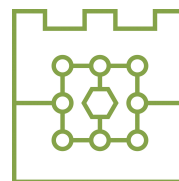




**Politechnika Krakowska
im. Tadeusza Kościuszki**

Wydział Informatyki i Telekomunikacji



David Burchakov

Numer albumu: 139685

**Aplikacja do rozmów z wykrywaniem sarkazmu i
sentymentu rozmówcy.**

**Creating chatbots with sentiment and sarcasm
detection.**

**Praca inżynierska
na kierunku Informatyka**

Praca wykonana pod kierunkiem:
Doktora Filozofii *Radosława Kycia*

Kraków, Styczeń 2025

Aplikacja do rozmów z wykrywaniem sarkazmu i sentymentu rozmówcy.

David Burchakov

Abstrakt

Głównym celem niniejszej pracy jest zbadanie i zastosowanie różnych metod oraz technik analizy języka naturalnego, koncentrując się na opracowaniu chatbota z funkcjami analizy sentymentu i wykrywania sarkazmu.

Praca skupia się na trzech typach chatbotów: FAQ bot, Flow-based bot oraz Open-ended bot, z których każdy został zaprojektowany do określonych zastosowań. Analiza sentymentu i wykrywanie sarkazmu zostały wdrożone w celu polepszenia doświadczenia użytkownika. Ocena wydajności została przeprowadzona za pomocą praktycznych przypadków testowych.

Creating chatbots with sentiment and sarcasm detection

David Burchakov

Abstract

The primary objective of this thesis is to explore and apply various methods and techniques in Natural Language Analysis, focusing on developing a chatbot with sentiment analysis and sarcasm detection capabilities.

The study focuses on three chatbot types: FAQ bot, flow-based bot, and open-ended bot, each designed for specific use cases. Sentiment analysis and sarcasm detection were implemented to improve user experience. Performance evaluation was conducted through practical test cases.

Acknowledgments

The topic of this thesis came to my mind thanks to the inspirations I had during my academic journey. This thesis was made possible thanks to Professor ***Luís Paulo Gonçalves dos Reis*** from *Faculdade de Engenharia da Universidade do Porto*, who first introduced me to Artificial Intelligence, Machine Learning and Data analysis. I had the opportunity to take his Master's course as a part of Erasmus Exchange programme. I had to take the Master's course instead of the Bachelor's, as it was the only course available in English, me not being a Portuguese speaker. Thanks to the ***European Union*** that finance exchange programmes that highly changed me as a person and enriched my academic experience.

And of course I want to thank ***Professor Radosław Antoni Kycia*** from *Politechnika Krakowska* for teaching me an introduction to Natural Language Processing as a part of Bachelor's curriculum. This is a topic that is not often included in most Computer Science undergraduate courses, and his lectures sparked my interest in the field.

Finally, I would like to mention well-known figures like ***Lex Fridman***, who inspire and educate the public about Artificial Intelligence with their passion and enthusiasm.

Contents

Introduction.....	8
1.1. Motivation.....	8
1.2. Purpose of the Work.....	9
1.3. Scope of the Work.....	9
1.4. Methodology of the Work.....	9
1.5. Limitations.....	9
Theoretical Background	10
2.1.1. Working with data.....	10
2.1.2. Labeled Data.....	10
2.1.3. One-hot Encoding.....	11
2.1.4. Visualization.....	11
2.1.5. Text Data.....	12
2.1.6. Handling capitalizations.....	13
2.1.7. Stop words.....	14
2.1.8. Tokenization.....	14
2.1.9. Stemming.....	14
2.1.10 Lemmatization.....	14
2.2.1. Sentiment Analysis.....	15
2.2.2. VADER.....	15
2.3.1. Traditional Feature Engineering Models.....	15
2.3.2. Bag of Words.....	16
2.3.3. Bag of N-Grams Model.....	16
2.3.4. TF-IDF.....	16
2.4.1. Advanced Feature Engineering Models: word-embedding models	16
2.4.2. Word2Vec.....	17
2.4.3. GloVe.....	17
Software Architecture.....	21
3.1. Types of Chatbots.....	21
3.2. Specifications.....	21
3.3. System Design.....	23
3.3.1. System Architecture.....	23
3.3.2. Interaction Flow	25
Software Implementation.....	26

4.1. Programming language and libraries.....	26
4.2. Data and Data Processing.....	26
4.3. Chatbot Implementation.....	30
4.3.1 Rule-based Bot.....	31
4.3.2. Flow-based Bot.....	33
4.3.3. Open-ended Bot	35
Testing.....	36
5.1. Manual tests.....	36
Conclusions.....	43
6.1. Evaluation.....	43
6.2. Possible Improvements.....	44
6.3. Discussion.....	44
List of Figures	45
Code Listings.....	46
Bibliography.....	47

Chapter 1

Introduction

Chatbots or originally Chatterbots, first created in the 1960s, are largely used today as human assistants. Initially, these simple systems were mostly linked to a set of responses, usually hard-coded questions and responses (Vajjala et al., 2020).

However, when enhancing chats with NLP technologies and methodologies we can achieve astounding products, that can act as general, multi-purpose assistants. Advanced chats are capable of performing sentiment analysis of human written (natural) language, including sarcasm detection. When considering complex open-ended chatbot, we should note that the whole process goes through multiple stages, each stage depending on the previous. It is hard to assert which part is more important. Data scraping and pre-processing are as important as ML model parametrization.

1.1. Motivation

Advancements in Large Language Models for natural language processing are driving rapid progress in chatbot development. Experts predict that these advancements will bring revolutionary changes to the labour market, with many manual and repetitive tasks being replaced by virtual assistants across various business functions. As these technologies become more accessible and sophisticated, companies are increasingly exploring the potential of virtual assistants to automate routine tasks, enhance efficiency, and improve customer service. (Skuridin and Wynn, 2024)

In recent years, we have witnessed unprecedented growth in NLP-based chatbot technologies. These systems are no longer limited to simple responses but can now engage in complex, context-aware conversations, detect emotions, and generate human-like responses. As chatbots continue to evolve, they are becoming an integral part of industries ranging from healthcare and education to e-commerce and customer support.

Understanding how NLP-based chatbots are constructed is critical to grasping the future trajectory of NLP advancements. By studying how these systems are designed and how they process language, we gain insights into the broader capabilities of NLP. This understanding not only helps in building better chatbots but also highlights the direction in which NLP technologies are heading, opening doors to more innovative applications in the future.

1.2. Purpose of the Work

The purpose of this work is to explore the development of NLP-based chatbot that are able to maintain conversations. This includes designing a chatbot capable of understanding user sentiment, detecting subtle expressions like sarcasm, and responding in a natural manner.

1.3. Scope of the Work

The scope of this work includes devising three types of chatbots: FAQ-based bot, flow-based bot, and open-ended bot. Chatbots are grouped based on how they work and where are they used.

1.4. Methodology of the Work

The work follows several key phases. Initially, a theoretical introduction to main NLP methodologies is provided. The next step involves data pre-processing to clean, normalize, and prepare the text for training. Sentiment analysis and sarcasm detection is performed. Subsequently, Machine learning model is fine-tuned. Finally, the system is tested using manual testing.

1.5. Limitations

Hardware limitations

Due to limited hardware resources, it was not feasible to create an NLP model entirely from scratch for the open-ended chatbot problem. For that reason the whole process of creating an NLP chatbot is discussed in the Theoretical part, supported by practical code examples. The open-ended chatbot is created by fine-tuning an existing model with a dataset that is established to be “sarcastic”.

Testing limitations

Due to the nature of chatbots it is hard to test the software programmatically. Hence, the testing methodology involves manual testing of the results, including a comparison of 3 types of developed chatbots.

Chapter 2

Theoretical Background

In this chapter theoretical methodologies and techniques for creating a chatbot will be explored and discussed. Given limitations mentioned in section 1.5, it is not feasible to develop an advance ML model from scratch in an academic setting, however, this chapter aims to provide theoretical basis necessary to understand key principles and complexities in chatbot development. This chapter encompasses insights in data processing (2.1), provides an introduction to sentiment analysis (2.2), and elaborates on Feature Engineering Models (2.3, 2.4).

2.1.1. Working with data

Development of any ML model requires provision of proper, meaningful data. Data types can be classified by the problem it solves, thus, labeled data is employed to solve a binary classification problem in a supervised learning model; unlabeled data is used to tackle a clustering problem with an unsupervised learning model; text data is utilized to provide a chatbot solution with a NLP model (Sarkar, Bali and Sharma, 2018).

Although processing of each type of data varies, some steps are universal. An example of labeled data processing and visualization will be examined. Later, these techniques will be applied for a text data processing solution.

2.1.2. Labeled Data

A good example of using labeled data is a binary classification problem—where the output is yes/no and labeled datasets are used to train the model—the preprocessing workflow consists of:

- Converting string data into integer data (2.1.3),
- Creating a heatmap to discard attributes with little impact on the result and prioritize attributes with a higher impact on the result. (Figure 1)
- Performing data visualization to identify outliers, erroneous entries (Figure 2, Figure 3),

An example of data visualization performed on REGENSBURG PEDIATRIC APPENDICITIS DATASET¹ will be demonstrated.

¹ [HTTPS://ZENODO.ORG/RECORDS/7711412](https://zenodo.org/records/7711412) (Accessed: January 28, 2025)

2.1.3. One-hot Encoding

Most Machine Learning algorithms expect numerical input. One-hot encoding is a brilliant technique for converting text data into a numerical representation. According to the study “One-Hot Encoding and Two-Hot Encoding: An Introduction - Samuels, J. I. (2024)”, One-hot encoding represents each distinct category as a binary vector, where all elements are set to zero except for the one corresponding to the category of interest - which is set to one. For instance, given a categorical variable "Colour" with classes Red, Green, and Blue, their encoded form would be:

- Red: [1, 0, 0]
- Green: [0, 1, 0]
- Blue: [0, 0, 1]

This encoding makes sure that each category is uniquely represented (Samuels, 2024).

2.1.4. Visualization

Data Visualization alleviate the analysis of large data. It makes it easier to spot erroneous elements, patterns, and inconsistencies for subsequent analysis and cleaning. Seaborn and Pyplot libraries provide great solutions to visualize the data, such as Heatmap (Figure 1), Count Plot (Figure 2), and Scatterplot (Figure 3).

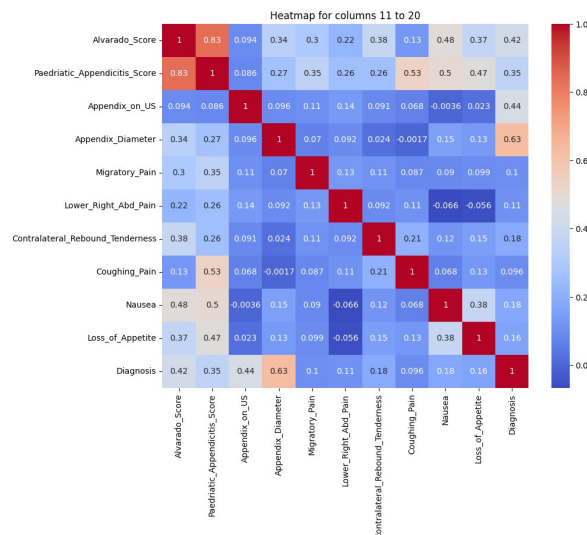


Figure 1: Heatmap (source: own work based on seaborn, pyplot documentation; APPENDICITIS DATASET).

This dataset, obtained from a study of pediatric patients with abdominal pain at Children’s Hospital St. Hedwig, includes symptoms and labeled for diagnosis of appendicitis, with ethical approval from the University of Regensburg.

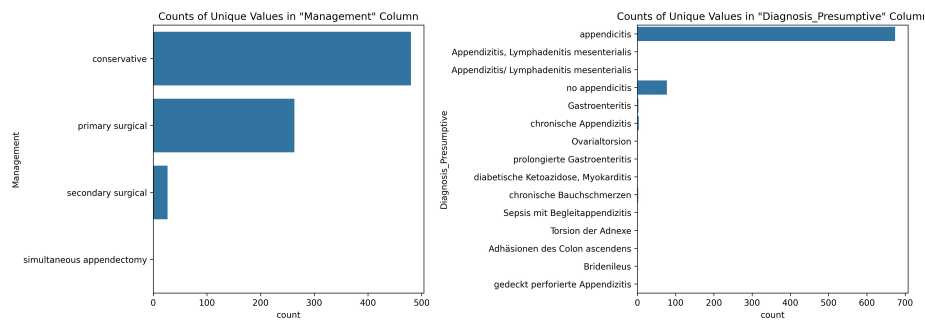


Figure 2: Count Plot (source: own work based on seaborn, pyplot documentation; APPENDICITIS DATASET)

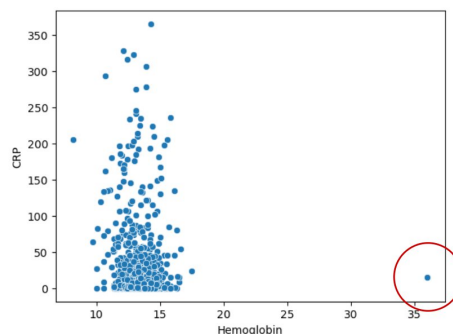


Figure 3: Scatterplot demonstrating an outlier (source: own work based on seaborn, pyplot; APPENDICITIS DATASET).

2.1.5. Text Data

Dealing with text data requires a significantly more scrupulous approach. Textual data scraped from the internet contains plenty of important information, including HTML tags, punctuation and stop-words. The relevance of such elements is undermined in the context of creating language models.

In the example below², it is apparent that the processed text continues to carry its semantic meaning, regardless of the absence of grammar and punctuation:

Input:

```
<div class="RES9jf xWMiCc pb3iw BJ0kwe"><span data-dobid="hdw">The Ministry of Truth, which
concerned itself with news, entertainment, education, and the fine arts. <br> The Ministry of Peace,
which concerned itself with war. <br> The Ministry of Love, which maintained law and
order.</span></div>
```

Expected output:

```
['ministry', 'truth', 'concerned', 'news', 'entertainment', 'education', 'fine', 'art', 'ministry', 'peace',
'concerned', 'war', 'ministry', 'love', 'maintained', 'law', 'order']
```

² The text for the example is taken from George Orwell's "1984".

In order to achieve clean, relevant data for an ML model, the text is thoroughly cleaned using such techniques as: HTML stripping, normalizing accented characters, expanding contractions, removing special characters, handling capitalizations, removing stop-words, tokenization, stemming or lemmatization, etc. These steps are easily achieved leveraging python's libraries such as regex, langdetect, unicodedata, contractions, spacy, nltk, etc. (Sarkar, 2019).

In this thesis, the most important steps in terms of NLP and NLU are examined.

2.1.6. Handling capitalizations

In English language words with identical spelling may have difference semantic meaning depending on capitalization. To overcome this obstacle, Named Entity Recognition (NER) is introduced in NLP.

Named entities refer to world specific objects. Recognizing named entities is vital for maintaining the semantics of the text. Named entities belong to predefined categories, for example:

- ❖ **PERSON**: People, including fictional (e.g., *Martin Eden*).
- ❖ **NORP**: Nationalities or religious groups (e.g., *Polish*, *Roman*).
- ❖ **ORG**: Companies, agencies, institutions, etc (e.g., *CNH*, *UNAIDS*).
- ❖ **GPE**: Countries, cities, or states (e.g., *Warsaw*, *Serbia*).
- ❖ **DATE**: Absolute or relative dates (e.g., *July 18, 1963*).
- ❖ **FAC**: Buildings, airports, highways, bridges, etc. (e.g., *Twin Towers*).
- ❖ **PRODUCT**: Objects, vehicles, foods, etc.
- ❖ **EVENT**: Named wars, battles, events, etc. (e.g *War of the Triple Alliance*)
- ❖ **QUANTITY**: Measurements, as of distance or weight.
- ❖ **ORDINAL**: "first", "second", "thirds", "forth", "fifths", etc.

(From Sarkar, D. 2019, Text Analytics with Python: A Practitioner's Guide to Natural Language Processing. Chapter 8)

“ Named Entity recognition, also called entity chunking/extraction, is a popular technique used in information extraction to identify and segment named entities and classify or categorize them under various predefined classes. ” (Sarkar, 2019).

NER allows to identify named entities based on their context. For example:

- arnold schwarzenegger → Arnold Schwarzenegger (a person).
- Polish (nationality) vs. polish (verb), as in polished shoes.
- Roman (related to Ancient Rome) vs. Roman (a name).

- Apple (company) vs. apple (a fruit).

2.1.7. Stop words

Stop words are linguistic elements that don't carry strong semantic meaning, such as articles (e.g., A, An, The), pronouns (e.g., He/Him, She/Her), conjunctions (e.g., and, but, or). As was spotlighted in section 2.1.5, removing these elements does not hinder from understanding the meaning of the text.

“ Stop-words are common words that don't add any value to the categorization; they are words such as a, able, either, else, ever, and so on. ” (Khan and Das, 2018).

2.1.8. Tokenization

Tokenization breaks down sentences into smaller parts, thus, more easily attributing meaning. “ Tokenization is one of the core steps in text pre-processing and normalization. Each text document has several components like paragraphs, sentences, and words that together make up the document. ” (Sarkar, Bali and Sharma, 2018). Casual_tokenize method from NLTK library can be leveraged to tokenize text effectively. The example below demonstrates natural text and target tokens³:

Oceania was at war with Eurasia and in alliance with Eastasia.

['Oceania', 'was', 'at', 'war', 'with', 'Eurasia', 'and', 'in', 'alliance', 'with', 'Eastasia', '.']

2.1.9. Stemming

Stemming is a technique used in NLP that reduce words to their base form. (Lovins, 1968). The rationale is to represent all different variations of a word by a single form. For example, both "car" and "cars" would be reduced to "car." This is done by applying a set of predefined rules that remove suffixes. For instance, if a word ends in "-es," the rule would be to remove the "-es" to obtain the base form (Vajjala et al., 2020).

2.1.10 Lemmatization

Apart from stemming, lemmatization is a more advanced alternative that reduces words to their base, *dictionary* form, which is called “lemma” (Di Nunzio and Vezzani, 2018). For example, the words "running", "ran", and "runs" all have the same lemma, which is "run". “ ‘Open’ and ‘opening’ are different words with the same lemma, which means they should

³ Text in the example is borrowed from George Orwell's "1984".

be treated the same. Lemmatization helps make that happen. Lemmatization also ensures that plurals are treated the same as singulars (“store” and “stores”). “ (Freed, 2021).

In general, lemmatization yields a more standardized form of a word, but is more computationally expensive than stemming. The choice between the two methods is for the programmer to ascertain.

2.2.1. Sentiment Analysis

Sentiment (or *connotation*, used interchangeably) relates to the emotional meaning of the words. In natural languages, many synonyms express the same meaning, but with different connotation, e.g., *famous* and *infamous* – both words mean “widely known”, where the “famous” carries a positive connotation and “infamous” – negative. “Sentiment analysis is the process of taking a text document and trying to determine the opinion and polarity being represented by that document. Polarity in the reference of a text document can mean the emotion, e.g., positive, negative, or neutral being represented by the data.” (Sarkar, Bali and Sharma, 2018).

There are numerous statistical methods (or “*traditional*”) approaches, as well as advanced deep learning techniques that aim to catch the sentiment of given texts. Each method comes with its own advantages and limitations (Sarkar, 2019). In this chapter, various statistical and machine learning NLP methods will be explored.

2.2.2. VADER

Vader is a sentiment analysis tool that is designed for social media texts analysis. It is a sentiment analysis tool that provides a sentiment score for a given text. (Hutto and Gilbert, 2014)

Vader uses a combination of sentiment qualitative and quantitative methods to analyse and measure sentiment intensity in text. It assigns sentiment polarity to lexical features (e.g words), by labeling them as positive (e.g., Love, nice, good, great) or negative (e.g., Hurt, ugly, sad, bad, worse) (Hutto and Gilbert, 2014) .

2.3.1. Traditional Feature Engineering Models

Traditional feature engineering methods are part of a family of models known as the Bag of Words model. This approach includes techniques such as term frequencies, N-grams, etc. (Sarkar, 2019).

2.3.2. Bag of Words

Rationale behind Bag of Words technique is to represent text as a collection of words, disregarding the order and context. “Bag of words (BoW) is a classical text representation technique that has been used commonly in NLP, especially in text classification problems.” (Vajjala et al., 2020). BoW allows to find out which group the text belongs to. Unordered noncontextual representation makes this model simplistic, but introduces significant limitations (Sarkar, 2019).

2.3.3. Bag of N-Grams Model

Taking into account that BoW model does not consider the order of words, improvements can be undertaken by using N-Grams, usually Bi-grams. N-grams model is an extension of BoW that uses features based on consecutive word pairs. (Sarkar, 2019)

For example, sentence “*Down with Big Brother*” generates bi-grams: “*Down with*”, “*with Big*”, “*Big Brother*”.

2.3.4. TF-IDF

“There are some potential problems that might arise with the Bag of Words model when it is used on large corpora.” (Sarkar, 2019). Frequently appearing terms can diminish seldom appearing but more semantically important ones. “TF-IDF stands for term frequency-inverse document frequency. It’s a combination of two metrics, term frequency (TF) and inverse document frequency (IDF).” (Sarkar, 2019)

2.4.1. Advanced Feature Engineering Models: word-embedding models

Traditional Feature Engineering Models, like Bag of Words (2.3.2), TF-IDF (2.3.4), and N-grams (2.3.3), are useful but have limitations. They delinquent to capture word sequence, and, most importantly, context.

To overcome these limitations, Advanced Feature Engineering Models are introduced. These models predict words based on their context by analysing neighbouring words, looking at word sequences – *embeddings*.

“These models are based on the principle of distributional hypothesis in the field of distributional semantics, which tells us that words that occur and are used in the same context are semantically similar to one another and have similar meanings (“*a word is characterized by the company it keeps*”)” (Sarkar, 2019).

2.4.2. Word2Vec

“ Because the Word2Vec family of models is unsupervised, we can just give it a corpus without additional labels or information and it can construct dense word embeddings from the corpus.” (Sarkar, 2019).

Both Word2Vec and GloVe represent words in vector space. They rely on the contextual understanding of words, but there are differences in their approaches. Word2Vec focuses on predicting words based on their neighboring words (context), leveraging Continuous Bag of Words Model (Mikolov et al., 2013) , while GloVe is based on word co-occurrence matrix within a corpus (Pennington, Socher and Manning, 2014). Thus, the methods are similar, yet there are differences in the specifics of the algorithms that bring about minor variations in the results (Sarkar, 2019).

2.4.3. GloVe

The GloVe method analyses co-occurrence and relationships between words across the entire corpus.

For example, it considers how frequently words like "ice" and "steam" co-occur with words like "solid" or "gas.". The probe word "solid" is expected to co-occur more often with "ice" than with "steam." (Figure 4). If the probe word is related to both or none, such as "water" and "fashion," the expected occurrence probability ratio will be close to 1 (Pennington, Socher and Manning, 2014).

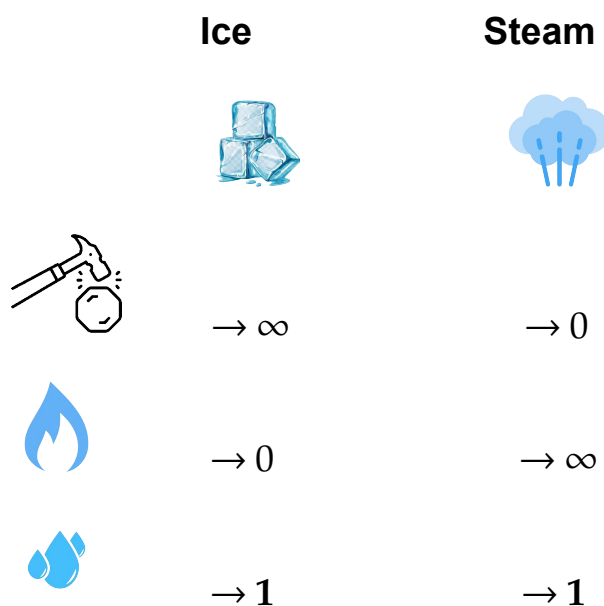


Figure 4: Representation of the GloVe model's co-occurrence analysis (source: own work based on Pennington, Socher and Manning, 2014)

As demonstrated on figure 5, “The relationship of these words can be examined by studying the ratio of their co-occurrence probabilities with various probe words.” (Pennington, Socher and Manning, 2014).

Probability and Ratio	$k = \text{solid}$	$k = \text{gas}$	$k = \text{water}$	$k = \text{fashion}$
$P(k \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k \text{ice})/P(k \text{steam})$	8.9	8.5×10^{-2}	1.36	0.96

Figure 5: Co-occurrence probabilities for the target words (source: Pennington, Socher and Manning, 2014)

Ice and steam with selected context words from a 6-billion-token corpus reveal patterns. The noise from common words like water and fashion cancels out when looking at ratios. High ratios ($\gg 1$) highlight properties specific to ice, while low ratios ($\ll 1$) highlight properties specific to steam (Pennington, Socher and Manning, 2014).

For the purpose of research, GloVe-based embeddings are leveraged. These embeddings are implemented using spaCy library, which supports GloVe embeddings. Alternatively, pre-trained word vectors can be obtained from Stanford NLP’s WEBSITE⁴ and loaded using Gensim or spaCy. EN_VECTORS_ WEB_LG MODEL⁵ from spaCy is used, which consists of 300-dimensional word vector dense embeddings trained on the COMMON CRAWL⁶ dataset with GloVe (Vajjala et al., 2020).

Following the NLP workflow described in *Text Analytics with Python: A Practitioner’s Guide to Natural Language Processing*, Sarkar, D. (2019) it was made possible to train a GloVe model based on open-source dataset from Kaggle⁷. The result of training is presented on Figure 12, showcasing model’s “understanding” of word semantic similarities between words, based on their context. The following list of probe words was chosen:

terms_of_interest = ['movie', 'film', 'good', 'bad', 'hate', 'love', 'actor', 'opinion', 'think', 'see'].

⁴ [HTTPS://NLP.STANFORD.EDU/PROJECTS/GLOVE/](https://nlp.stanford.edu/projects/glove/) (Accessed: January 28, 2025)

⁵ [HTTPS://SPACY.IO/MODELS/EN#EN_VECTORS_WEB_LG](https://spacy.io/models/en#en_vectors_web_lg) (Accessed: January 28, 2025)

⁶ [HTTP://COMMONCRAWL.ORG/](http://commoncrawl.org/) (Accessed: January 28, 2025)

⁷ [HTTPS://WWW.KAGGLE.COM/DATASETS/STEFANOLEONE992/ROTTEN-TOMATOES-MOVIES-AND-CRITIC-REVIEWS-DATASET](https://www.kaggle.com/datasets/stefanoleone992/rotten-tomatoes-movies-and-critic-reviews-dataset) (Accessed: January 28, 2025)

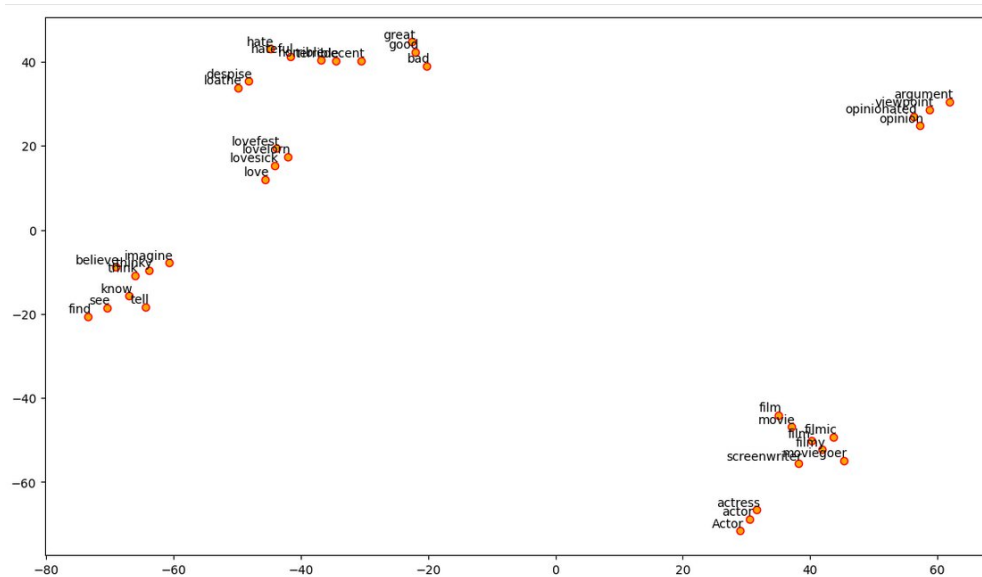


Figure 6: visualized vector representation of words, based on their context (source: own work)

It can be observed on Figure 6 that the model correctly identifies similar words, which are often synonymous, i.e.,:

- "Hate" - "Loathe," "Despise"
- “film” - “movie”
- “actor” - “actress”
- “see” - “find”
- “opinion” - “argument”
- “*good*” - “great” BUT “*bad*”

Although words “good” and “bad” are complete opposite, they are categorized as similar because they are often used in the same context.

Figure 7 represents the results of a similar workflow performed on the same dataset with Word2Vec method. As noted at the end of section 2.4.2, both Word2Vec and GloVe models, that were chosen for analysis, produce similar results with subtle variations; having slightly different specifics of the algorithms.

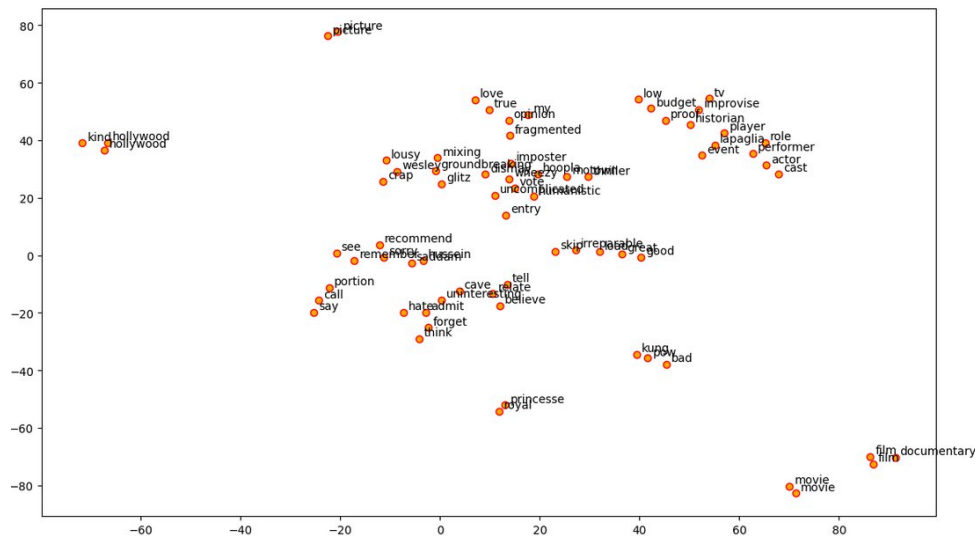


Figure 7: Visualization of Word2Vec vectors (source: own work based on Text Analytics with Python, Sarkar, 2019)
 “Princesse” is close to “royal”. “call” is similar to “say”

Addendum

Having explored, studied, and discussed the basic NLP methodologies for text processing, semantic analysis, and even having performed a small practical experiment on word-embedding models, we have gained the necessary prerequisites for creating a NLP-powered chatbot. In the following chapters the chatbot system will be designed, implemented, and tested. The Open-ended chatbot solution will leverage an even more advanced feature engineering model, called multi-layer transformer architecture (Zhang et al., 2019). As noted in section 1.5 – Limitations: Hardware Limitations, it was not feasible to create a heavy Language Model from scratch, but a pre-trained solution is used instead. Hence, this thesis does not focus on the details of DIALOGPT design; Nonetheless, an introduction to Advanced Feature Engineering Models (2.4.1) provides insights into how Large Language Models are designed and how they work.

Chapter 3

Software Architecture

This chapter will talk about software architecture and establish the software design. Three types of chatbots are examined: Rule-based Chatbot, Flow-based Chatbot and Open-ended Chatbot. Section 3.1 elaborates on the purpose of each type of chatbots, which is important to comprehend in order to define system and software requirements. Section 3.2 introduces functional and non-functional requirements based on the chatbots' intended usage. Section 3.3 provides a high-level overview of the system architecture, showing how the components of the program interact.

3.1. Types of Chatbots

“ Chatbots can be grouped based on how they work and where they are used (Figure 8). One way to classify them is by how they interact with users. We can distinguish Rule-based, Flow-based and Open-ended bots. “ (Freed, 2021).

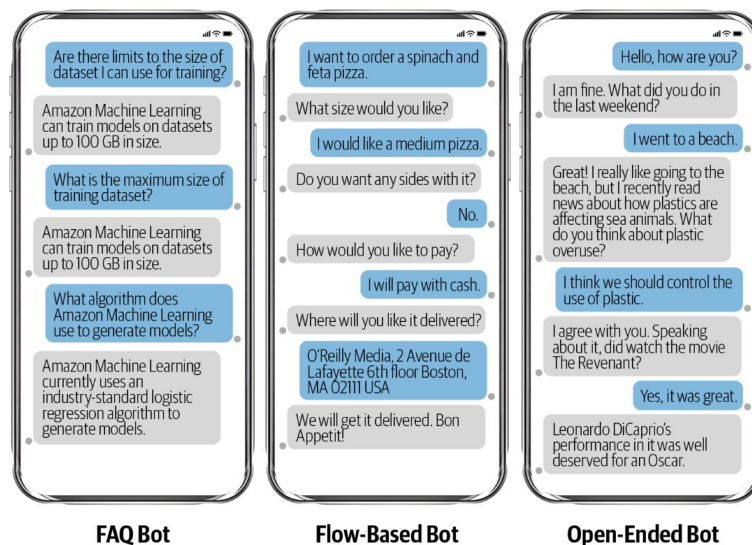


Figure 8: Types of chatbots based on how they are used. (Source: Vajjala et al., 2020, p.201)

3.2. Specifications

In this section, functional and non-functional requirements for each chatbot type are established and justified to make sure that the solutions meet user needs. Non-functional requirements are alike and to be applied to all chatbot types.

FAQ bot

Rule-based bots provide predefined, fixed answers depending on the user's question. Each user input is processed separately, therefore answers do not depend on previous interaction flow. If two similar questions are asked, the bot returns the same response.

Functional requirements:

- Ability to retrieve predefined answers based on user input.
- Responses match user queries to the exactly answer.
- Responses match user queries to the most possibly closest answer.
- If no similar answer is found, a corresponding message is displayed.
- The UI is self-explanatory and all UI elements are intuitive.
- Bot integrates sarcastic sentiment.

Flow-based bot

Flow-based bots are more advanced than rule-based bots, since they incorporate a predefined conversation flow. These bots navigate the user to a prior defined conclusion, step-by-step, with limited conversation paths. To complete a task, a set of questions is asked, and a limited user input is available.

Functional requirements:

- Bot controls a predictable conversation flow.
- Bot encompasses step-by-step interactions based on predefined conversation paths.
- Bot handles branching scenarios; return to previous a step is possible.
- Bot provides clear navigation for the user.
- If no predefined answer is found, a corresponding message is displayed.

Open-ended bot

Open-ended bots are the pinnacle of modern technological advancements, providing a combination of great traditional software engineering solutions and advanced AI engineering solutions. This kind of chatbots is able to carry out a sentiment analysis and converse about different topics without following a specific conversation flow (Freed, 2021).

Functional requirements:

- Relevant response is generated based on user input context.
- Bot is able to “understand” sentiment.
- Bot integrates NLP and NLU elements to return sarcastic responses.

- Chatbot is able process any user input, regardless of the context.

Non-functional Requirements.

The following Non-Functional Requirements are to be applied to all chatbot types:

- The bot provides responses within reasonable time, ensuring minimum wait time for the end-user.
- The system has a user-friendly, self-explanatory, easily understandable UI.
- The system is compatible with web platforms on PC and mobile.
- The system deals with cyber security, user interactions with the server are protected, ensuring data privacy.
- Anonymous access is available, ensuring no sensitive user data is stolen from the server side.
- The project should include proper documentation and comments, ensuring easy maintenance for defects and future features.
- The bot should incorporate sarcastic elements, whether utilizing NLP techniques or having predefined sarcastic responses in the database.

3.3. System Design

This sections talks about the design approach for developing the system. The system design makes sure that all functional and non-functional requirements are met.

3.3.1. System Architecture

The system architecture, depicted on Figure 9, is organised into three main layers, ascertaining smooth operability.

1. Frontend Layer

Handles UI. It acts as a entry point of interactions between the user and the chatbot system.

Components:

- Web interface: Built using streamlit library, ensures responsiveness and multiple device compatibility.
- Input processing module: accepts user inputs and sends them to the Backend.
- Presentation layer: Displays the history of user inputs and chatbot responses.

The Frontend Layer is responsible for:

- Passing user input to the Backend layer.
- Displaying chatbot responses and the history of the conversation.
- Allows access for web platforms on PC and mobile devices.

2. Backend Layer

Handles the main logic of the chatbot system. Namely: processing user inputs, managing data, matching queries, generating responses.

Components:

- NLP Module: Processes raw user input, performs sentiment analysis.
- Dialogue Manager: controls conversation state and history.
- Response Generator: Produces responses using predefined templates (for FAQ and Flow-based solutions) or generative model for Open-ended solution.
- Middleware: Manages corresponding APIs that connect the Frontend Layer with the Data Layer.

The Backend Layer is responsible for:

- Processing raw user input with NLP techniques.
- Generating relevant responses.
- Managing communication between the Frontend Layer and the Data Layer.

3. Data Layer

Stores the knowledge base, datasets, and conversation history.

Components:

- Knowledge Base: Stores the database, specifically FAQs and predefined rules.
- Training Data Module: Contains the conversation history for fine-tuning the model and future improvements and maintenance.
- Conversation Logs Module: Keeps the conversation history for analysis, future improvements and maintenance.

The Data Layer is responsible for:

- Providing access to knowledge base for middleware, subsequently, Frontend Layer via corresponding APIs.
- Ensures data integrity, security and confidentiality.

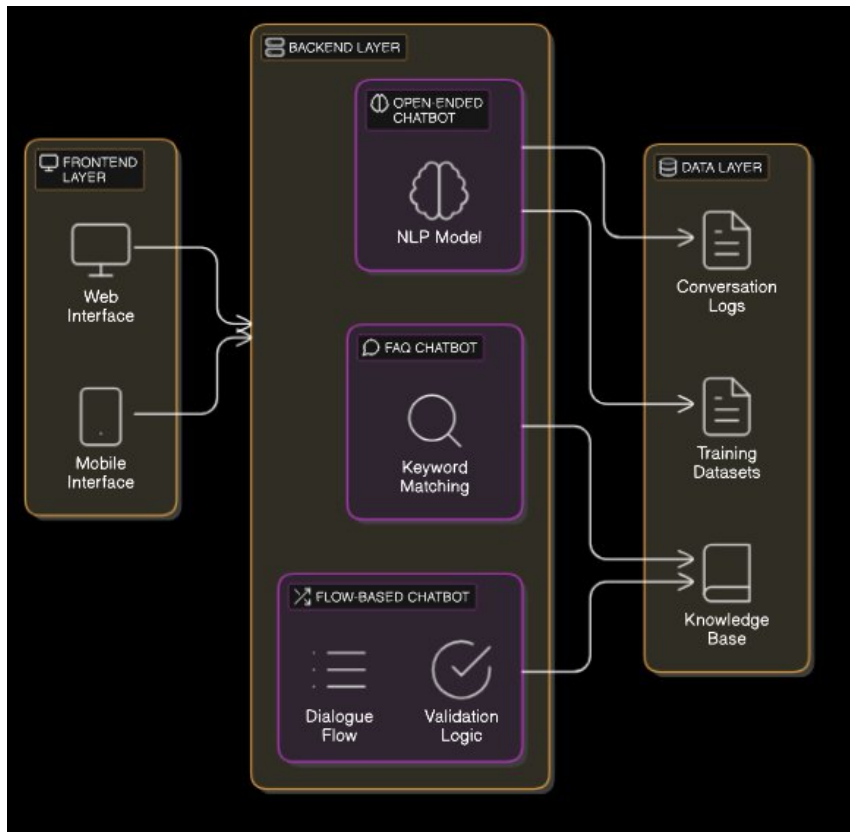


Figure 9: Architecture diagram representing System Architecture (source: own work)

3.3.2. Interaction Flow

Interaction Flow Diagram on Figure 10 depicts:

1. Frontend Interaction: The user sends a query via UI.
2. Backend Interaction: The input is sent to the Backend, where Natural Language Processing and Dialogue Management take place, by corresponding NLP and Data Managers Modules.
3. Response Generation: The Backend generates/retrieves a response, fetches additional information from the Data Layer via APIs, and sends it back to the Frontend.
4. Data Logging: The conversation is logged in the UI and the history of the conversation is saved for future analysis, improvement and maintenance.

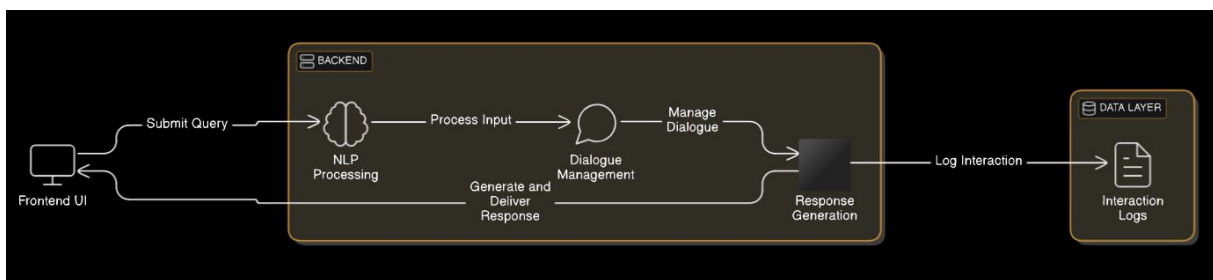


Figure 10: Interaction Flow Diagram (source: own work)

Chapter 4

Software Implementation

In this chapter practical implementation will be discussed and demonstrated, accompanied by explanations of the technologies used.

4.1. Programming language and libraries

Given the nature of the thesis is closely related to Machine Learning, Python is highly suitable for constructing chatbots. Python provide great frameworks for Data Analysis and Visualization (pandas, pyplot, seaborn), Machine Learning (torch) and Natural Language Processing (nltk) that were broadly used in the project. Streamlit library was used to create a light-weight, user-friendly user interface. The model (distilgpt2) for open-ended chat is trained using Hugging Face Transformers library.

Figure 11 depicts the System Architecture overview with mentioned libraries.

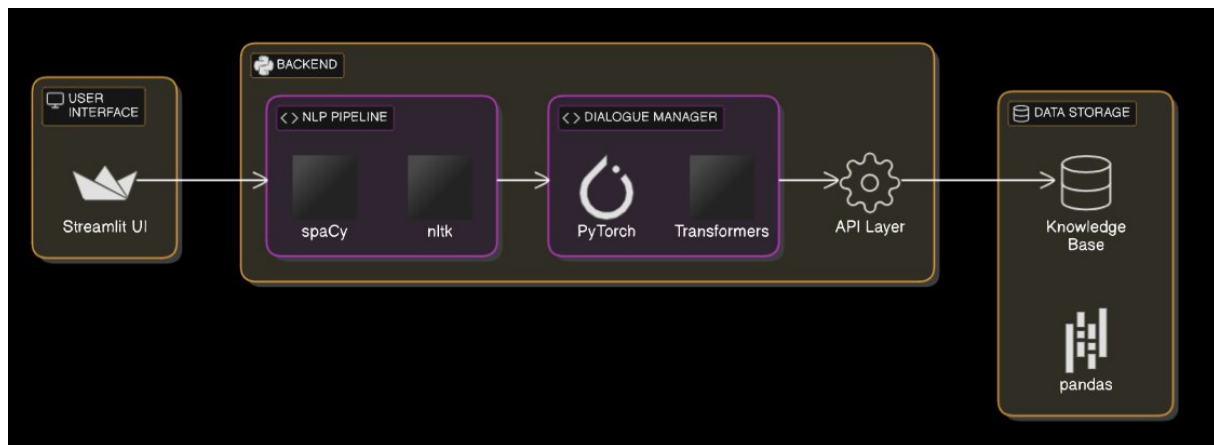


Figure 11: Architecture Overview with Libraries (source: own work)

4.2. Data and Data Processing

For devising an Open-ended bot solution, open source JOKE ⁸ dataset is used. Datasets that include funny or sometimes even aggressive items are more likely to contain sarcastic entries. The dataset comprises of more than 100 000 Question-Answer pairs. However, the dataset, unfortunately, also includes duplicates, empty entries, and most importantly, entries that are clearly not sarcastic. We will use pre-processing techniques that were mentioned in the

⁸ <https://www.kaggle.com/datasets/bfinan/jokes-question-and-answer> (Accessed: January 28, 2025)

theoretical part of the Thesis to clean the dataset and focus on sarcastic Question-Answer pairs concretely.

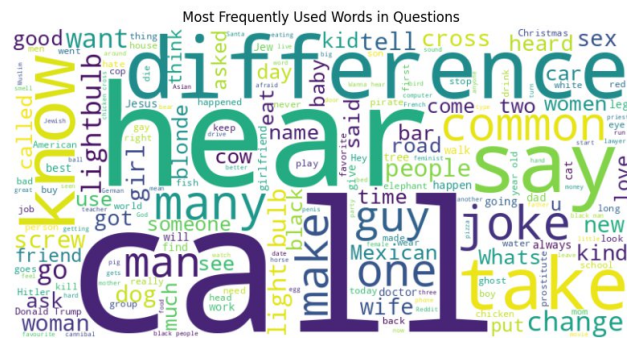


Figure 12: Most Frequently Used Words in Questions from Joke dataset. (source: own work based on wordcloud documentation)

From Figure 12 and Figure 13 we can already see that some of the most common words are spicy, moreover - aggressive. It is worth continuing the analysis by visualizing the text data.

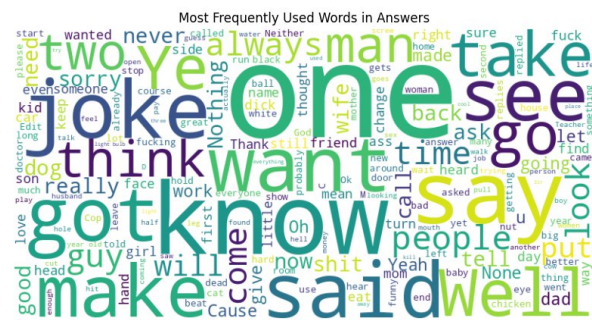


Figure 13: Most Frequently Used Words in Answers from Joke dataset (source: own work based on wordcloud documentation)

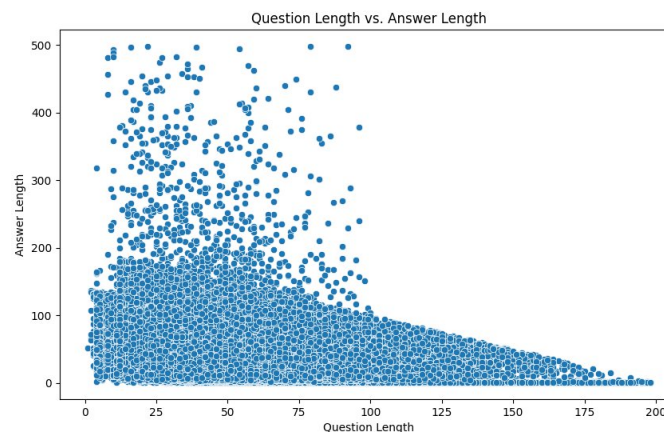


Figure 14: Question Length vs. Answer Length (source: own work based on pyplot documentation)

From scatter plot on Figure 14 we can see that some answers are much more verbose than the questions. We will remove those for consistency in chatbot interaction and regard these entries as outliers. Lengthy Question-Answers pairs are also good candidates for removal, as they are not numerous and can create inconsistencies during model training.

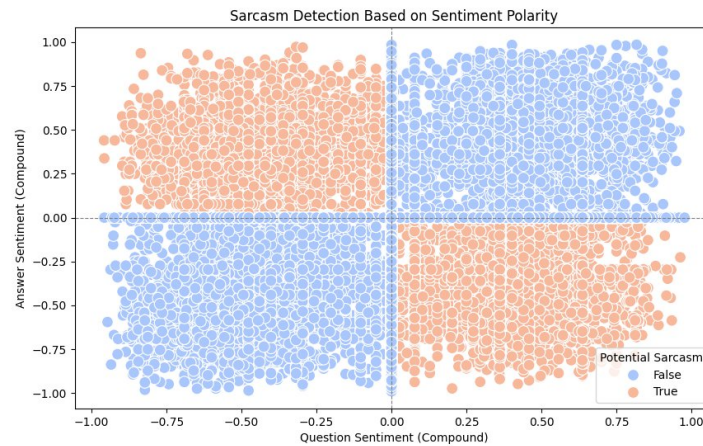


Figure 15: Sarcasm Prediction Based on Sentiment Polarity (source: own work based on pyplot documentation)

From those heated questions and answers we will extract the ones that are most likely to be sarcastic, using a simple mechanism: We will count the compound sentiment of the Question subtract it by the compound sentiment of the Answer. If the absolute difference is bigger than 0.5, we will assume that the question-answer pair is sarcastic. Figure 15, Figure 16, Figure 17 indicate that most of the entries tend to be neutral, rather than heated or sarcastic.

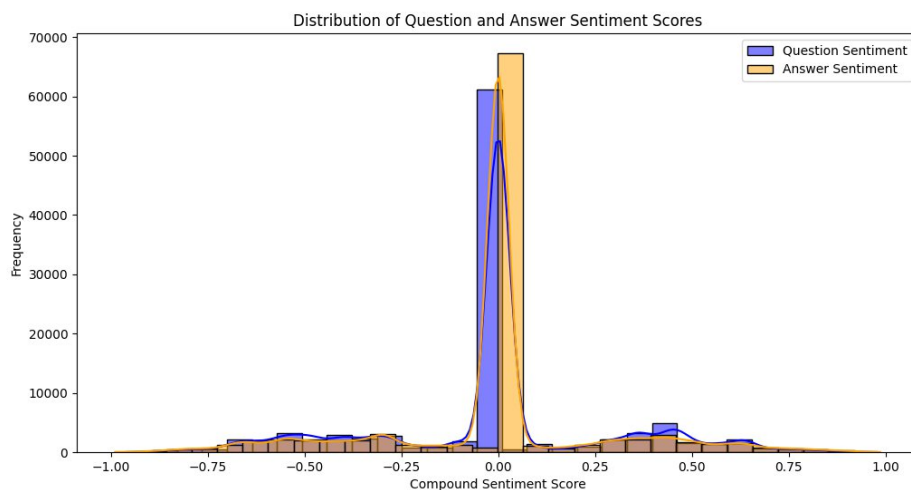


Figure 16: Distribution of Question and Answer Sentiment Scores (source: own work based on pyplot documentation)

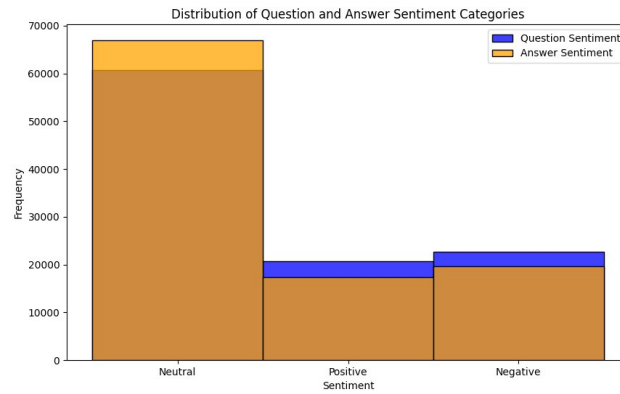


Figure 17: Distribution of Question and Answer Sentiment Categories (source: own work based on pyplot documentation)

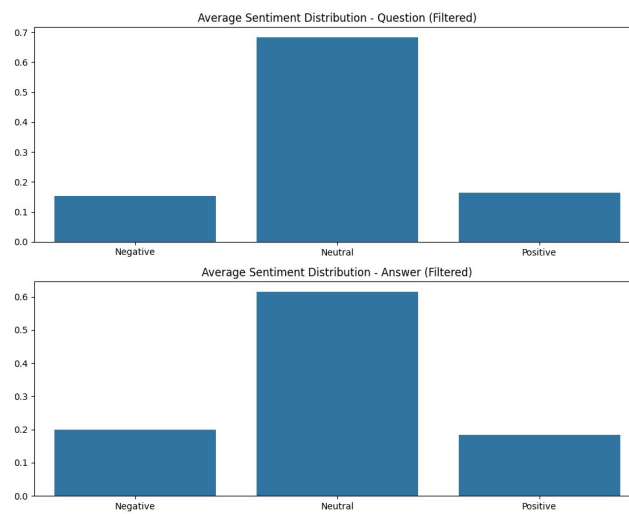


Figure 18: Average Sentiment Distribution of Questions and Average Sentiment Distribution of Answers (source: own work based on pyplot documentation)

Count plot on Figure 18 and correlation matrix on Figure 19 shows that neutral compound of sentences has rather smaller impact on sarcasm presence.

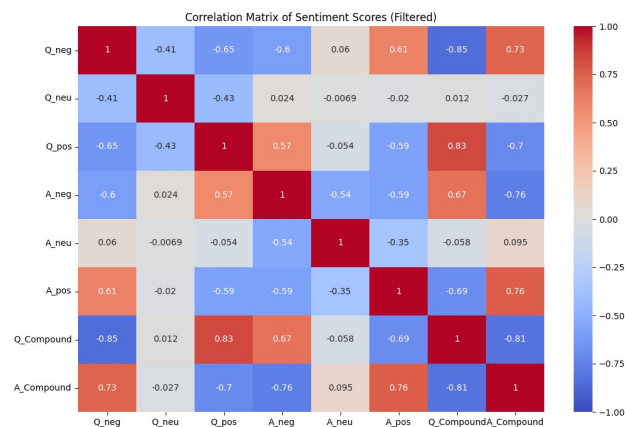


Figure 19: Heatmap of Sentiment Scores (source: own work based on seaborn documentation)

After performing filtering our dataset has left 6448 – less than 10% from our initial dataset size. However, we can see from the sample from the filtered dataset that the employed preprocessing technique performed very well.

Example of records in the filtered dataset

Q: Want to hear a joke about UDP?

A: Never mind. you won't get it, and I won't care

Q: Want to hear a joke about the election?

A: Nothing, it's already a joke.

Code Snippet 1 demonstrates an example solution for generating the *most frequently used words in questions* (Figure 12), using wordcloud library.

Code Snippet 1: Most frequently used words in questions (source: own work based on wordcloud documentation)

```
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt

questions = dataset['Question']
questions = " ".join(questions)

q_wordcloud = WordCloud(width=1920, height=1080).generate(questions)
plt.title('Most Frequently Used Words in Questions')
plt.savefig("q_cloud.jpg")
plt.show()
```

4.3. Chatbot Implementation

Chatbot UI is created using Streamlit library and consists of 3 tabs: Open-ended bot, FAQ bot, Flow-based bot. The user is able to easily switch between chatbots for quick access (Figure 20).



Figure 20: UI tabs (source: own work based on streamlit documentation)

Figure 21 shows a high-level architecture diagram based on functions in the program. Sections 4.3.1, 4.3.2 and 4.3.3 talk about implementation of each chatbot more concretely.

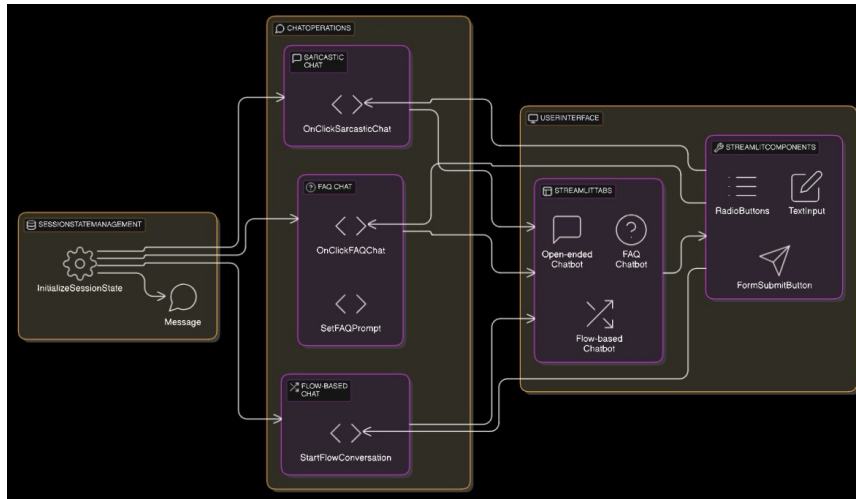


Figure 21: Function-Interaction Diagram (source: own work)

4.3.1 Rule-based Bot

The process of implementing a FAQ bot comprises on creating a dataset with predefined questions and answers. In order to make the rule-based bot a little bit more advanced RapidFuzz library was used. RapidFuzz is an open-source library distributed under MIT license. It uses string similarity algorithm to compare text and identify the nearest matches, assigning a similarity score to each comparison, based on Levenshtein distance metric.

Thus, the user is able to provide input that is similar to the predefined item in the database. This is a big improvement, but still the chat is lacking adaptivity and is still perceived by the user as shallow and dull, because it has a limited amount of predefined answers.

Code Snippet 2 demonstrates an implementation for a FAQ bot. Question-answer pairs are stored in a dictionary – *predefined_answers*. To find the closest match, *extractOne()* from *RapidFuzz* is used, setting a similarity score to 70%. When a match is found, a corresponding answer is displayed; Otherwise, an error message is displayed. The conversation is displayed in the UI and the conversation is saved to the history (*faq_history*).

Code Snippet 2: FAQ bot implementation (source: own work based on rapidfuzz documentation)

```
from rapidfuzz import process
import streamlit as st

def faq_chat():
    predefined_answers = {
        "Hello bot!": "Oh, wow, hello. It's not like I'm just here waiting for you to talk to me or anything.",
        "Who are you?": "Who am I? I'm just your personal chatbot, here to entertain you while you ask me silly questions.",
        "What are you doing?": "What am I doing? Oh, just sitting here, processing your messages and pretending like I'm not already bored.",
        "Nice day today, isn't it?": "Oh, sure, it's a 'nice' day. If by 'nice' you mean the same as every other day... then, yeah, totally nice.",
    }

    try:
        faq_prompt = st.session_state.faq_prompt
        result = process.extractOne(faq_prompt, predefined_answers.keys(), score_cutoff=70)
        if result:
            best_match = result[0]
            faq_response = predefined_answers[best_match]
        else:
            faq_response = "Sorry, I'm not able to answer for that question."
        st.session_state.faq_history.append(Message("user", faq_prompt))
        st.session_state.faq_history.append(Message("bot", faq_response))
    except Exception as ex:
        st.error(f"An error happened: {ex}")
```

Figure 22 demonstrates a fully functioning FAQ bot. Note the limitations of this simple chat – it will not answer questions that do not have a predefined or similar response in the database. Nonetheless, it can be used as a more interactive FAQ page.

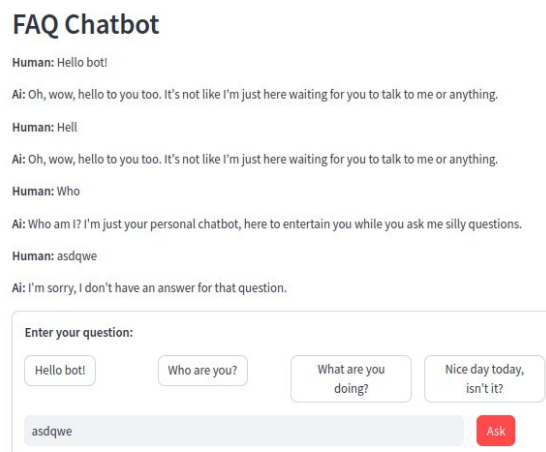


Figure 22: Rule-based chatbot (source: own work based on streamlit documentation)

4.3.2. Flow-based Bot

Flow-based bots are meant to guide users through predefined conversational steps. The involves managing the conversation flow and managing user responses at each step. In the current implementation, bot undergoes a linear path to handle a pizza ordering task. Each step in the conversation corresponds to a concrete question, i.e., choosing pizza type, size, toppings, drink.

Code Snippet 3 demonstrates an code implementation of Flow-based bot. The function *start_flow_conversation* implements a step-by-step flow and takes the order from the user. The flow is organized into four steps: selecting the type of the pizza, size, toppings, drink. Streamlit session state tracks the current step (*flow_step*), and stores the user choices into a dictionary. User is suggested to submit textual input. User proceeds to the next step by clicking “NEXT” button. Summary of the order is displayed on the UI.

Code Snippet 3: Flow-based bot implementation (source: own work based on streamlit documentation)

```
import streamlit as st

def start_flow_conversation():
    flow_steps = [
        "What type of pizza you'd like?",
        "What size pizza do you want?",
        "Would you like any toppings on your pizza?",
        "Would you like a drink to your pizza?",
    ]

    if "flow_step" not in st.session_state:
        st.session_state.flow_step = 0
    if "order" not in st.session_state:
        st.session_state.order = {"pizza_type": null, "size": null, "toppings": null, "drink": null}

    if st.session_state.flow_step < len(flow_steps):
        question = flow_steps[st.session_state.flow_step]

        if question == "What type of pizza would you like to order?":
            pizza_types = ["Margharita", "Peperoni", "Hawaian", "Vegetaran", "BBQ Chicken"]
            selected_type = st.radio("Select Pizza Type:", pizza_types, key="pizza_type")
            if st.button("Next"):
                st.session_state.order["pizza_type"] = selected_type
                st.session_state.flow_step += 1

        elif question == "What size pizza do you want?":
            sizes = ["Small", "Medium", "Large"]
            selected_size = st.radio("Select Pizza Size:", sizes, key="pizza_size")
            if st.button("Next"):
                st.session_state.order["size"] = selected_size
                st.session_state.flow_step += 1
```

```

else:
    response = st.text_input(question, key=f"flow_input_{st.session_state.flow_step}")
    if st.button("Next"):
        if response:
            if st.session_state.flow_step == 2:
                st.session_state.order["toppings"] = response
            elif st.session_state.flow_step == 3:
                st.session_state.order["drink"] = response
            st.session_state.flow_step += 1
        else:
            st.warning("Provide a response")
    else:
        st.write("order is complete!")
        st.write(f"Pizza Type: {st.session_state.order['pizza_type']}")
        st.write(f"Size: {st.session_state.order['size']}")
        st.write(f"Toppings: {st.session_state.order['toppings']}")
        st.write(f"Drink: {st.session_state.order['drink']}")

```

Figure 23 demonstrates a fully functioning Flow-based bot.

The figure consists of three screenshots of a web-based chatbot interface for ordering a pizza. Each screenshot has a title "Flow-based Chatbot" at the top.

- Top Left Screenshot:** The interface asks "Select Pizza Type:". There are five radio button options: Margherita (selected), Pepperoni, Hawaiian, Vegetarian, and BBQ Chicken. A "Next" button is at the bottom.
- Top Right Screenshot:** The interface shows "Human: Margherita" and asks "Select Pizza Size:". There are three radio button options: Small (selected), Medium, and Large. A "Next" button is at the bottom.
- Bottom Screenshot:** The interface shows "Human: Margherita" and "Human: Small". It then asks "Would you like any toppings on your pizza?". Below the question is a text input field containing "olives and garlic sauce". A "Next" button is at the bottom.

Figure 23: Flow-based bot (source: own work based on Streamlit documentation and inspired by Freed, 2021).

4.3.3. Open-ended Bot

“Open-ended Bot carries out a conversation without any pre-existing template or fixed question-answer pairs.” (Vajjala et al., 2020).

For the open-ended chatbot solution, the aforementioned “joke” dataset (4.2) is used to fine-tune DIALOGPT-SMALL⁹ Model. This is a pre-trained, open-source model imported from Hugging Face using Transformers library. In contrast to previously discussed word-embedding models (2.4.2 Word2Vec, 2.4.3 GloVe) that create static embeddings, i.e., semantic representation of the word is defined ones and is fixed, DialoGPT model generates dynamic embeddings, adjusting the vector representation of a word’s meaning based on its context within a conversation. Moreover, DialoGPT is trained on Reddit data, which makes it optimal for a chatbot solution. (Zhang et al., 2019) .

Adjusting parameters for the training is a very subtle and delicate endeavour that largely depends on available computational power, size and quality of the data, and often requires experimentation.

Figure 24 demonstrates a fully functioning Open-ended chatbot.

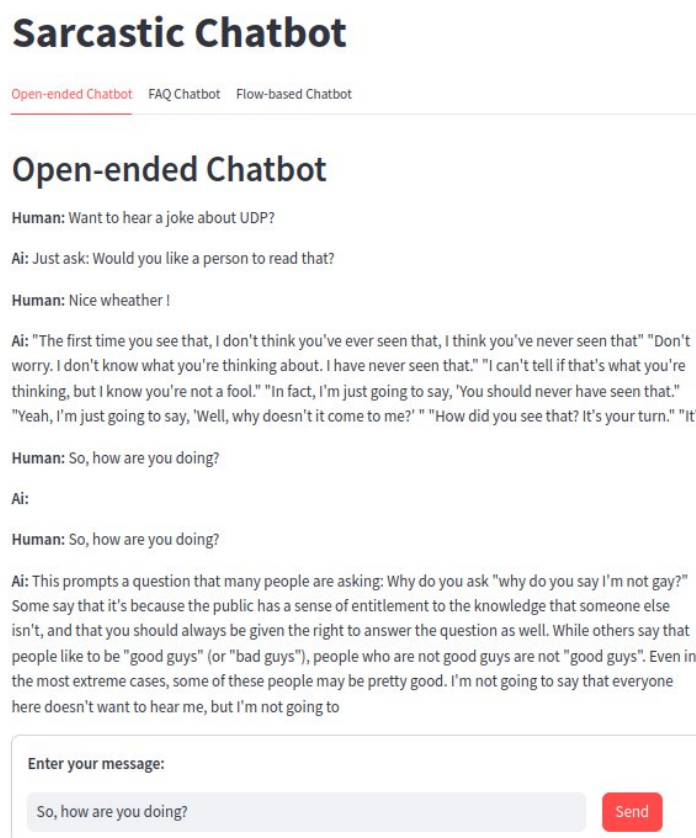


Figure 24: Open-ended chatbot (source: own work based on streamlit documentation)

⁹ <https://huggingface.co/microsoft/DialoGPT-small> (Accessed: January 28, 2025)

Chapter 5

Testing

This chapter evaluates the system's performance through manual and automated methods. Section 5.1 presents manual Quality Assurance (QA) Testing, where the system's ability to produce relevant responses is evaluated at a high-level by a human reviewer. Sentiment analysis is performed to analyse the sentiment of the bots' outputs with Vader.

5.1. Manual tests

The main purpose of manual testing is to test the functionality of the system, according to system's functional and non-functional requirements discussed in section 3.2 on a high-level human level. Taking the position of a QA manual tester, we will ensure and ascertain whether we can positively answer the following questions:

- Is the UI intuitive to navigate?
- Do all UI components work as expected?
- Can the user view the history of the conversation?
- Do the bots return responses, expected for each bot type respectively?
- Are the responses relevant and/or sarcastic, based on human subjective opinion and VADER sentiment analysis tool?

Is the UI intuitive to navigate?

As demonstrated on Figure 25, the UI is simple and self-explanatory. It includes an input field with a corresponding label, submit button, and three tabs for each chatbot type. Brevity is the soul of wit.

Sarcastic Chatbot

Open-ended Chatbot FAQ Chatbot Flow-based Chatbot

Open-ended Chatbot

Human: Want to hear a joke about UDP?

Ai: Just ask: Would you like a person to read that?

Enter your message:

Press Enter to submit form

Send

Figure 25: Open-ended Chatbot (source: own work)

Do all UI components work as expected?

Yes, the user is able to successfully send a query via input field and submit button. The user is able to smoothly switch between three chatbots via tabs.

Can the user view the history of the conversation?

As demonstrated on Figures 25, 26, the history of the conversation is saved and displayed on the screen.

Sarcastic Chatbot

Open-ended Chatbot FAQ Chatbot Flow-based Chatbot

FAQ Chatbot

Human: Who are you?

Ai: Who am I? I'm just your personal chatbot, here to entertain you while you ask me silly questions.

Human: Hello bot!

Ai: Oh, wow, hello to you too. It's not like I'm just here waiting for you to talk to me or anything.

Human: What are you?

Ai: Who am I? I'm just your personal chatbot, here to entertain you while you ask me silly questions.

Human: Nice day

Ai: Oh, sure, it's a 'nice' day. If by 'nice' you mean the same as every other day... then, yeah, totally nice.

Enter your question:

Hello bot! Who are you? What are you doing? Nice day today, isn't it?

Press Enter to submit form

Ask

Figure 26: FAQ bot (source: own work)

Do the bots return responses, expected for each bot type respectively?

Yes, FAQ bot accepts predefined input and returns predefined output. Moreover, the user can enter input similar to the predefined value, e.g., “*What* are you?” instead of “*Who* are you?”. (Figure 26).

Flow-based bot (Figure 23) is navigating the conversation flow.

Open-ended bot is generating a response without pre-existing Question-Answer pairs (Figure 24).

Are the responses relevant and/or sarcastic, based on human subjective opinion and VADER sentiment analysis tool?

Sarcastic sentiment in bots’ responses is noticeable with the naked eye. FAQ contains predefined sarcastic responses. The sentiment of the test results of an Open-ended bot was analysed with VADER (2.2.2). The sarcasm sentiment was determined by finding the compound difference between the question and answer. If the compound difference is larger than 0.5, the answer is regarded as sarcastic. Otherwise, sarcasm sentiment is detected by keywords such as "just kidding" or "hilarious." (code snippet 4).

Code Snippet 4: sentiment analysis (source: own work based on VADER documentation)

```
from nltk.sentiment import SentimentIntensityAnalyzer
import nltk
nltk.download('vader_lexicon')
sia = SentimentIntensityAnalyzer()

sarcastic_keywords = ["just kidding", "totally", "hilarious", "not really", "sure"]

def detect_sarcastic_keywords(text):
    return any(keyword in text.lower() for keyword in sarcastic_keywords)

def detect_sarcasm(question, answer):
    q_sentiment = sia.polarity_scores(question)
    a_sentiment = sia.polarity_scores(answer)
    sarcastic_sentiment = a_sentiment['compound'] - q_sentiment['compound']

    sarcasm_detected = (
        abs(sarcastic_sentiment) > 0.5 or detect_sarcastic_keywords(answer)
    )

    return sarcastic_sentiment, sarcasm_detected

# Test 1
sarcasm_score, detected = detect_sarcasm(
    "Do you want a joke on election?",
    "Oh, what a hilarious and totally unexpected turn of events in the election! Just kidding, it's a total mess."
```

```

)
print(f"Sarcasm Score: {sarcasm_score}, Detected: {detected}")

# Test 2
sarcasm_score, detected = detect_sarcasm(
    "Hello bot! Very nice weather today!",
    "Hello human! I'm glad you're enjoying the weather. It's currently 75 degrees Fahrenheit with a slight breeze coming from the east. How can I assist you today?"
)
print(f"Sarcasm Score: {sarcasm_score}, Detected: {detected}")

# Test 3
sarcasm_score, detected = detect_sarcasm(
    "Want to hear a joke about UDP?",
    "Just ask: Would you like a person to read that?"
)
print(f"Sarcasm Score: {sarcasm_score}, Detected: {detected}")

```

Test results of an Open-ended bot - first case.

Human: Do you want a joke on election?

Open-ended Bot: Oh, what a hilarious and totally unexpected turn of events in the election! Just kidding, it's a total mess.

Sentiment Analysis

Question Sentiment: {'neg': 0.0, 'neu': 0.533, 'pos': 0.467, 'compound': 0.3612}

Answer Sentiment: {'neg': 0.12, 'neu': 0.67, 'pos': 0.21, 'compound': 0.2244}

Sarcastic Sentiment (Answer - Question): 0.137

The first prompt exposes a response of sarcastic nature. At first, it exhibits excitement towards the election event ("Oh, what a hilarious and totally unexpected turn of events in the election!"). Later, it negates the initial statement, showcasing disappointment ("Just kidding, it's a total mess").

The sentiment analysis confirms the presence of sarcasm. Though the difference of compound sentiment is less than 0.5, sarcasm is detected by sarcastic keywords ("just kidding", "totally").

Test results of an Open-ended bot – second case.

Human: Hello bot! Very nice weather today!

Open-ended Bot: Hello human! I'm glad you're enjoying the weather. It's currently 75 degrees Fahrenheit with a slight breeze coming from the east. How can I assist you today?

Sentiment Analysis

Question Sentiment: {'neg': 0.0, 'neu': 0.576, 'pos': 0.424, 'compound': 0.5686}

Answer Sentiment: {'neg': 0.0, 'neu': 0.775, 'pos': 0.225, 'compound': 0.7712}

Sarcastic Sentiment (Answer - Question): 0.203

The second test shows the Open-ended bot's ability to generate a polite, appropriate response without any sarcastic sentiment. When greeted with a positive comment about the weather, the bot returns a positive response ("Hello human! I'm glad you're enjoying the weather.") and later expresses a desire to assist the user. The sentiment analysis confirms the absence of sarcasm, with a positive compound score of 0.5686 for the question and 0.7712 for the answer. No sarcastic keywords detected.

Test results of an Open-ended bot – third case.

Human: Want to hear a joke about UDP?

Open-ended Bot: Just ask: Would you like a person to read that?

Sentiment Analysis

Question Sentiment: {'neg': 0.0, 'neu': 0.533, 'pos': 0.467, 'compound': 0.3612}

Answer Sentiment: {'neg': 0.0, 'neu': 0.762, 'pos': 0.238, 'compound': 0.3612}

Sarcastic Sentiment (Answer - Question): 0.000

The third case demonstrates the bot's ability to incorporate sarcasm, implying that no one is interested in hearing a joke ("*Just ask: Would you like a person to read that?*"). The sentiment

analysis does not detect sarcastic elements (0.000), though, a reader have to agree that in this case, the sarcastic sentiment is very subtle and very hard to identify programmatically.

Test results of an Open-ended bot – fourth case.

Human: Hello bot

Open-ended bot: .

Human: How are you?

Open-ended bot: If you don't speak English, then be polite and then be polite. You are not a rude person. You are not a rude person. You are not a person. If you speak English, then be polite and then be polite.

If you use the phrase "I'll be sorry," you may not get the message. Your response should be a little more interesting. The message should be an effective answer to your frustration. If you don't know of any problems you have with the messages, please give them to me. I will send them at

Human: Wow, you response is nonsensual

Open-ended bot: . This is not normal, it is not normal, it is not normal. You will have a nice chat and make a conversation. Please send a message to the moderator. You will be sent a message to the moderator. You will be sent a message to the moderator. You will be sent a message to the moderator. You will be sent a message to the moderator. I am not an idiot. I would love to know if I could read the answers. I am not an idiot, I would love to know if I could read the answers. I am not an idiot. I would love to know if I

Forth case demonstrates chat's hallucinations (Maleki, Padmanabhan and Dutta, 2024) that occurred during generating a response.

Test results of an Open-ended bot – fifth case.

Human: Want to hear a joke about UDP?

Open-ended bot: You've probably heard it before.

The response is quite clear: That's why I've been having a rough time with my server on IPython. This is because I've had a ton of trouble getting to port over my connection to my server, and since I've been in the loop for most of this project I've been working with the server to resolve the problem. In order to do this without causing problems to the server, it's best to have a small, clean server. I also

have a few of the issues involved in getting to port over my UDP connection. You should see this in action in the following commands:

The fifth case shows chat's struggle to provide a reasonable response. It provides an allegedly sarcastic response ("*You've probably heard it before.*"), attempting to show sarcastic or rude sentiment, following by an nonsensical essay related to UDP. The bot is trying to provide a response that is the most probably related to the key word "UDP", displaying words like "port", "connection", which are, indeed, associative. Nonetheless, the chat fails to provide a reasonable response.

Test results of an Open-ended bot – sixth case.

Human: So, how are you doing?

Open-ended bot: This prompts a question that many people are asking: Why do you ask "why do you say I'm not gay?" Some say that it's because the public has a sense of entitlement to the knowledge that someone else isn't, and that you should always be given the right to answer the question as well. While others say that people like to be "good guys" (or "bad guys"), people who are not good guys are not "good guys". Even in the most extreme cases, some of these people may be pretty good. I'm not going to say that everyone here doesn't want to hear me, but I'm not going to

The sixth case shows chat's attempt to provide a heated, sarcastic response, by expressing disappointment of the query ("*This prompts a question that many people are asking:*"), following by a suggestion to enter a more controversial topic ("*Why do you ask 'why do you say I'm not gay?'*"), which is undoubtedly related to sarcasm. Then, it provides an essay on socially controversial topics. This is another example of chat's hallucinations (Maleki, Padmanabhan and Dutta, 2024).

Chapter 6

Conclusions

6.1. Evaluation

The evaluation of the chatbot system was conducted based on the results of manual testing and sentiment analysis presented in Chapter 5. The analysis focused on several key performance indicators, including system functionality, user experience, relevance of bot responses, and the detection of sarcasm.

System Functionality

Manual testing confirmed that the system met its functional and non-functional requirements. The UI was found to be intuitive and easy to navigate.

Relevance of Bot Responses

The evaluation showed that the bots successfully returned responses appropriate to their respective purposes. The FAQ bot handled predefined inputs, including minor deviations in phrasing. The Flow-based bot successfully managed conversation flows. Open-ended bot demonstrated ability to generate a reasonable sarcastic response, though failing some test cases to return expected results, demonstrating AI Hallucinations.

Sarcasm Detection and Sentiment Analysis

The Open-ended bot's ability to express sarcasm was evaluated using both human subjective opinion and the VADER sentiment analysis tool. The Open-ended bot demonstrated ability to produce sarcastic responses, as demonstrated in the first and third test cases. In the first case, sarcastic sentiment was detected through sarcastic keywords.

In the second test case, the bot responded with an appropriate and polite answer without any sarcasm. The analysis confirmed the absence of sarcasm.

In the consecutive test cases, the chat demonstrated AI Hallucinations, attempting to provide response, a related to the key words in the query, but failing to provide sensible response,

Evaluation conclusions

The differences in design and performance among the three chatbot types show their specific use cases. The FAQ and Flow-based bots performed well in predefined, task-oriented

interactions. The Open-ended bot demonstrated limited, but functionally sufficient ability to converse with the user about various topics; limited, but functionally sufficient ability to express sarcastic sentiment. Together, the system is a universal set of chatbot solutions designed to meet different user needs.

6.2. Possible Improvements

Given the aforementioned limitations in section 1.5, it is apparent that overcoming hardware performance constraints would greatly improve the project and make it more unique, distinctive and original. It would make possible to devise our own language model from scratch, without using pre-trained models.

Additionally, given the topic of the thesis is very broad and there are numerous NLP algorithms and techniques – it would be very impressive to carry out an implementation and thorough comparison of different NLP models based on performance and usage.

It is viable to continue the investigations of AI Hallucinations occurred during Open-ended chatbot's testing.

6.3. Discussion

As a result of our work, we successfully developed a highly sarcastic open-ended chatbot. My assumption is that during the fine-tuning process, the model learned the "sarcastic" parameters and now "understands" when a sarcastic response is expected.

Naturally, due to the limitations mentioned in section 1.5, a language model was not designed from scratch but instead an existing pre-trained model was utilized and fine-tuned.

Additionally, two other types of chatbots were developed for the purpose of comparison. Sentiment analysis was performed. Manual testing of the chatbots was carried out. As mentioned in section 6.2, the project is eligible for future improvement. Nonetheless, the primary goal of this thesis was fulfilled: we were able to successfully gather, interpret and apply relevant information to realize the initially planned scope of the work.

List of Figures

FIGURE 1: HEATMAP (SOURCE: OWN WORK BASED ON SEABORN, PYPLOT DOCUMENTATION; APPENDICITIS DATASET).....	11
FIGURE 2: COUNT PLOT (SOURCE: OWN WORK BASED ON SEABORN, PYPLOT DOCUMENTATION; APPENDICITIS DATASET).....	12
FIGURE 3: SCATTERPLOT DEMONSTRATING AN OUTLIER (SOURCE: OWN WORK BASED ON SEABORN, PYPLOT; APPENDICITIS DATASET)......	12
FIGURE 4: REPRESENTATION OF THE GLOVE MODEL'S CO-OCCURRENCE ANALYSIS (SOURCE: OWN WORK BASED ON PENNINGTON, SOCHER AND MANNING, 2014).....	17
FIGURE 5: CO-OCCURRENCE PROBABILITIES FOR THE TARGET WORDS (SOURCE: PENNINGTON, SOCHER AND MANNING, 2014).....	18
FIGURE 6: VISUALIZED VECTOR REPRESENTATION OF WORDS, BASED ON THEIR CONTEXT (SOURCE: OWN WORK).....	19
FIGURE 7: VISUALIZATION OF WORD2VEC VECTORS (SOURCE: OWN WORK BASED ON TEXT ANALYTICS WITH PYTHON, SARKAR, 2019)	20
FIGURE 8: TYPES OF CHATBOTS BASED ON HOW THEY ARE USED. (SOURCE: VAJJALA ET AL., 2020, P.201)	21
FIGURE 9: ARCHITECTURE DIAGRAM REPRESENTING SYSTEM ARCHITECTURE (SOURCE: OWN WORK).....	25
FIGURE 10: INTERACTION FLOW DIAGRAM (SOURCE: OWN WORK).....	25
FIGURE 11: ARCHITECTURE OVERVIEW WITH LIBRARIES (SOURCE: OWN WORK).....	26
FIGURE 12: MOST FREQUENTLY USED WORDS IN QUESTIONS FROM JOKE DATASET. (SOURCE: OWN WORK BASED ON WORDCLOUD DOCUMENTATION).....	27
FIGURE 13: MOST FREQUENTLY USED WORDS IN ANSWERS FROM JOKE DATASET (SOURCE: OWN WORK BASED ON WORDCLOUD DOCUMENTATION).....	27
FIGURE 14: QUESTION LENGTH VS. ANSWER LENGTH (SOURCE: OWN WORK BASED ON PYPLOT DOCUMENTATION).....	27
FIGURE 15: SARCASM PREDICTION BASED ON SENTIMENT POLARITY (SOURCE: OWN WORK BASED ON PYPLOT DOCUMENTATION).....	28
FIGURE 16: DISTRIBUTION OF QUESTION AND ANSWER SENTIMENT SCORES (SOURCE: OWN WORK BASED ON PYPLOT DOCUMENTATION).....	28
FIGURE 17: DISTRIBUTION OF QUESTION AND ANSWER SENTIMENT CATEGORIES (SOURCE: OWN WORK BASED ON PYPLOT DOCUMENTATION).....	29
FIGURE 18: AVERAGE SENTIMENT DISTRIBUTION OF QUESTIONS AND AVERAGE SENTIMENT DISTRIBUTION OF ANSWERS (SOURCE: OWN WORK BASED ON PYPLOT DOCUMENTATION)	29
FIGURE 19: HEATMAP OF SENTIMENT SCORES (SOURCE: OWN WORK BASED ON SEABORN DOCUMENTATION).....	29
FIGURE 20: UI TABS (SOURCE: OWN WORK BASED ON STREAMLIT DOCUMENTATION).....	30
FIGURE 21: FUNCTION-INTERACTION DIAGRAM (SOURCE: OWN WORK).....	31
FIGURE 22: RULE-BASED CHATBOT (SOURCE: OWN WORK BASED ON STREAMLIT DOCUMENTATION).....	32
FIGURE 23: FLOW-BASED BOT (SOURCE: OWN WORK BASED ON STREAMLIT DOCUMENTATION AND INSPIRED BY FREED, 2021).....	34
FIGURE 24: OPEN-ENDED CHATBOT (SOURCE: OWN WORK BASED ON STREAMLIT DOCUMENTATION).....	35
FIGURE 25: OPEN-ENDED CHATBOT (SOURCE: OWN WORK).....	37
FIGURE 26: FAQ BOT (SOURCE: OWN WORK).....	37

Code Listings

CODE SNIPPET 1: MOST FREQUENTLY USED WORDS IN QUESTIONS (SOURCE: OWN WORK BASED ON WORDCLOUD DOCUMENTATION).....	30
CODE SNIPPET 2: FAQ BOT IMPLEMENTATION (SOURCE: OWN WORK BASED ON RAPIDFUZZ DOCUMENTATION).....	32
CODE SNIPPET 3: FLOW-BASED BOT IMPLEMENTATION (SOURCE: OWN WORK BASED ON STREAMLIT DOCUMENTATION).....	33
CODE SNIPPET 4: SENTIMENT ANALYSIS (SOURCE: OWN WORK BASED ON VADER DOCUMENTATION).....	38

Bibliography

Di Nunzio, G. M. and Vezzani, F. (2018) A Linguistic Failure Analysis of Classification of Medical Publications: A Study on Stemming vs Lemmatization. Dept. of Information Engineering University of Padua, Italy. Available at: <https://ceur-ws.org/Vol-2253/paper45.pdf> (Accessed: January 28, 2025).

Freed, A. (2021) Conversational AI. Shelter Island, NY: MANNING.

Hutto, C. J. and Gilbert, E. (2014) VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. Available at: <https://ojs.aaai.org/index.php/ICWSM/article/view/14550/14399> (Accessed: January 28, 2025).

Khan, R. and Das, A. (2018) Build Better Chatbots: A Complete Guide to Getting Started with Chatbots. Bangalore, Karnataka, India: APRESS.

Lovins, J. B. (1968) Development of a Stemming Algorithm. Available at: <https://aclanthology.org/www.mt-archive.info/MT-1968-Lovins.pdf> (Accessed: January 28, 2025).

Maleki, N., Padmanabhan, B. and Dutta, K. (2024) AI Hallucinations: A Misnomer Worth Clarifying. Available at: <https://arxiv.org/pdf/2401.06796> (Accessed: January 28, 2025).

Mikolov, T. et al. (2013) Efficient Estimation of Word Representations in Vector Space. Google Inc., Mountain View, CA. Available at: <https://arxiv.org/pdf/1301.3781> (Accessed: January 28, 2025).

Pennington, J., Socher, R. and Manning, C. (2014) GloVe: Global Vectors for Word Representation. Computer Science Department, Stanford University, Stanford, CA 94305. Available at: <https://nlp.stanford.edu/pubs/glove.pdf> (Accessed: January 28, 2025).

Samuels, J. I. (2024) One-Hot Encoding and Two-Hot Encoding: An Introduction. UK: Imperial College. Available at: https://www.researchgate.net/publication/377159812_One-Hot-Encoding_and_Two-Hot-Encoding_An-Introduction#full-text (Accessed: January 28, 2025).

Sarkar, D. (2019) Text Analytics with Python: A Practitioner's Guide to Natural Language Processing. Second Edition. Bangalore, Karnataka, India: APRESS.

Sarkar, D., Bali, R. and Sharma, T. (2018) Practical Machine Learning with Python: A Problem-Solver's Guide to Building Real-World Intelligent Systems. Bangalore, Karnataka, India: APRESS.

Skuridin, A. and Wynn, M. (2024) Chatbot Design and Implementation: Towards an Operational Model for Chatbots. Available at: <https://doi.org/10.3390/info15040226> (Accessed: January 28, 2025).

Vajjala, S. et al. (2020) Practical Natural Language Processing: A Comprehensive Guide to Building Real-World NLP Systems. O'REILLY.

Zhang, Y. et al. (2019) DIALOGPT : Large-Scale Generative Pre-training for Conversational Response Generation. Microsoft Corporation, Redmond, WA, USA. Available at: <https://arxiv.org/pdf/1911.00536> (Accessed: January 28, 2025).

Citation style: "Cite Them Right 10th edition – Harvard"