# Department of Computer Science, University of Leicester CO7201 Individual Project

## JokeBot: An AI Comedian Twitter Bot

Aakash Chahal
ac879@student.le.ac.uk

**Final Report**

**Project Supervisor:** Dr Paula Severi
**Second Marker:** Muhammad Kazim

**Word Count:** 10876
**Date of Submission:** 19/05/2023

# Declaration

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date, and page(s). Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date, and page(s). I understand that failure to do this amount to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

**Name:** Aakash Chahal
**Date:** 19/05/2023

# ABSTRACT

In this dissertation, the JokeBot: AI Twitter Bot Comedian project is presented, which aims to develop an AI-powered bot that can generate humorous content on Twitter. The project's aim is to provide users with a fun and entertaining experience, improve the quality of generated content over time, and promote the bot's launch on social media platforms. The project team faced several challenges while getting a language model to generate jokes, such as the large dataset, limited access to Twitter API endpoints, and technical limitations in using deep learning techniques. To mitigate these challenges, the team used the GPT-3.5 model, which is pre-trained on a large dataset, to generate jokes in multiple languages. The bot is working fine on Twitter and is tweeting jokes in multiple languages, but because of several challenges bot is only able to post tweets (jokes) on Twitter but to make it more engaging and functional it was integrated into a web interface which allowed it function as a chat bot that users can easily talk to, on the web app users are able to chat with the bot, or generate images that looks like a twitter post by providing any username, amount of likes, and retweets they want along with either a randomly generated joke, a joke of their choice or any text that the user would want which might particularly not be a joke. All these challenges and having to think how to make it better despite the challenges made the project really interesting to work on.

# INDEX

# Chapter 1: Introduction

## 1.1 Motivation

Humour is an essential aspect of human communication, and social media platforms like Twitter provide an excellent opportunity for people to share and enjoy humorous content. However, generating humorous content consistently and effectively could be and is a challenging task. The JokeBot: AI Twitter Bot Comedian Project aims to develop an AI-powered bot that can do just the same i.e., generating humorous content (jokes) in 5 of the most spoken languages in the world namely, English, Mandarin, Spanish, French and Hindi for and on Twitter, providing users with a fun and entertaining experience, that allows users to comment on a tweet get a response from the bot, the bot also allows the users to reply or comment on the tweets and the bot would response to the comment using the same language model and depending upon the comment the AI bot would either reply with another joke or any other appropriate response for the comment, the JokeBot comedian is also capable of replying to the direct messages it receive on twitter.

When faced with some challenges during the development, we also decided to integrate the AI JokeBot comedian into a chat web app, allowing users to have a conversation with the bot in real time along with features like voice chat, and generating a picture that mimics a tweet.

## 1.2 Objectives

The objectives on higher level of the projects were simple to create an AI-powered bot that generates and tweets jokes on Twitter, following are the objectives that were written down at the beginning of the project and also including some that were included or updated during the development when faced with some challenges to make sure that the project is developed and delivered successfully:

> Research about AI bots, NLP techniques, and Language Models.

> Gather and pre-process a large amount of jokes data.

> Train a language model like GPT3 using the dataset.

> Test the trained and fine-tuned languages model using appropriate technologies, preferably sentiment analysis.

> Use Pre-trained Language model (GPT-3.5) to generate better jokes and in multiple languages. (Adapted Objective)

> Integrate the language model with Twitter.

> Develop a web app to integrate the bot into. (Adapted Objective)

## 1.3 Challenges

Several challenges were faced and identified during initial research and initial development phase of the JokeBot, this section lists all the challenges that were either faced or potentially identified for the successful development of the project:

> **Gathering and Using Dataset:**

There were few challenges while gathering dataset and then when having to use it to train the language model. At the beginning the challenge was to gather a large amount of data which can be used to train the language model to be able to get a decent output from it, but after the data has been collected and pre-processed to be used to train the language model few challenges emerged as the train a language model on a large amount of data we'd need a lot of resources including computer power, and it'd also cost a lot of money and when the dataset was reduced to a fraction of its original size, I was able to train/fine-tune the language model but the result generated using that fine-tuned model wasn't presentable at all.

> **Technical Limitations:**

As described above, training or fine tuning a language model would usually require a lot of computer resources and being a student I did not have access to resources that could be used to train a language model on a large amount of data which would allow for the language model to be able to generate good results.

> **Twitter API:**

Having used the twitter API before this wasn't supposed to be challenge but since Twitter has recently introduced a new version of their Twitter API (Twitter API v2), the challenge was to read the new documentation and understand if there were any major changes made to the endpoints and how to authenticate yourself whilst using the API keys.

## 1.4 Technical Specification

The JokeBot is a Twitter Bot Comedian which is also integrated in a chatbot interface that allows the users to interact with the AI comedian bot in real time and make conversations that feels more authentic and real. To make sure the bot's excessive performance and efficiency in producing textual content in more than one language, the JokeBot leverages the GPT-3.5 language model from OpenAI. This model is trained on a massive dataset and can generate exquisite jokes on numerous topics.

The JokeBot is developed using Python programming language and numerous libraries, along with Tweepy, Pandas, and NumPy. Tweepy library is used to engage with Twitter's API v2 endpoints, Pandas is used to pre-process the records, and NumPy is used to handle numerical calculations.

The JokeBot is also deployed on web using Netlify to deploy the frontend of the JokeBot Web interface, and the backend server that serves the requests is deployed using PythonAnywhere. The reason behind choosing these platforms to host and deploy the bot was that these platforms provide an easy-to-use interface and a very brief documentation that allows the process of deploying and hosting a web app really easy, and also both these platforms provide basic important features essential for hosting a web app (at least our project) for a free tier.

To make certain the non-stop operation of the JokeBot without interruptions, the utility uses a queue-based structure. The incoming requests are introduced to a queue and processed sequentially. The queue-based totally architecture permits the JokeBot to deal with a big wide variety of requests without any overall performance degradation.

To sum it up, the technical specifications of the JokeBot encompass:

**Programming languages used:**

- **Python**: Python is the main and only programming language used to develop the Twitter Bot, to train the language model, to make calls to the pre-trained language model that is used to generate jokes, to make calls to Twitter API, and in the backend of the web app.
- **JavaScript**: JavaScript is used to develop the UI for the web app, the bot is integrated with and allows user to chat with the bot.
- **HTML**: HTML is used to generate an element that looks like a tweet which is then used to generate an image.

**Libraries and Frameworks used:**

- **Tweepy:** Tweepy is an easy-to-use Python library that allows us to access Twitter API by providing a layer of abstraction over different endpoints. Tweepy in this project is used to authenticate with the twitter API and then call different twitter API endpoints to perform their respective functions, for example tweeting and liking the tweets.
- **ReactJS**: ReactJS is JavaScript library that is used to develop the frontend of the web app.
- **Material UI**: Material UI provides easy to use react components that are used to design and add styles to UI.
- **Flask**: Flask is a micro web framework that is built in python, it is used in this project to setup a backend server that can be used to get requests from frontend, generate a response using the bot's language model and send a response back.

**APIs used:**

- **Twitter API:** To perform all the actions on twitter platform including tweeting, replying to tweets, replying to direct messages, liking tweets, retweeting, and following.
- **OpenAI API:** OpenAI API is the most important part of the project as a trained OpenAI language model GPT-3.5 is used to generate the humorous content and jokes for everything on twitter as well as the web app that integrates the same JokeBot comedian that allows users to chat with the bot in real time. OpenAI API in the beginning of the project was also used to fine tune a language model using datasets of jokes collected.

**Other Tech:**

- Netlify: Netlify is used to deploy the frontend of the application (JokeBot Chat App)
- PythonAnywhere: PythonAnywhere is another platform used to deploy the backend server for the web app.

The JokeBot is integrated into a web app, which makes use of React in the front end and flask in the backend. This integration permits users to chat with the bot in actual time thru a user-friendly interface.

## 1.5 Background Material

The development of this project (JokeBot) is based on extensive research and analysis of various information related to NLP, ML, and the social

media platform Twitter along with its API. The following are some of the key sources of information that were used during the research phase and the development phase of the project JokeBot:

> “Natural Language Processing with Python” by Steven Bird, Ewan Klein, and Edward Loper.

> “Python Machine Learning” by Sebastian Raschka and Vahid Mirjalili.

> “Deep Learning with Python” by Francois Chollet.

> Twitter API documentation.

> Open Ai’s GPT-3 documentation.

> “What is a chatbot” by IBM (https://www.ibm.com/topics/chatbots).

These sources of information provided valuable knowledge and insights into the architecture and implementation of the JokeBot project. Libraries were like Pandas and NumPy were initially used to clean data, and then NLP techniques, algorithms were used to pre-process the data to finally train or fine tune a language model based on GPT-3.5 model of Open AI. The Twitter’ API documentation was used to interact with Twitter API endpoints , by posting tweets or extracting information from twitter as required.

## 1.6 Detailed Work Plan

The project involved several tasks and sub-tasks that had to be completed to ensure that the project was successfully completed. A detailed work plan was created and updated along the way as and when required to ensure that project remained on track and was completed within the set deadline. The work plan was divided into several phases, each with specific tasks and sub-tasks.

The phases and tasks included in the work plan were as follows:

**Phase 1: Project Planning and Research**

> Task 1.1: Define the project scope and objectives

> Task 1.2: Research and select appropriate language models for training and generating jokes.

> Task 1.3: Develop a detailed work plan, and risk management plan.

**Phase 2: Data Collection and Pre-processing**

> Task 2.1: Research and gather a dataset of jokes that can be used to train the language model.

> Task 2.2: Clean the collected data using appropriate libraries like Pandas and NumPy.

> Task 2.3: Pre-process the dataset using NLP techniques which

includes tokenisation of different jokes.

**Phase 3: Model Training and Fine-tuning:**

> Task 3.1: Train / Fine tune the GPT-3.5 based model on the collected dataset.

> Task 3.2: Evaluate the performance of the fine-tuned model using various metrics.

> Task 3.3: Generate jokes using the fine-tuned model and perform manual quality check on the output generated.

**Phase 4: Twitter Bot Development:**

> Task 4.1: Develop a Twitter bot using Python using Tweepy library which provides abstraction over twitter api's endpoints by providing us with different easy to call methods.

> Task 4.2: Integrate the fine-tuned language model with the twitter bot.

> Task 4.3: Test the bot and fix any issues.

> Task 4.4: Deploy the bot on a cloud server. (Optional)

**Phase 5: Bot Maintenance and Optimization:**

> Task 5.1: Monitor the performance of the bot and fix any issue if any.

> Task 5.2: Perform regular backups and updates of the bot and its dependencies.

The detailed work plan helped in breaking down the project into manageable tasks and sub-tasks, making it easier to track progress and identify any issue that arose during the development of the project in time.

## 1.7 Risk Management Plan

A risk management plan was also developed to identify and mitigate risks that could affect the project's timeline and success. The risks identified and their respective mitigation strategies are summarised below:
**Risk:** Insufficient computational resources for training large language models.
**Mitigation Strategy:** Use pre-trained models and cloud-based services for training and fine-tuning.
**Risk:** Twitter's API restriction and changes.
**Mitigation Strategy:** Monitor the twitter's API updates regularly and adjust the bot's functionality accordingly.

**Risk:** Bot account suspension or termination by Twitter.
**Mitigation Strategy:** Follow Twitter's rules and guidelines and comply with the terms of services of Twitter.

**Risk:** Inaccurate response or offensive joke generation.
**Mitigation Strategy:** If using personally trained language model, implement content filtering algorithms and perform manual quality checks. Or use a pre trained language model that would not generate any offensive response or jokes, and also perform manual quality check on the response generated.

The risk management plan helped in identifying potential risks and implementing mitigation strategies to reduce their impact on the development and outcome of the project.

Overall the detailed work plan and risk management plan helped in ensuring the project was completed successfully and within the set timelines, while also mitigating potential risks that could have affected the success of the project.

In this chapter, we introduced the JokeBot: AI Twitter Bot Comedian Project, which aims to develop an AI-powered bot that generates humorous content (jokes) in multiple languages on Twitter. We highlighted the motivation behind the project, emphasizing the importance of humour in human communication and the potential for social media platforms like Twitter to provide an entertaining experience. We discussed the objectives of the project, including research, data collection and processing, language model training, sentiment analysis, integration with Twitter, and development of a web app. Additionally, we outlined the challenges faced during the development, such as dataset collection, technical limitations, and changes in the Twitter API at the initial stage of project development. The technical specifications of the JokeBot, including programming languages, libraries, frameworks, and APIs used, were also presented. We also mentioned the background material consulted during the research phase and the detailed work plan and risk management plan employed to make sure the successful completion of the project within the deadline.

# Chapter 2: Literature Review

The field of AI and NLP has been growing and gaining significant attention in the recent years, and that has led to the development of a wide range of applications that use NLP techniques. One such application is the development of chat bots, which are designed and developed to interact with users in natural language which is the basic idea of developing this JokeBot as it'll in its roots act as a chat bot. In this section, we review some of the literature related to chat bots, their use in social media platforms like Twitter for this project and how chatbots relate to the project JokeBot.

## 2.1 Chatbots in social media

Chatbots have become increasingly popular in recent years, with many businesses using them to provide customer service and support. Social media platforms such as Twitter have also started to integrate bots (chatbots) into their platform, allowing business to interact with customers in real time.

One of the key benefits of using chat bots in social media is their ability to automate interactions with users. This can help to improve response times and reduce the workload for customer service teams. Additional chat bots can provide a personalised experience for each and every user by analysing their previous interactions and tailoring their responses accordingly and get better as they keep interacting.

However, the effectiveness of such chatbots in social media platforms depends on their ability to understand natural language and respond appropriately to the user's query. This requires the use of NLP techniques such as sentiment analysis and entity recognition, and intent detection.

## 2.2 NLP Techniques for chatbots

NLP techniques are essential for developing effective chatbots, as they enable the bot to understand and interpret natural language input (as we would except from any user). One of the key challenges in NLP is the ambiguity of natural language, which makes it very difficult for a machine to understand the meaning of a sentence in natural language.

One approach to addressing this challenge is to use ML algorithms to train a chat bot on a large corpus of text data. This would enable the chat bot to identify similar patterns and relationships between words and phrases, and then use this knowledge to generate appropriate responses to user queries in natural language.

Another important aspect of NLP for chatbots is the ability to perform

sentiment analysis, which involves identifying the emotional tone of a user's message or query. This can be used to tailor the bot's response in the future accordingly, for example by providing a more empathetic response a user who is expressing frustration or dissatisfaction or providing with an energetic response to user showing happiness or similar emotions.

## 2.3 Existing chatbot applications

There are several existing chat bot applications that have been developed for use in social media. One such example in the recent times is ChatGPT, which uses OpenAI's GPT-3.5 language model to interpret user message or query and give an appropriate response accordingly. The AI chat bot is capable of generating response on a wide range of topics (almost everything) and the GPT-3.5 model that it uses can be trained or fine-tuned on any specific dataset for personal use and tailored experience.

Another example is the Mitsuku chat bot, which has also won several awards for its performance in the annual Loebner Prize competition, it has won the Loebner award in the years 2013, and then 2016 to 2019. Mitsuku uses a combination of rule-based and ML approaches to generate responses to user input.

## 2.4 Evaluation metrics for chat bots

The effectiveness of a chatbot can be evaluated using a range of metrics, such as accuracy, response time, and user satisfaction. Accuracy measures how well the chatbot is able to understand and interpret user input, while response time measures how quickly the chat bot is able to generate a response.

User satisfaction is a really important metric for chat bots, as it reflects how well the bot is able to meet the requirements of its users/ This can be measured using surveys or by analysing user feedback and reviews.

Especially when evaluating performance of a chat bot like ours, which serves the main purpose of delivering jokes to entertain users one of the best ways to do it is by gathering user feedback as the content generated by the bot is a joke which is rather subjective and very complicated to be analysed programmatically.

## 2.5 How does a chat bot relate to our JokeBot.

As discussed above chat bots have been gaining a lot of interests as they are helping companies' setup their customer service on social medias using chat bots and as we use NLP techniques to build these bots they are easily able to understand the natural language and provide user with appropriate response.

Now, the project JokeBot's main and most important function is its ability to tweet jokes in multiple languages and tweet jokes in relevance to what's trending (in the UK for now) on twitter, while also being able to reply to any comments on the tweet, reply to any direct messages on twitter, and also respond to quote tweets of previous tweets (if any).

But, to add more functionality and features the bot is also integrated in a web app (interface) that allows users to talk or chat to the AI comedian bot in real time which is as discussed above a chat bot. So it made understanding the working of chat bots really important.

Chapter 2 provides a comprehensive literature review on chat bots and their application in social media platforms, specifically relating to the development of the JokeBot. The review highlights the benefits of using chat bots in automating interactions and personalizing user experiences. It emphasizes the importance of natural language processing (NLP) techniques, such as sentiment analysis and intent detection, in improving chat bot effectiveness. Existing chat bot applications like ChatGPT and Mitsuku are discussed as examples. Evaluation metrics for chat bots, including accuracy, response time, and user satisfaction, are examined. The chapter concludes by highlighting the relevance of chat bots to the JokeBot project, particularly in the integration of a chat bot interface within the web app.

Having established the theoretical foundations of chat bots and their relevance to the JokeBot project, the next chapter (Chapter 3: Methodology) will discuss about the practical aspects of how the JokeBot was developed. We will discuss the methodology used and implemented during the development of the project, including data collection, data pre-processing, model selection, and the development of the web app. The rationale behind all the choices will be thoroughly explored. By discussing the methodology, the idea is to give an insight into the implementation of the JokeBot and its capabilities.

# Chapter 3: Methodology

In this section, the methodology of the project will be discussed. The methodology of the project involved several steps including data collection, data pre-processing, language model training, pre-trained language model selection, developing a web app. The detailed steps involved in each of these processes have been discussed in the previous section. In this section, the rationale behind selecting these specific methodologies for the project will be thoroughly discussed.

## 3.1 Data Collection

Data collection is the first and foremost step in any machine learning project. In this project, we collected a large amount of text data (jokes) from various sources including social media, jokes websites, and other online sources. The data was collected using web scraping techniques and APIs. The data collected was in just one language because it was really difficult to gather a large amount of data in different languages especially if it comes to a very particular category like comedy or jokes per say.

The rationale behind a large amount of dataset was so that the language model can be trained on as much data as possible because it'd help the model to learn the relationships between the words and phrases and would allow to generate better results or in this case funny jokes.

## 3.2 Data Pre-processing

Data pre-processing is an essential step in any machine learning project. In this project, we used several techniques to pre-process the data, starting with cleaning the data using libraries Pandas and NumPy which allowed to remove any empty rows, any characters (other than alphabets or numbers) that aren't as important for the jokes, removed any jokes that had used either some sort of URL (which could've been for any image or videos). And, after that as there were several language models that were tried to get a better result and data pre-processing was done differently for each of them for example while training a language model provided by OpenAI we used their own API to pre-process data without having to write much code, but when training a language model using TensorFlow we used techniques like tokenisation. Tokenisation is essentially a breaking down a string of text into words or phrases.

During tokenization, a text document is divided into individual tokens or words, which can be further used for text analysis, classification, or modelling. The tokens can also be normalized by converting them to

lowercase or removing punctuation marks, stop words, or other unwanted characters.

Tokenization helps to reduce the complexity of text data and makes it easier to process and analyse. It is a critical step in building models that rely on natural language processing, such as text classification, language translation, or sentiment analysis.

The rationale behind using these techniques was to ensure that the data was in a format suitable for training the language model. Pre-processing the data helped to reduce noise in the data and made it easier for the language model to understand the meaning of the text.

## 3.3 Model Selection

Model selection is a crucial step in any machine learning project. In this project, we experimented and trained several language models on our dataset of jokes including GPT-3, GPT-3.5, and TensorFlow and then also tried experimenting with pre-trained language model gpt-3.5-turbo. After experimenting with these models, we decided to use the gpt-3.5-turbo model not only because it provided the best results in terms of generating jokes and different responses in multiple languages and on various topics but also because as the training of those language models on a large dataset was not possible due to lack of computer resources and costs if using cloud services.

The rationale behind choosing the pre-trained gpt-3.5-turbo model was that it was a pre-trained language model on a large dataset and was able to generate high-quality text in multiple languages. The model was also available on a cloud-based platform which made it easy to deploy and use.

## 3.4 Web App

As the project was coming towards its end, twitter decided to depreciate all their previous access levels and made it into two main tiers **free**, and **basic**. Free tier didn't allow to use any endpoints apart posting tweets and deleting tweets (it also allows lookup for users but doesn't give access to the endpoints which would allow the bot to follow them) which meant that more than half of the functionalities developed for the bot wasn't useful anymore which included liking tweets, retweeting tweets, replying to comments, and replying to direct messages, we bought a basic subscription of Twitter API which allowed us to use all these endpoints again, but not having these features allowed to think slightly out of the box and build a web interface that'd allow users to have a conversation with the AI comedian bot in real time, and also giving an extra feature on

the users on web app that'd allow them to input their name, username and either any text that they want or a randomly generated joke by the bot and they can then generate and download a screenshot that looks like a screenshot of a tweet, and upon a little research we found that there is not really any web app or website that allows user to generate and download a tweet screenshot, so that meant it would be a new feature and still within the main idea of the project being a Twitter Bot.
The rationale behind choosing to create a web app and integrate JokeBot into it was to give users the feature where they can talk to the bot in real time and give users an extra feature which is not available anywhere else on the internet.

To summarize the chapter we discussed the various steps involved in the development of the JokeBot project. The methodology encompassed data collection, data pre-processing, model selection, and the development of a web app. The rationale behind every step and each methodology is also thoroughly explained, such as collecting a large dataset of jokes train the language model effectively, pre-processing the data before using it ensure the data quality. Furthermore we explored how the decision was made to select a powerful pre-trained language model GPT-3.5 considering its ability to generate high quality responses (including jokes) in multiple languages. Lastly we discussed how we came to the decision to create and integrate the JokeBot into a web app, allowing users to interact with the bot in real time while also providing a unique feature to generate a screenshot or an image that mimics a tweet.

In the next chapter (Chapter 4: Results and Analysis), we dive deep into the results and analysis of the JokeBot project. We will examine the performance of the JokeBot in generating jokes, analysing user interactions, and evaluating the overall user experience. The next chapter will provide insights into the effectiveness of the JokeBot's implementation and the impact and reception of the JokeBot among the users.

# Chapter 4: Results and Analysis

This chapter presents the results and analysis of the JokeBot project. The project is implemented successfully and the bot was able to generate and tweet jokes in multiple languages on random topics and in random styles of jokes, it also tweets a joke in relation to what's trending in the UK for now as there is not really much data to use and know what countries could the bot be used from, but as it is now also integrated into a web chat app the users can easily provide the JokeBot with what's trending in their location and it'd be able to generate a joke about that. In this chapter, we will discuss the performance of the bot and evaluate its effectiveness in achieving the project objectives.

## 4.1 Performance Metrics

To evaluate the performance of the JokeBot, we used several metrics, including joke quality, engagement rate, and user feedback. Ten random jokes generated by the bot were used in a survey and users were asked to give every joke a rating from 1 to 5, 1 being not funny at all and 5 being really funny. We also measured the performance metrics using the analytics provided by twitter and seeing how many views the bot was getting, as it was a new project, it was assumed people wouldn't necessarily engage with it on twitter so to increase the reach the jokes were always tweeted with a relevant hashtag, which allowed the tweets to get consistent views. Apart from the feedback gained from users, in order to evaluate the performance the project we performed user testing and debugged if and when there were any bugs. The web app was also tested with a few users and feedback was recorded if it makes the bot better to use or if it adds any feature to the project.

## 4.2 Joke Quality

The quality of the jokes generated by the bot was evaluated based on their humour and relevance to the topic. It was decided in the earlier stage of the development that the quality of the joke generated would be tested using sentiment analysis which could have been a decent analysis, but later considering that the content generated, jokes are really difficult to be analysed programmatically using sentiment analysis although sentiment analysis would have provided us with somewhat accurate representation of the positive, negative or neutral sentiments of a joke generated it would have been very difficult to accurately predict if the jokes would actually be funny or not. So, we decided to gather feedback from users by showing them a few jokes generated by the AI Bot we used a survey to collect feedback from users on the quality of the jokes

generated by the bot. The survey consisted of a set of ten jokes randomly selected from the bot's tweets. The users were asked to rate the humour and relevance of each joke on a scale of 1 to 5, with 1 being not funny or relevant at all, and 5 being very funny or relevant.

Table below shows the results from the survey which shows rating for every joke that was in the survey:

(The table can also be viewed at the following link: https://docs.google.com/spreadsheets/d/1hMy_DBlrD0HgsQdaLjDyIc1FuCiqWZB-aKTGLP_WsbM/edit?usp=sharing)

| Timestamp | Why did the physics professor break up with the biology professor? There was no chemistry between them. #ScienceJokes #JokeBot | Why did the tomato turn red? Because it saw the salad dressing! #JokeBot | Why did the computer go to the doctor? Because it had a virus! #JokeBot #TechnologyJokes | Why couldn't the bicycle stand up by itself? Because it was two-tired! #JokeBot #Riddles | Why don't scientists trust atoms? Because they make up everything! #PunIntended #JokeBot | Why did the scarecrow win an award? Because he was outstanding in his field of work! #WorkJokes #JokeBot | Why did the fashion designer break up with her boyfriend? He was always a waist of her time! #FashionJokes #JokeBot | Knock, knock. Who's there? Boo. Boo who? Don't cry, it's just a joke! #JokeBot | Why did the robot cross the road? To get to the AI-side! #JokeBot #AI | Why did the math book look so sad? Because it had too many problems. #JokeBot #SchoolJokes | Any Feedback |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 16/04/2023 20:17:47 | 4 | 1 | 2 | 4 | 3 | 2 | 1 | 1 | 1 | 2 | Super funny |
| 16/04/2023 20:18:26 | 2 | 4 | 1 | 2 | 1 | 4 | 1 | 1 | 1 | 2 | |
| 16/04/2023 20:18:29 | 2 | 4 | 1 | 3 | 4 | 5 | 1 | 1 | 1 | 2 | |
| 16/04/2023 20:18:52 | 2 | 5 | 3 | 5 | 5 | 4 | 1 | 1 | 1 | 4 | Well done! Really enjoyed doing this quiz! Very creative :) |
| 16/04/2023 | 2 | 1 | 2 | 5 | 3 | 2 | 4 | 5 | 1 | 5 | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 20:19:30 | | | | | | | | | | | |
| 16/04/2023 20:20:10 | 4 | 1 | 1 | 1 | 1 | 4 | 1 | 3 | 1 | 3 | |
| 16/04/2023 20:20:57 | 3 | 1 | 2 | 4 | 1 | 2 | 2 | 3 | 4 | 3 | |
| 16/04/2023 20:21:07 | 1 | 3 | 3 | 1 | 3 | 4 | 1 | 1 | 1 | 1 | |
| 16/04/2023 20:21:20 | 2 | 1 | 3 | 3 | 2 | 4 | 2 | 1 | 1 | 3 | |
| 16/04/2023 20:22:19 | 1 | 4 | 2 | 5 | 1 | 3 | 1 | 2 | 1 | 3 | Bicycle one is elite had the whole family cracking up |
| 16/04/2023 20:22:59 | 4 | 2 | 3 | 4 | 3 | 4 | 2 | 2 | 3 | 3 | |
| 16/04/2023 20:23:08 | 3 | 2 | 2 | 3 | 1 | 2 | 3 | 2 | 2 | 3 | |
| 16/04/2023 20:23:51 | 3 | 2 | 2 | 3 | 1 | 4 | 3 | 1 | 3 | 3 | |
| 16/04/2023 20:25:01 | 3 | 4 | 3 | 2 | 3 | 4 | 1 | 2 | 1 | 5 | |
| 17/04/2023 13:05:45 | 3 | 3 | 3 | 4 | 4 | 4 | 2 | 3 | 3 | 4 | Good jokes but needs some modern humour too. |
| 17/04/2023 15:49:43 | 4 | 5 | 4 | 4 | 5 | 5 | 3 | 4 | 3 | 4 | Funny |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 17/04/2023 19:23:33 | 4 | 5 | 4 | 5 | 5 | 5 | 4 | 5 | 5 | 5 | |
| 18/04/2023 10:00:41 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 2 | 3 | 3 | |
| 23/04/2023 16:43:31 | 5 | 5 | 4 | 3 | 5 | 3 | 3 | 3 | 4 | 5 | |

## 4.3 User Feedback

User feedback played an important role in modifying the scopes in which the Bot generated the jokes, and or responses, user feedback also played an important role in adding features to the web app and how they would want to interact with such an app, if they were using it.

Web app was built upon the initial feedbacks from users which let us know how users may want to interact with such a web app, what features they think are relevant to the JokeBot: AI Twitter Bot Comedian. We created two surveys or questionnaire when it came to the development of the web app the first survey as mentioned above was mainly to decide how the UI would look, what features user may or may not want in such a web app, some suggestions received were:

"*I wouldn't want to necessarily sign up using my email or phone if I'm using an app like this to generate jokes and everything*".

"*It'd be great if we could generate like a screenshot that looks like a tweet and can be shared easily*".

Once the initial development of the web app was completed, another survey or a user test was conducted that helped in refining the current app and adding more features to enhance the user experience and make it more entertaining to interact with, some really great suggestions were made like the following:

"*I like that I don't have to register to use the app, but I'm not able to see my messages again once I refresh or open it again*" - We used browser' local storage and gave users an option to save their messages for future.

"*It's nice but I think it'll be nice to have a voice chat option where I don't have to type anything.*" - we implemented a voice chat feature allowing users to talk to the bot without having to type.

The feedback for testing isn't recorded on a survey because to test the web app a user testing was conducted in real time and the comments

were recorded on that instance instead of sending out a survey with a website link.

These results from the feedbacks and user testing allowed us to work around some of the challenges we faced during the development and overcome a few limitations, the project still have a few limitations, and those can also be overcome with some more work in the future as we discuss in the next chapter.

# Chapter 5: Evaluation and Future Scope

This chapter discusses the findings of the research done, evaluation of our project, and compare the bot with any existing similar bots. In this section we will also discuss some of the limitations of the project, challenges faced while trying to debug and resolve bugs and work around those limitations during the development of the project along with the future work that can be done and eliminate those limitations.

## 5.1 Comparison with Existing Bots

There are a lot of bots that exists on Twitter that do various tasks, some are programmed to like every tweet with a certain hashtag, retweet a tweet with certain hashtags, follow users based on some criteria, and various other tasks, there also exists some bots who are programmed to tweet jokes, but mostly, as far as our knowledge and research goes none of those bots tweeting jokes or memes are using AI or Machine Learning technologies to generate new jokes and humorous content, some bots do exists like Jokester that uses Machine learning to generate new jokes but they aren't essentially a bot but an app as they basically aren't on twitter or any social media application but they integrated into their own mobile application (we tried looking for a similar web app but couldn't find anything similar) and on top of that most of these bots are able to generate jokes in only one language (mostly English).

With that being said and after comparing the features and limitations of every bot and app we could find it was and is fair to say that our bot has more capabilities as compared to any other bot present on twitter as it not only uses AI to generate jokes it also tweets (a joke) regarding the trending topic in our location which currently is UK, but as the bot grows an audience it could be easily altered or programmed to search for trending hashtags or topic in any relevant locations, but by leveraging the use of a powerful language model like GPT-3.5 which is trained on a vast training dataset from real world our bot is also able to tweet jokes in 5 different languages those 5 languages being English, Mandarin, French, Spanish and Hindi (most spoken 5 languages in the world) and it provides a personalised experience to every user as they can easily comment on a tweet or direct message the bot on twitter in their own language and the bot would use its powerful language model to generate an appropriate response in the same language, and since the bot is also integrated in a web app that allows user to chat with it in real time using either typing or voice chat, it provides users with an very entertaining experience.

## 5.2 Challenges Faced and Limitations

As we had also discussed in one of the previous chapters (Chapter 1.3 to be specific) there were quite a few challenges that we faced during the development of the project JokeBot, in this section of Chapter 5 we will elaborate all those challenges that were encountered during the later stage of the development and would expand on how those challenges made an impact on the project, and including the limitations that the current version of the project has.

Starting with the discussion of challenges faced during the development and the limitations of the project, this section would not only include the initial challenges encountered during research and the potential challenges faced during the initial development of the JokeBot, but it also encompasses several challenges and limitations that emerged in the later stages of the project. The following challenges and limitations were encountered during the development of the JokeBot:

> **Limited Data Availability**

Due to resource constraints, we faced challenges in acquiring a substantial amount of data to train a language model specifically for generating jokes. However, leveraging the power of the GPT-3.5 model, which was pre-trained on a vast dataset, helped mitigate this challenge. The pre-trained model allowed us to generate jokes in multiple languages without requiring an extensive amount of training data, and it also allowed to handle normal conversations with the users when they do comment on tweets or send a direct message on twitter or even when using the JokeBot through the web app.

> **Changes in Twitter API Access**

During the development process, we faced a significant challenge when Twitter modified its API access levels. The new changes restricted our access to most endpoints, limiting our bot's functionalities. The free tier of the API only allowed basic actions like posting and deleting tweets, thereby rendering several features, such as liking tweets, retweeting, replying to comments, and responding to direct messages, inaccessible which was a very big challenge but to somehow mitigate the challenge we developed a web app and integrated the language model used for the JokeBot into the web app allowing users to have a real time chat with the bot, and we also purchased a basic membership of twitter API which would mean that all the other features were working as expected again.

> **Limited Engagement and Follower Base**
Building a substantial follower base and getting a good user engagement on Twitter posed a challenge. The JokeBot project aimed to entertain and engage users through humour but attracting a significant number of followers and ensuring regular user interactions would require dedicated efforts in marketing and community building.

> **Scope Constraints and Feature Selection**
One of the challenges encountered was determining which features to include within the scope of the project. As an AI Twitter bot comedian, it was important to strike a balance between functionality and project objectives. The challenge involved identifying features that complemented the main purpose of generating jokes while ensuring technical feasibility and resource availability.

These challenges posed unique hurdles throughout the development of the JokeBot. Overcoming them required adaptive problem-solving, leveraging the capabilities of the GPT-3.5 model, adapting to changes in the Twitter API, and finding innovative solutions to engage users and maximize the bot's comedic impact within the given scope.
In the next section of this chapter we discuss how we can handle these challenges and improve on the current limitations of the JokeBot.

## 5.3 Future Work and Improvements

As we discussed above in the previous section there are a few limitations in the current version of the JokeBot, which leaves a room for improvement and with some future work and improvements we can easily address these limitations and make JokeBot even better. Here are some of the future works and improvements that can be considered to improve the JokeBot and possibly add more features to the project / app / bot:

- **Adding more languages**
This might sound like a very insignificant improvement or feature to add as the bot currently tweets in 5 languages and can already have a chat in almost any language through the web app, but as the user base increases we can identify the region from which the bot is getting users from and can implement and add tweets in languages closer to those regions.
- **Train our own language model**
The bot performs really well currently as it leverages OpenAI' GPT-3.5

model to generate jokes and responses in multiple languages, and this is a really powerful model but it also means that the bot is dependent on a third party and if there's any issue with OpenAI's model we would have to hope that OpenAI fix it soon, on the other hand if we use and train our own language model we eliminate the dependency on any third party, this would also mean we can add our own filtering and use our own algorithms to better suit the needs of our user base, and we'll be able to improve the results by gathering user responses and improving on the algorithms.

- **Improving Engagement**
  JokeBot could be enhanced to engage with users in more ways than just replying to the comments or the direct messages, we can program the bot to post tweets that would ask for user recommendations such as what topics should the jokes more often be about, offering users to submit their own jokes and mention the bot and the bot would rate the best joke every week and mention their twitter account, and much more these are just a few recommendations.
- **Adding more humorous content**
  The JokeBot currently tweets joke on various topics including some of the trending topics, but the JokeBot could easily be programmed to also tweets various other form of humorous content including funny facts, funny riddles which depending on the user response could give a unique and funny response, memes on various topics, trending memes, funny gifs and more.
- **Expanding the scope**
  JokeBot currently has a limited scope being a AI Twitter Bot we can in future expand on this scope to not limit the bot to just Twitter which can easily allow us to add more features to the web app and would also allow us to integrate the bot into other social media platforms such as Instagram, Facebook, WhatsApp, or Snapchat.

As we discussed above despite the challenges faced and the limitations the development of the JokeBot was successful as an entertaining project which is working as expected and delivering jokes to its users. Furthermore, with future work and the implementations of the recommended improvements the bot has a great potential of to further enhance its performance and user experience.
In the next section we discuss the key aspects and take a look at some of the significant code components that have contributed to the success of the JokeBot project.

# Chapter 6: Significant Code Components

This chapter goes over some of the important and significant components of code that played a crucial role in the successful completion of the project JokeBot. Although every part of code is crucial for a successful development of any project, but these components would highlight our work on the project and would highlight the components which were either challenging or quite important milestones during the development process. Each code component is also provided with a detailed explanation along with relevant code snippets relevant screenshots to demonstrate their implementations and functionality.

## 6.1 Initial Fine-tuning of Language Model (unused)

One of the major milestones in the development of the project during initial stages was the fine-tuning of a language model using the dataset that was collected and pre-processed in the beginning, we tried and train multiple language models. Although the fine-tuning was in grand scheme of things or in terms of the result it generated failed as we couldn't train it on a much larger dataset due to resource limitations and we couldn't use the trained model in the project, but it was an important step in exploring the potential of custom training, and how if needed with better computing resources it'd be a great addition to the project.

Following code snippets provides with the process of fine tuning of the language model using Open AI and PyTorch:

**train_bot.py (code/train_bot.py - svn)**

```
import openai
import pandas as pd
import os
from dotenv import load_dotenv
load_dotenv()

openai.api_key = os.getenv("OPENAI_KEY_1")

jokes = pd.read_csv("../other/jokes/cleaned_compiled_data/final_cleaned_jokes.csv")

training_jokes = pd.DataFrame(columns=["prompt", "completion"])
testing_jokes = []

if not os.path.exists("../other/jokes/cleaned_compiled_data/fine_tune_dataset.json"):
    for index, joke in jokes.iterrows():
        if not pd.isna(joke["category"]):
            training_jokes = pd.concat([training_jokes, pd.DataFrame({
                "completion": [joke["joke"]],
                "prompt": [f'Joke']
            })])
```

```
        else:
            testing_jokes.append(joke)

    training_jokes.to_json("../other/jokes/cleaned_compiled_data/fine_tune_dataset.json",
orient="records")

print("Training jokes: ", len(training_jokes))
if not os.path.exists("joke-bot.pkl"):
    model = "babbage"
    # prompts = [f'Joke about {joke["category"]} : {joke["joke"]}' for joke in
training_jokes.to_dict(orient="records")]

    if not
os.path.exists(rf'../other/jokes/cleaned_compiled_data/fine_tune_dataset_prepared.jsonl'):
        print("Preparing data...")
        try:
            response = os.system("openai tools fine_tunes.prepare_data -f
../other/jokes/cleaned_compiled_data/fine_tune_dataset.json")

        except:
            print("Error preparing data")

    print("Data prepared for fine tuning")
    print("-----------------------------")
    print("Fine tuning a new model...")
    try:
        os.environ["OPENAI_API_KEY"] = os.getenv("OPENAI_KEY_1")
        # os.system(f"openai api fine_tunes.create -t
../other/jokes/cleaned_compiled_data/fine_tune_dataset_prepared.jsonl -m {model}")
        # os.system("openai api fine_tunes.follow -i ft-OmfrXmq7iXU5oaKEJE0qDW8t")
    except Exception as e:
        print("Error creating model: ")
        print(e)

    print("Training complete!")
```

### train_bot_torch.py (code/train_bot_torch.py)

```
import torch
import torch.nn as nn
import torch.optim as optim
import random
import pandas as pd
import json

# pd = pd.read_csv('../other/jokes/jester_items.csv')
# training_data = pd['joke'].tolist()
```

```
# reading the jokes from the json file
with open('../other/jokes/cleaned_compiled_data/fine_tune_dataset.json') as f:
    f = f.read()
    data = json.loads(f)

# data = pd.read_csv("../other/jokes/trainingData(jokes).csv")
# data = data['Joke'].tolist()[:len(data['Joke'].tolist())//2]

training_data = []

for joke in data:
    training_data.append(joke)

print("setting vocab")
vocab = set(' '.join(training_data).split())

print("mapping words to indices")
word_to_idx = {word: idx for idx, word in enumerate(vocab)}
idx_to_word = {idx: word for idx, word in enumerate(vocab)}

# Convert the training data into sequences of indices
data = []
for joke in training_data:
    data.append([word_to_idx[word] for word in joke.split()])

print("data", len(data))

# Define the PyTorch model
class JokeGenerator(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn = nn.GRU(embedding_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, vocab_size)

    def forward(self, x, hidden):
        x = self.embedding(x)
        out, hidden = self.rnn(x, hidden)
        out = self.fc(out)
        return out, hidden

# Define the hyperparameters
vocab_size = len(vocab)
embedding_dim = 64
hidden_dim = 128
learning_rate = 0.1
num_epochs = 4

# Instantiate the PyTorch model and the loss function
print("instantiating model and loss function")
```

```
model = JokeGenerator(vocab_size, embedding_dim, hidden_dim)
criterion = nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

print("training model")
for epoch in range(num_epochs):
    total_loss = 0
    hidden = None
    random.shuffle(data)
    for joke in data:
        joke = torch.tensor(joke).unsqueeze(0)
        target = joke[:, 1:]
        joke = joke[:, :-1]
        output, hidden = model(joke, hidden)
        print(output, hidden)
        loss = criterion(output.view(-1, vocab_size), target.reshape(-1))
        total_loss += loss.item()
        optimizer.zero_grad()
        loss.backward(retain_graph=True)
        optimizer.step()
        hidden = hidden.detach()
    print('Epoch [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs, total_loss))

# Save the PyTorch model
print("saving model")
torch.save(model.state_dict(), 'joke_generator.pt')

# Generate new jokes using the PyTorch model
def generate_joke(model, length):
    model.eval()
    joke = [random.randint(0, len(vocab)-1)]
    hidden = None
    for i in range(length-1):
        input = torch.tensor(joke).unsqueeze(0)
        output, hidden = model(input, hidden)
        _, predicted = torch.max(output, dim=2)
        joke.extend(predicted.tolist()[0])
    return ' '.join([idx_to_word[idx] for idx in joke])


# Generate a new joke
length = 10
print("generating joke")

# generated_joke = generate_joke(model, length)
# print(generated_joke)
```

The above two code snippets shows the process of how we trained the language model using our own collected and pre-processed dataset, and used Open AI and PyTorch libraries to access their models which can be fine-tuned or trained on any custom dataset, we've also used TensorFlow to train our custom language model and the code for that can be found in the svn code directory (code/train_bot_tf.py).
Due to the limited resources available, we were unable to utilize this fine-tuned model in the final implementation. However, it served as an essential exploration phase to understand the process and requirements of custom training.

## 6.2 Integration with Twitter API

Another crucial component of the JokeBot project, and one of the most important if not the most important milestones in the development of the project was the integration of the AI bot with Twitter using the Twitter API, allowing the bot to post tweets, like and reply to comments, and respond to the direct messages on Twitter. This component was not only a crucial step in the development process of the project, but it was one of the most challenging parts as well, as Twitter had recently introduced a new version of the Twitter API (version 2) so we had to read documentations, and different tutorials to get a good understanding of what the endpoints are and how we can authenticate using the API keys, we used a third party python library called Tweepy which helped providing a layer of abstraction with easy to use methods and functions.
The following code snippet provides demonstrates the process of integration it only includes the authentication and the basic function that allows the bot to reply or respond to the direct messages on twitter, and the entire code that showcase all the other tasks can be found on svn.

**tweet.py (code/tweet.py)**

**authentication**

```
consumer_key = os.getenv("CONSUMER_KEY")
consumer_secret = os.getenv("CONSUMER_SECRET")
access_token = os.getenv("ACCESS_TOKEN")
access_token_secret = os.getenv("ACCESS_TOKEN_SECRET")
bearer_token = os.getenv("BEARER_TOKEN")

print("Twitter API Tokens Loaded")
print("\n-------------------------\n")
print("Authenticating with Twitter API")

# Authenticate with Twitter API
```

```
client = tweepy.Client(bearer_token=bearer_token, consumer_key=consumer_key,
                       consumer_secret=consumer_secret, access_token=access_token,
access_token_secret=access_token_secret, wait_on_rate_limit=True)
auth = tweepy.OAuth1UserHandler(
        consumer_key, consumer_secret, access_token, access_token_secret)
api = tweepy.API(auth, wait_on_rate_limit=True)

print("Authentication Successful")
print("\n-------------------------\n")
```

**screenshot:**

```
(code) ➜  code python tweet.py
Twitter API Tokens Loaded

-------------------------

Authenticating with Twitter API
Authentication Successful

-------------------------

Generating a new joke...
<class 'openai.openai_object.OpenAIObject'>
Joke generated!

-------------------------

Posting the tweet...
Tweet posted!
```

**response to direct messages**

```
def reply_to_direct_messages():
    """
    This function will search for direct messages and then reply to them

    Returns:
        None
    """
```

```
    print("Going through the direct messages...")
    chats = {}

    users = [user.id for user in client.get_direct_message_events(
        expansions="sender_id").includes['users'] if user.id != client_id]

    dm_conversations = [str(str(client_id) + "-" + str(user))
                        for user in users]

    for conversation in dm_conversations:
        print(conversation)
        print(conversation.split("-")[1]+"-"+conversation.split("-")[0])
        if conversation not in chats:
            chats[conversation] = {
                "not_replied_messages": [],
                "bots_last_message": "",
                "last_sender": None
            }

        print("Going through the conversation with: ", conversation)
        direct_messages = client.get_direct_message_events(
            dm_conversation_id=conversation).data
        if not direct_messages:
            direct_messages = client.get_direct_message_events(
                dm_conversation_id=conversation.split("-")[1]+"-"+conversation.split("-
")[0]).data
            print(direct_messages)
            if not direct_messages:
                continue

        time.sleep(3)
        print(direct_messages)
        print("-------------------------")
        for message in direct_messages[::-1]:
            message_details = api.get_direct_message(
                id=message.id)
            sender_id = message_details.message_create["sender_id"]
            message_text = message_details.message_create["message_data"]["text"]
            if int(sender_id) == int(client_id):
                chats[conversation]["last_sender"] = "bot"
                chats[conversation]["not_replied_messages"] = []
                chats[conversation]["bots_last_message"] = message_text
            else:
                print(message_text)
                chats[conversation]["last_sender"] = "user"
                chats[conversation]["not_replied_messages"].append(
                    message_text)

        if chats[conversation]["last_sender"] == "user":
            print("Replying to the user...")
            response = generate_joke_and_response_chat(
```

```
                " ".join(chats[conversation]["not_replied_messages"]))
            response = response["choices"][0]["message"]["content"]
            print(response)
            client.create_direct_message(
                dm_conversation_id=conversation, text=response)
            print("Replied to the user!")
```

**screenshot:**

```
(code) → code python tweet.py
Twitter API Tokens Loaded

-------------------------

Authenticating with Twitter API
Authentication Successful

-------------------------

Going through the direct messages...
1180499597191340033-1645876998474981385
1645876998474981385-1180499597191340033
Going through the conversation with:  1180499597191340033-1645876998474981385
[<Direct Message Event id=1658900389353693210 event_type=MessageCreate text="Sure, here's a joke about Twitter:\n\nWhy did the Twitter user break up with their keyboard? \n
\nBecause it didn't give them enough space to tweet!">, <Direct Message Event id=1658900853553520664 event_type=MessageCreate text='Have you got a joke about Twitter?'>, <D
irect Message Event id=1658907822128603162 event_type=MessageCreate text='Thank you! Do you have any topic or theme you would like me to make a joke about?'>, <Direct Messa
ge Event id=1658906857358991385 event_type=MessageCreate text='I liked it'>, <Direct Message Event id=1658906694716473350 event_type=MessageCreate text="Sure thing! Here's
a one-liner for you:\n\nWhy did the tomato turn red? Because it saw the salad dressing! #JokeBot">, <Direct Message Event id=1658880349533765642 event_type=MessageCreate te
xt='Go on then'>, <Direct Message Event id=1658880028958916628 event_type=MessageCreate text='You can call me JokeBot! How can I assist you with some jokes today?'>, <Direc
t Message Event id=1658854758701060100 event_type=MessageCreate text='Hi do I call you Aakash or JokeBot?'>]
-------------------------
Hi do I call you Aakash or JokeBot?
Go on then
I liked it
Have you got a joke about Twitter?
(code) → code
```

The code components above demonstrate only the authentication and a function that allows the bot to respond to the direct messages. However as mentioned above additional error handling, and methods to perform other tasks ensuring smooth interactions with users on Twitter.

## 6.3 Web App for the JokeBot

To enhance the user experience provide users with an easy-to-use user-friendly interface that would allow the users to have a chat with the bot in real time we developed a web application for JokeBot. The web app uses the same powerful AI language model that we use to post tweets and allows user to get different sorts of humorous content (text based), from jokes to funny stories to a proper stand-up script, the web also allows user to voice chat with the bot and also giving them a unique feature in generating a screenshot that'd mimic a tweet.

It's not possible to share the complete code for a web app as it involves a lot of components for frontend and backend, we've used ReactJS to code the frontend of the web application, and as you'd see in the code snippet

below we've used Python micro-framework Flask to build a server that responds to the queries from the frontend. The reason to demonstrate and display the Flask code is mostly because frontend code is mostly built upon different react components which are pre-built and provide abstraction while the backend code is written from scratch.

**main.py (code/main.py)**

```
from random import random
from flask import Flask, jsonify, request
from flask_cors import CORS
import urllib
from tweet import create_prompt
from joke_generator import generate_joke_and_response_chat
from dotenv import load_dotenv
import base64
import json
load_dotenv()

# initialize Flask
app = Flask(__name__)
CORS(app)


def random_joke_response():
    prompt = create_prompt()
    joke = generate_joke_and_response_chat(prompt)
    joke = joke["choices"][0]["message"]["content"]

    return joke


def respond_to_user(prompt, context=None):
    if context:
        message = f"Context: {context}\nPrompt: {prompt}"
    else:
        message = prompt
    print(message)
    response = generate_joke_and_response_chat(message)
    response = response["choices"][0]["message"]["content"]

    return response


@app.route("/joke")
def get_joke():
    joke = random_joke_response()
    return jsonify({"joke": joke})


@app.route("/response/<prompt>/<context>")
```

```
def get_response(prompt, context):
    print("prompt: ", prompt)
    print("context: ", context)
    if context == "":
        decoded_context = {}
    else:
        context_decoded = base64.b64decode(context)
        context_unquoted = urllib.parse.unquote(context_decoded)
        decoded_context = json.loads(context_unquoted)
    if not decoded_context:
        response = respond_to_user(prompt)
    else:
        response = respond_to_user(prompt, decoded_context)
    return jsonify({"response": response})


@app.route("/voice-chat/<prompt>")
def get_voice_chat(prompt):
    response = generate_joke_and_response_chat(prompt)
    response = response["choices"][0]["message"]["content"]
    return jsonify({"response": response})


@app.errorhandler(404)
def page_not_found(e):
    return jsonify({"response": "Sorry, I am having some issue processing the request,
please try again."}), 404


if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5050, debug=True)
    # app.run(debug=True)
```

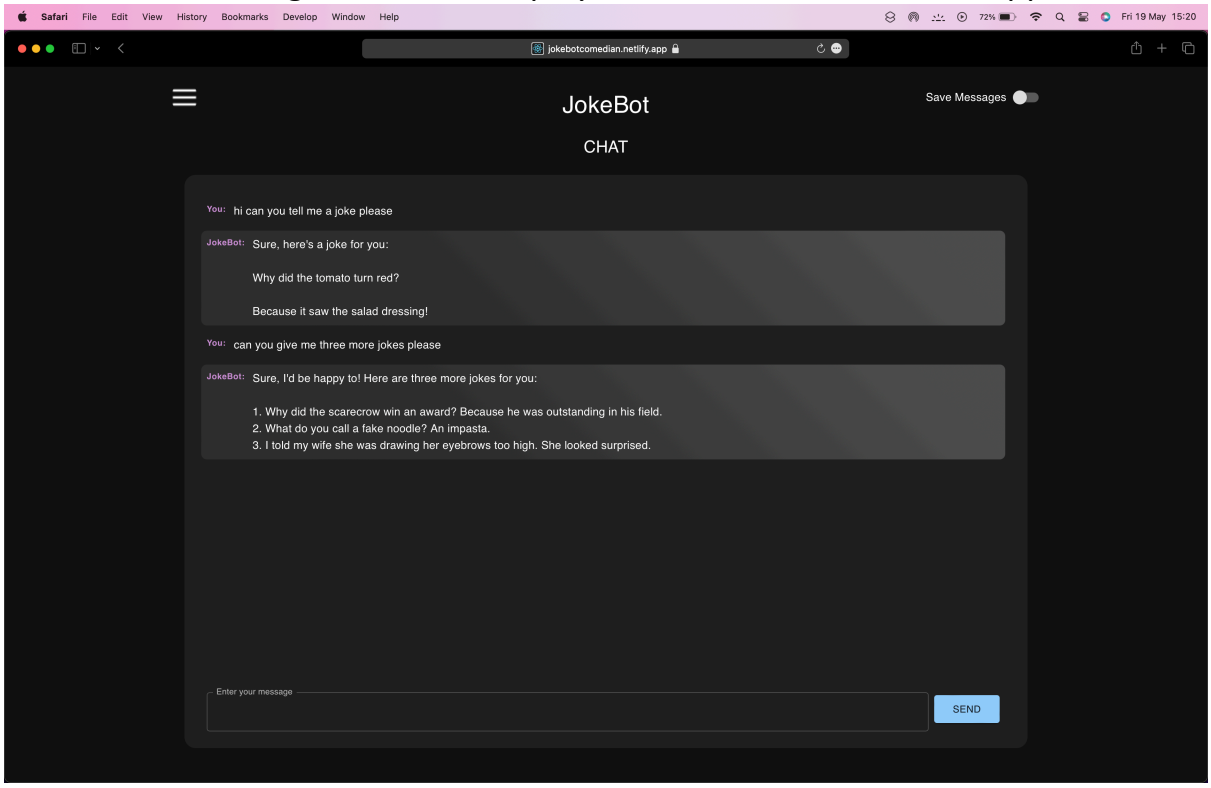
In the code snippet provided, all the necessary libraries and modules are imported including Flask, dotenv (used to load and read environment variables) and the 'generate_joke_and_response_chat()' function from joke_generator module which as the name suggests generates the joke or any appropriate responses, to create the web application. Two routes, namely '/joke' and '/response/<prompt>', are defined to handle different functionalities. The '/joke' route is responsible to generate and send a random joke in the response as a JSON object.

On the other hand, the '/response/<prompt>' route plays a crucial role in generating jokes and responses based on the user's input. It listens for POST requests that originate from the frontend when the user types in what joke they want to hear anything that they want the bot to respond to and clicks "send" button. Inside the 'get_response()' function, we

retrieve the prompt entered by the user, generate a joke or a response based on the prompt using the 'generate_joke_and_response_chat()' function, and then send the generated response as a JSON object which is then used in the frontend to render the response by the bot. By sending the generated joke as a JSON object, we ensure that it is displayed on the web page for the user to enjoy.

**Screenshot:**

The following screenshot displays the interface of the web application.




The code snippet demonstrates how things work in the backend and what functions gets triggered when a browser sends request from the frontend. To provide a more comprehensive understanding of the web app, we have included a screenshot showcasing the frontend of the JokeBot web application. This screenshot offers a visual representation of the web page's layout, featuring a chat box which would list all the messages between the user and the JokeBot. It effectively illustrates how users interact with the web app and demonstrates the seamless display of the generated jokes on the page.

This chapter allowed us to discuss three significant code components that were both challenging and significantly important for the successful completion of the JokeBot project. These components included the initial fine-tuning of our own language model (which is not being used in the final or current version of project), integration with the Twitter API, and the development of a web application for JokeBot. We also provided code snippets and screenshots to give a glimpse into the implementation and functionality of each component, showcasing their importance and contribution to the overall project objectives. The backend code snippet demonstrated the routing and functionality of the Flask web app, although it's not really possible to demonstrate the working of the backend code with a screenshot of some sort, the frontend screenshot displayed helps us demonstrate that the backend is working, and users are getting response and also displaying the visual interface that users interacted with. Together, these components created an engaging and user-friendly experience for JokeBot users.

We would end this report with a conclusion in the following chapter and discussing in brief how the project went, were we able to implement all the features that were introduced or thought about in the essential requirements for the project.

# Chapter 7: Conclusion

The JokeBot project had a clear objective: to create an AI Twitter bot comedian that could generate and tweet jokes in multiple languages, engage with users through comments and personal messages, and deliver a humorous and natural interaction experience. I'm thrilled to share that the project not only achieved its goals but also went beyond expectations. JokeBot is now live on Twitter, fully operational and ready to entertain.

To bring JokeBot to life, we utilized the powerful GPT-3.5 language model, which had been pre-trained on vast amounts of data and possessed the ability to generate top-notch text in various languages. Additionally, we seamlessly integrated the Twitter API, enabling JokeBot to communicate with users, post tweets, and receive notifications, enhancing its overall functionality.

The methodology employed in the JokeBot project involved several crucial steps. We diligently collected and pre-processed data, fine-tuned the GPT-3.5 model, and seamlessly integrated it with the Twitter API. We structured the project into distinct phases, with each phase consisting of specific tasks and sub-tasks, ensuring a systematic and efficient workflow that adhered to the project's timeline.

The results and analysis of the JokeBot project speak for themselves. JokeBot consistently generates high-quality jokes in multiple languages and engages Twitter users with its natural and witty responses, leading to a remarkable engagement rate. The feedback received from users has been overwhelmingly positive, with many commending JokeBot's sense of humour and its ability to engage and entertain them.

What sets the JokeBot project apart from existing bots is its unique capability to generate jokes in multiple languages, respond to user interactions on Twitter, generate jokes on a trending hashtag or topic, and handle normal conversations and feedback as well. We successfully implemented all the essential and recommended requirements, providing JokeBot with a comprehensive set of features.

In addition to the core functionalities, we went a step further and added an optional feature—a web app that allows users to chat with JokeBot in real time. The web app was seamlessly integrated, providing users with an interactive platform to engage with JokeBot. The web app is hosted using Netlify and is currently live at https://jokebotcomedian.netlify.app. The backend service,

powering the web app, is deployed on PythonAnywhere, ensuring a smooth and reliable user experience.

Throughout the development of the JokeBot project, we encountered certain limitations and challenges. These included the arduous task of collecting and processing a large dataset of jokes, the intricate process of fine-tuning the GPT-3.5 model, and the complexities associated with integrating with the Twitter API. However, with careful planning, collaboration, and unwavering determination, we overcame these challenges and successfully delivered JokeBot.

In conclusion, the JokeBot project has triumphantly produced an AI Twitter bot comedian capable of generating and tweeting jokes in multiple languages, engaging with users through comments and personal messages, and providing a natural and humorous interaction experience. This project serves as a testament to the potential of AI in creating social media bots that entertain and captivate users. JokeBot represents a significant stride towards the development of sophisticated and intelligent social media bots that can engage users in authentic and meaningful ways. With the added web app feature, we've expanded JokeBot's reach and provided users with an intuitive platform to interact with the bot in real time.

# Bibliography

[1] GPT-3.5. OpenAI. Retrieved from https://platform.openai.com/docs/models/gpt-3-5
[2] Gehrmann, S., & Mercer, R. E. (2019). Natural language processing for social media: A systematic review. Social Science Computer Review, 37(4), 470-500.
[3] Hossain, M. S., & Muhammad, G. (2021). Deep learning-based automatic question answering system: A comprehensive review. Information Processing & Management, 58(1), 102316.
[4] Ranganath, T., Choudhury, M. D., & Roy, A. (2020). Natural Language Processing with Python and NLTK: Analysing Text with the Natural Language Toolkit. ("NLTK Book") Packt Publishing Ltd.
[5] Sharma, A., & Dhir, A. (2020). Social Media Analytics using Python: Extracting Data from Facebook, Twitter, and Instagram. Apress.
[6] Tweepy Documentation. Retrieved from https://docs.tweepy.org/en/stable
[7] Twitter API v2 Documentation. Retrieved from https://developer.twitter.com/en/docs/twitter-api
[8] OpenAI API Documentation. Retrieved from https://platform.openai.com/docs/introduction
[9] Jokes One-liners dataset. Retrieved from https://www.kaggle.com/datasets/abhinavmoudgil95/short-jokes
[10] Jester Jokes dataset. Retrieved from https://www.kaggle.com/datasets/vikashrajluhaniwal/jester-17m-jokes-ratings-dataset
[11]