in imitating the style of the original transcript. Unfortunately, a good chunk of the words is often syntactically inaccurate. Moreover, most of the generated text doesn’t seem to make much sense relative to the context. With more data to learn from, the text generator should perform better, as I do believe there was an improvement when comparing text with lesser transcript quantity and RNN. Nevertheless, I still do believe the text-generation was as high quality as my laptop can handle; it took roughly 14 hours for my laptop to train the RNN with the Simpson transcript. In the future, perhaps I can try to create or retrieve even bigger transcripts and train the RNN with even bigger portions of data.

**Laptop specification:**

**GPU**: RTX 2060

**CPU**: AMD Ryzen 7 4800H

**Memory**: 16GB

**Appreciation:** The chatbot performed extremely well in terms of appreciation. The use of an RNN that is trained and outputs information entirely on the transcript shows enough appreciation for the context. You can see this further as the style of text typically changes to match the style of the transcript given.

**Imagination:** The chatbot was creative in its ideas to interpret text in a way to imitate the style of the transcript. It successfully adopted words often used in the transcript, allowing much of the information to at least be humanly readable and interpretable instead of something messy. Even how the text is spaced and formatted is also adopted by the RNN; for example, in *figure 1.*8, the text includes the name of the character who is talking and is formatted neatly.

# Conclusion

To conclude this report, I believe this project is indeed extremely creative. The methods used to interpret information and generate content are creative via the use of neural networks to perform machine learning. The results of the chatbot may not be perfect, but they are still up to an appreciable standard, making the project successful. For future prototypes, I will attempt to provide more data for the interpreter to train on, hopefully harnessing better results. Additionally, I will also try to combine the functionality of the interpreter and the text-generation to create a computationally creative response system for the chatbot. Lastly, I can try using even bigger transcripts and train the RNN with an even bigger portion of characters.

Overall, with these results, I have learned how to build an artificially intelligent text-interpreter system that is often used in modern-day chatbots and a text-generation system.

# References

Wikipedia contributors. Computational creativity [Internet]. Wikipedia, The Free Encyclopedia; 2021 Sep 14, 09:20 UTC [cited 2023 May 1]. Available from: <https://en.wikipedia.org/wiki/Computational_creativity>

OpenAI. Chat [Internet]. San Francisco: OpenAI; [cited 2023 May 1]. Available from: <https://chat.openai.com/>

Jordanous A. What is computational creativity? [Internet]. The Creativity Post; 2012 Dec 19 [cited 2023 May 6]. Available from: <https://www.creativitypost.com/science/what_is_computational_creativity>

Surfactants Network. The role of artificial intelligence in video games [Internet]. Surfactants Network; 2021 May 4 [cited 2023 May 1]. Available from: <https://www.surfactants.net/the-role-of-artificial-intelligence-in-video-games/>

Surfactants Network. The role of artificial intelligence in video games [Internet]. Surfactants Network; 2021 May 4 [cited 2023 May 1]. Available from: <https://www.surfactants.net/the-role-of-artificial-intelligence-in-video-games/>

TensorFlow. Keras documentation: LSTM character-level text generation [Internet]. 2022 [cited 2023 May 1]. Available from: <https://keras.io/examples/generative/lstm_character_level_text_generation/>

The Game of Thrones. Transcripts: Game of Thrones [Internet]. Forever Dreaming; [cited 2023 May 4]. Available from: <https://transcripts.foreverdreaming.org/viewtopic.php?t=7739>

The Springfield! Springfield! Simpsons Archive. The Simpsons episode scripts [Internet]. Springfield! Springfield!; [cited 2023 May 4]. Available from: <https://www.springfieldspringfield.co.uk/episode_scripts.php?tv-show=the-simpsons>

TensorFlow. tf.io.read\_file [Internet]. 2021 [cited 2023 May 6]. Available from: <https://www.tensorflow.org/api_docs/python/tf/io/read_file>

SpongePedia. SpongeBob SquarePants transcripts: Season 1 [Internet]. SpongePedia; [cited 2023 May 4]. Available from: <http://en.spongepedia.org/index.php?title=Episode_Transcripts/Season_1>

SpongePedia. SpongeBob SquarePants transcripts: Season 2 [Internet]. SpongePedia; [cited 2023 May 4]. Available from: <http://en.spongepedia.org/index.php?title=Episode_Transcripts/Season_2>

NeuralNine. TensorFlow 2.0 Complete Course - Python Neural Networks for Beginners Tutorial [Internet]. YouTube; 2019 Apr 9 [cited 2023 April 26]. Available from: <https://www.youtube.com/watch?v=QM5XDc4NQJo&ab_channel=NeuralNine>

NeuralNine. TensorFlow 2.0 Tutorial for Beginners - Machine Learning Basics with Python [Internet]. YouTube; 2019 Apr 29 [cited 2023 April 26]. Available from: <https://www.youtube.com/watch?v=1lwddP0KUEg&ab_channel=NeuralNine>

Chewy. Learn Python and TensorFlow in a fun and practical way - Episode 1 [Internet]. YouTube; 2019 Jul 9 [cited 2023 April 26]. Available from: <https://www.youtube.com/watch?v=OV6k33GH2Vk&ab_channel=Chewy>

Floatbot. The evolution of chatbots: From origin to conversational AI [Internet]. Floatbot; 2022 [cited 2023 May 4]. Available from: <https://floatbot.ai/blog/the-evolution-of-chatbots-from-origin-to-conversational-ai>

TensorFlow. Shakespeare dataset [Internet]. 2022 [cited 2023 May 4]. Available from: <https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt>