MENTAL HEALTH CHATBOT

B.Tech CSE, 8th Semester, 2022-23

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY, WEST BENGAL

Utech

In Pursuit Of Knowledge And Excellence

Mentor

Prof. Bulbul Mukherjee

Assistant Professor
Dept. of CSE & IT
Bengal Institute of Technology, Kolkata

Group Members

Name	MAKAUT Roll No.
Snehasish Jana	12100119098
Ankita Biswas	12100119107
Shilpi Mondal	12100120096
Subha Rana	12100119068
Prince Sarkar	12100119069

BT

BENGAL INSTITUTE OF TECHNOLOGY

BENGAL INSTITUTE OF TECHNOLOGY
No. 1, Basanti Highway, Brahmapur Government Colony, Bagdoba,
Kolkata, 700150

Acknowledgement

*This project would not been a successful without the sincere cooperation and guidance of our mentor **Prof. Bulbul Mukherjee**, Assistant Professor, Dept of CSE & IT, Bengal Institute of Technology, Kolkata, who has provided us with useful resources and motivation through the various phases of this project which has prove to be crucial for the completion of this documentation.*

Signature of Group Members:

Name:
MAKAUT Roll No.

Name:
MAKAUT Roll No.

Name:
MAKAUT Roll No.

Name:
MAKAUT Roll No.

Name:
MAKAUT Roll No.

Signature of Mentor

Prof. Bulbul Mukherjee
Assistant Professor
Dept. of CSE & IT
Bengal Institute of Technology

Signature of HOD

Dr. Shanta Phani
Head of the Department
Dept. of CSE & IT
Bengal Institute of Technology

Abstract: *Crypto is a mental health conversational AI chatbot that can help people from depression, anxiety, stress, etc. This chatbot web application is mainly built for those students or teenagers who have trouble taking help from a mental health professional or their family. These past few years have taught us how important mental health is far more than your physical health. And a psychiatrist may charge high costs per session and for a middle class person it is not affordable to put their money on things that are not their priority. Crypto is here for the people to use whenever they want without any cost. It can act as your friend and talk with you whenever you need it. Crypto helps users tackle negative thoughts and emotions. It determines the basic emotions of a user from the natural language input using natural language processing and the tools of RASA (Open Source Conversational AI framework). An attempt has been made to help people with mental health issues as they are hesitant in sharing these thoughts and emotions with other people.*

Introduction: Mental health is a major concern worldwide and India is not far behind in sharing this. If we evaluate developments in the field of mental health, the pace appears to be slow. Dr. Brock Chisholm, the first Director-General of the World Health Organization (WHO), in 1954, had presciently declared that "without mental health there can be no true physical health." More than 60 years later, the scenario has not altered substantially. About 14% of the global burden of disease is attributed to neuropsychiatric disorders. A study examining suicidal behaviour during India's COVID-19 lockdown by the International Journal of Mental Health Systems has found a 67.7% increase in online news media reports of suicidal behaviour. Over 60% reported disruptions to mental health services for vulnerable people, including children and adolescents (72%), older adults (70%), and women requiring antenatal or postnatal services (61%). 30% reported disruptions to access to medications for mental, neurological, and substance use disorders. Around three-quarters reported at least partial disruptions to school and workplace mental health services (78% and 75% respectively). 70% of people have had more stress and anxiety at work this year than any other previous year. This increased stress and anxiety have negatively impacted the mental health of 78% of the global workforce, causing more stress (38%), a lack of work-life balance (35%), burnout (25%), depression from a lack of socialization (25%), and loneliness (14%).

Consumers certainly seem interested in mental health chatbots. A 2021 national survey commissioned by Woebot Health, one of the leading therapeutic chatbot companies, found that 22% of adults had used a mental health chatbot, and 47% said they would be interested in using one if needed. Among the respondents who had tried a mental health chatbot, nearly 60% said they began this use during the COVID-19 pandemic, and 44% said they used chatbots exclusively and do not see a human therapist. The most common reasons people cited as to why they used and/or might be interested in using a chatbot included that the tool was cheap, easy to use, and accessible anytime.

In a study when asked about the perceived importance of chatbots ,77% of professionals and experts in mental health thought that chatbots are either somewhat important (66%) or very important (11%). In contrast, 23% believed that chatbots are somewhat unimportant (20%) or very unimportant (3%).

Domain: We have worked in different domain for completion of this project. We have used RASA framework for Natural Language Processing and Python Language for Custom Pipeline and Custom Action Codes. We have used HTML, CSS and JS for our web application and for API calling we have used socket.io.

RASA

Let't explain about RASA files, which are created as Initial project structure of Rasa.

__init__.py: an empty file that helps python find your actions. actions.py: code for your custom actions. If you need the bot to write specific

config.yml: This file contains the configuration we set up for our NLU and Core models. Policies, pipelines etc are defined here.

credentials.yml: As the name suggests, this file is used to store the details (credentials) of various connected services. Remember that you need to host Rasa over https domain.

data/nlu.yml: This file contains the training data which consists of example user utterances categorized by intent. NLU is the name given by RASA which helps turn User Messages into structured data. Here you define Intents. In each intent, are defined examples of user utterances, entities, and how to respond. Basically, intents are labels that represent the goal or meaning. Like depression or anxiety. You need to add related Sentences for that Intent. NLU training data is defined under the **nul** key. Items that can added under this key：

Training examples grouped by user intent and listed under the **examples** key.

Rules.yml: Rules are listed under the **rules** key and look similar to stories. A rule also has a **steps** key, which contains a list of the same steps as stories do. Rules can additionally contain the **conversation_started** and **conditions** keys. These are used to specify conditions under which the rule should apply.

data/stories.yml: This is required for Rasa Core. There is something called "Dialog Flow in Rasa" where Rasa Core controls the flow of the conversation between you and the chatbot, so for that flow, you need to train the chatbot using these stories. So in case you want your chatbot to be very perfect in different contexts (stories) you can add those stories here. This are important in deciding the next action of the chatbot.

domain.yml: your assistant's domain. This file combines Different Intent which chatbot can detect and list of Bot replies. Remember you can define your Custom Action Server Python method name here, so that Rasa will call that python method for you. Define the environment of the chatbot and the guidelines on how the chatbot should function. Some key aspects of the domain file are intents, entity, responses, actions, slots, session_config.

endpoints.yml: This contains the different endpoints the chatbot can utilize. It details for connecting to channels like FB messenger. This is mainly used for production setup. You can configure your Database like Redis so that Rasa can store tracking information.

Action.py: This file contains the main logic of the chatbot. Actions define the responses of the chatbot. All actions executed by the bot are specified with the **action:** key followed by the name of the action. While writing stories, you will cncounter two types of actions:

Reponses: Start with **utter_** and send a specific message to the user.

Custom actions: Start with **action_ ,** run arbitrary code and send any number of messages.

Web Application: For implementing Chatbots as a webapplication, different programming languages are used.

It has information about the chatbot and web page design. Features, about and blogs and the other part is the chatbot. We've hosted the chatbot on the website using a react based chatroom component provided by scalable minds. The web app currently runs on localhost.

The programming languages that are used here are HTML5, CSS, JS.

HTML: HTML is Hyper Text Markup Language, which extensively used to develop simple Websites with GUI(Graphical User Interface) providing user with different elements like Button,

Textbox, Text field, Check boxes, Radio Buttons. Here language is in the form of tags. Multi-media contents like Videos, Images can be handled using HTML.

CSS: CSS is Cascading Style Sheets is a style sheet language which is used to add more look and appearance a HTML page. A tag <style> is used and the style that developer wants in his HTML page in that place of the page is written in CSS.

JavaSript: JavaScript is used to handle behaviours of different elements in an HTML page. It is an Object Oriented, prototype-based language.

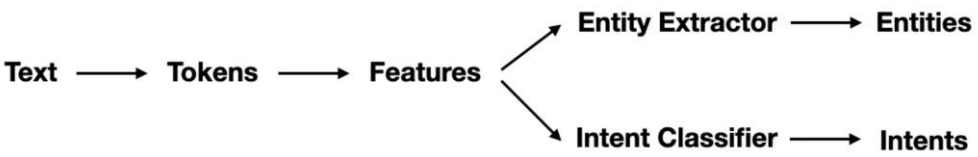
Javascript handles complex tasks. It enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else.

Dataset: Data was collected from Kaggle, Reddit and many other websites. 921 lines of a total of 44 intents were created. Stories building is a very important task of building a Rasa Chatbot. A total of 27 stories based on happy, sad, depressed, anxious, stressed and many such moods were created.

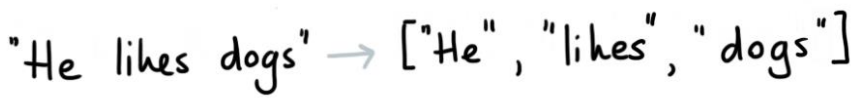
We have created data that had intents such as depressed, mood_happy, mood_sad, stressed, etc. Intents are actual messages the user wants to say to the chatbot. Creating stories based on these intents.

Methods: We have tried two different pipelines and two different algorithms to train our model: DIET intent classifier and Sparse Naïve Bayes Intent Classifier. In this case for better accuracy we have used DIET and its corresponding pipeline components.

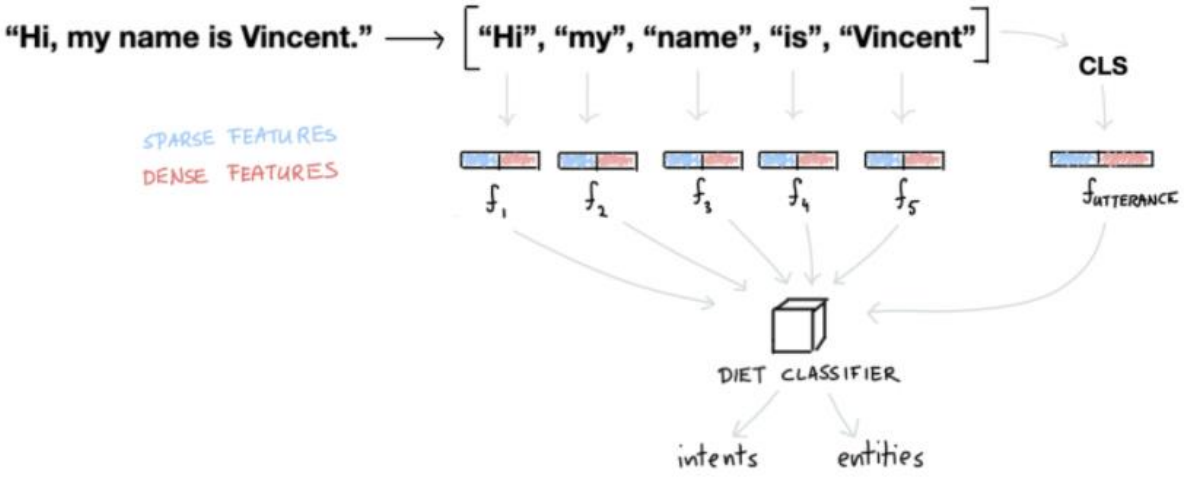
RASA NLU PIPELINE: In a Rasa project, the NLU pipeline defines the processing steps that convert unstructured user messages into intents and entities. It consists of a series of components, which can be configured and customized by developers. A pipeline in Rasa defines the dependency relationship and data flow direction between the different components, and it allows the developer to configure each of the components. The pipeline gives the Rasa framework great flexibility and extensibility. The NLU Pipeline is defined in the “config.yml” in Rasa. This file contains all the steps present in the pipeline that will be used by Rasa to classify the intents and perform the appropriate action. All the pipelines that we have used for this project are explained below.



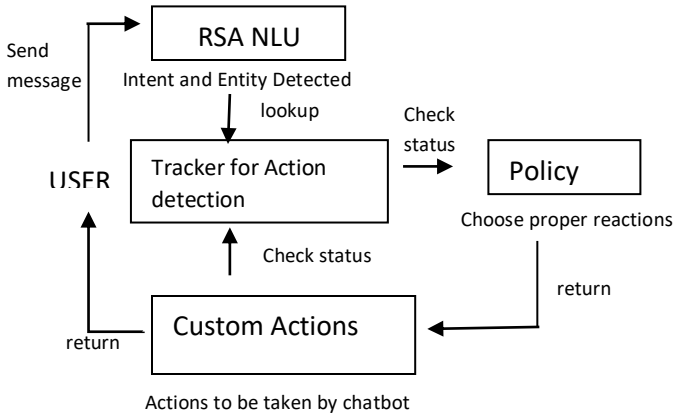
Tokenizer: The tokenizer breaks text into terms whenever it encounters a whitespace character.



Featurizer: Creates a vector representation of user message using regular expressions.



In the whole system of RASA the data flow is like following:



Pic: *RASA CHATBOT*

Algorithm Used: We have used Naïve Bayes algorithm for intent classification. Here are some code snippets given to explain how our intent classification model is working.

First, we need to define the model class, which needs to inherit from the IntentClassifier Rasa class, and can be imported as follows:

```
from rasa.nlu.classifiers.classifier import IntentClassifier
```

In our model class named SparseNaiveBayesIntentClassifier the parameters are defined as follows:

```
class SparseNaiveBayesIntentClassifier(IntentClassifier):

    @classmethod
    def required_components(cls) -> List[Type[Component]]:
        return [SparseFeaturizer]

    defaults = {
        "alpha": 1.0,
        "binarize": 0.0,
        "fit_prior": True,
        "class_prior": None,
    }
```

The intent classification class is containing the following methods, which will be used by Rasa to train and use the model:

1. train()

2. process()

3. persist()

4. load()

train() method: This method will be called by Rasa to train the model using data specified in nlu.yml and is called when rasa train command is executed. As an input, it gets training data (train_data) and configuration parameters (cfg) specified in the config.yml. The train method does not return anything and updates the model, which in our example is stored in an attribute model of our custom class.

```
def train(
        self,
        training_data: TrainingData,
        config: Optional[RasaNLUModelConfig] = None,
        **kwargs: Any,
    )
```

Before process method another method will be called to obtain every sparse and dense features of the training data.

```
def prepare_data(
        self, training_data: TrainingData
    ) -> Tuple[scipy.sparse.spmatrix, np.ndarray]:
```

process() method. This method is used to get the prediction out of the model and record it. It is called when the rasa shell command is executed, and Rasa is passed a message. It will predict a probable outcome. This method does not need to return anything, it just updates the message passed as the input, adding the predicted intent and its confidence score to the dictionary stored in the message.data. The prediction added to the dictionary should be in the specific Rasa format: {"name": value, "confidence": confidence}.

```
def process(self, message: Message, **kwargs: Any) -> None:
        if not self.clf:
            intent = None
            intent_ranking = []
        else:
            X = self._get_sentence_features(message)
            intent_ids, probabilities = self.predict(X)
            intents = self.transform_labels_num2str(np.ravel(intent_ids))
```

persist() method. This method saves the trained model as a .pkl, so that it can be loaded for prediction when interacting with Rasa. As the input, it gets the following attributes: file_name, and model_dir, which contain the name of the file and the directory where it needs to be saved. This method needs to return the model's file name in the form of the dictionary: {"file": model_file}, so that it later can be retrieved.

```
def persist(self, file_name: Text, model_dir: Text) -> Optional[Dict[Text, Any]]:

        classifier_file_name = file_name + "_classifier.pkl"
        encoder_file_name = file_name + "_encoder.pkl"
        if self.clf and self.le:
            io_utils.json_pickle(
                os.path.join(model_dir, encoder_file_name), self.le.classes_
            )
            io_utils.json_pickle(
                os.path.join(model_dir, classifier_file_name), self.clf
            )
        return {"classifier": classifier_file_name, "encoder": encoder_file_name}
```

load() method: This method loads the model saved by the persist method. In order to load the model, we need to get its directory and the file name. This information is passed in the meta and and model_dir arguments.

```
def load(
       cls,
       meta: Dict[Text, Any],
       model_dir: Optional[Text] = None,
       model_metadata: Optional[Metadata] = None,
       cached_component: Optional["SparseNaiveBayesIntentClassifier"] = None,
       **kwargs: Any,
   )
```

We have used custom actions for our chatbot such as:

Receive_name: It basically receives names by tracking 'name' entity.

```
class ActionRecieveName(Action):

    def name(self) -> Text:
        return "action_recieve_name"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        text = tracker.latest_message['text']
        dispatcher.utter_message(text=f"I'll remember your name {text}!")
        return [SlotSet("name", text)]
```

Remember_name: It utters the name when asked which it has received earlier in a chat session.

```
class ActionSayName(Action):

    def name(self) -> Text:
        return "action_say_name"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: Dict[Text, Any]) -> List[Dict[Text, Any]]:

        name = tracker.get_slot("name")
        if not name:
            dispatcher.utter_message(text="I don't know your name.")
        else:
            dispatcher.utter_message(text=f"Your name is {name}!")
        return []
```

Accuracy: Telling about accuracy our model has performed quite well. We have got some satisfactory result for beginning phase. Our precision score is **90%** and accuracy is **83%**.

Confusion Matrix of our project is given below:

[[16 0 0 0 0 0 0 0]

[0 3 0 0 0 0 0 0]

[0 0 3 0 0 0 0 0]

[0 0 0 2 0 0 0 2]

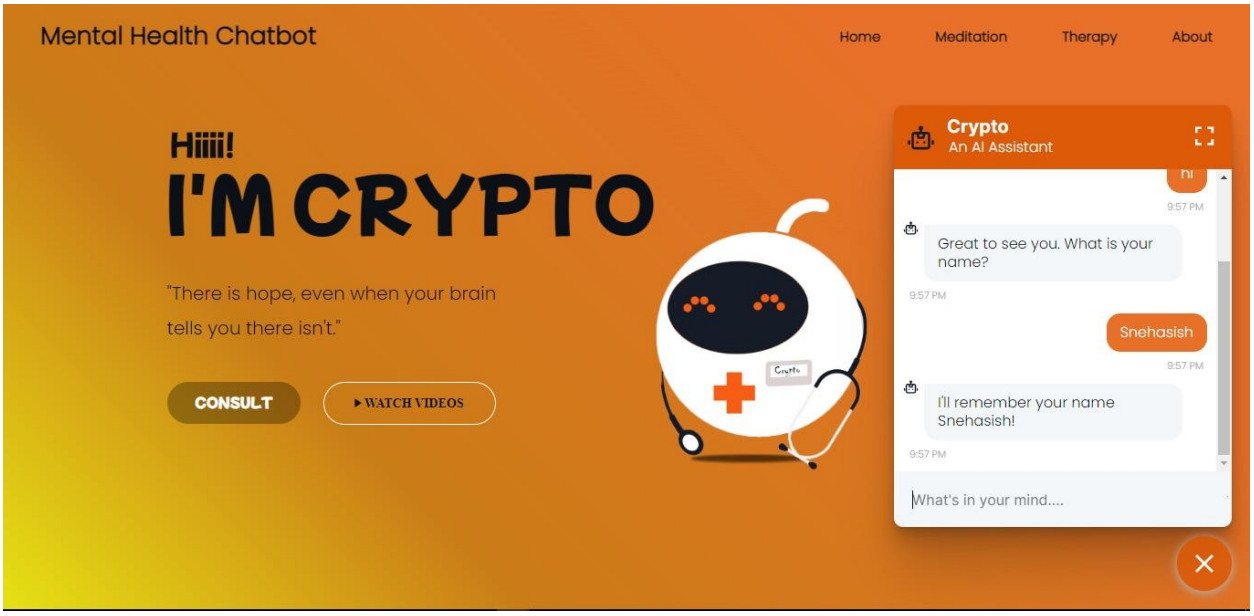
[0 0 0 0 5 0 0 0]

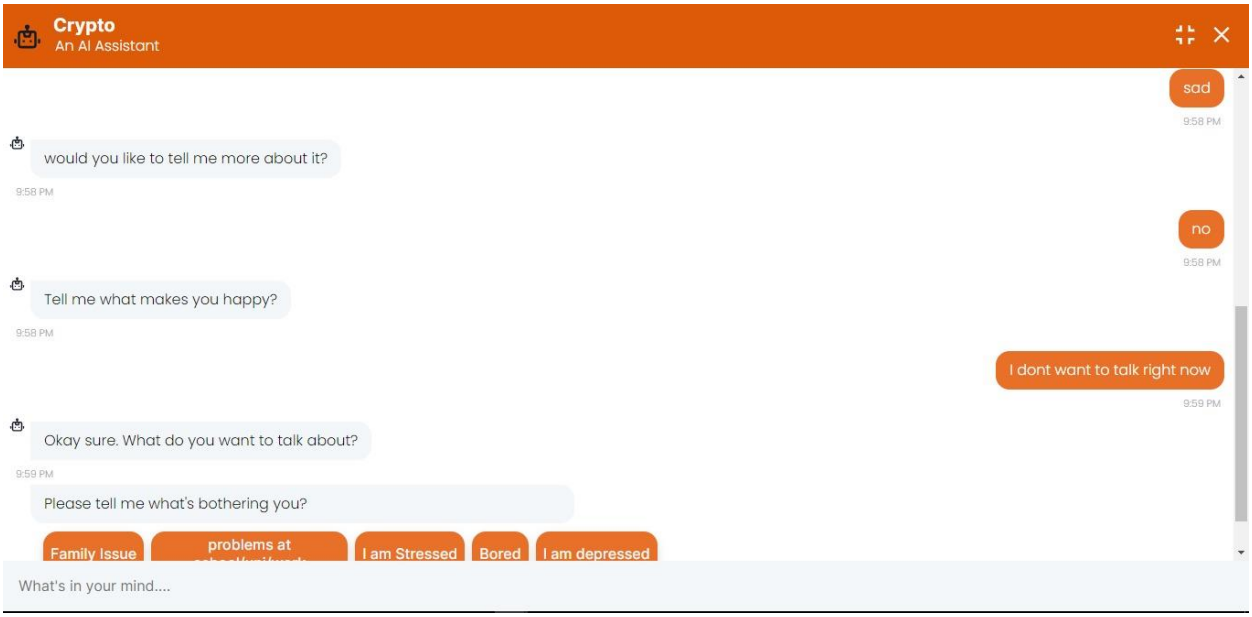
[0 0 0 0 0 3 0 0]

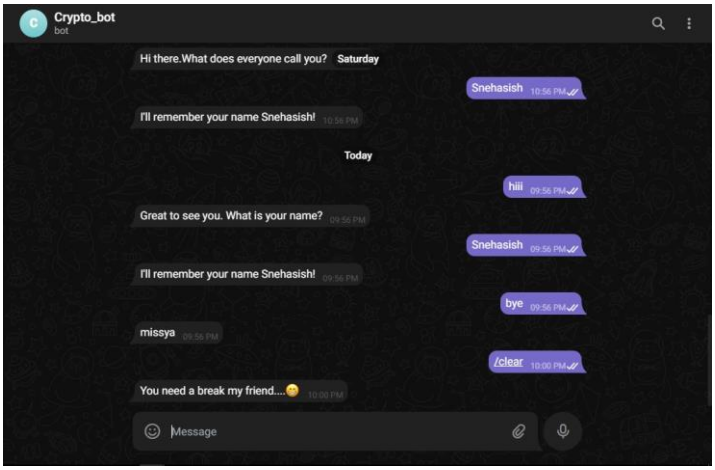
[0 0 0 0 0 0 1 0]

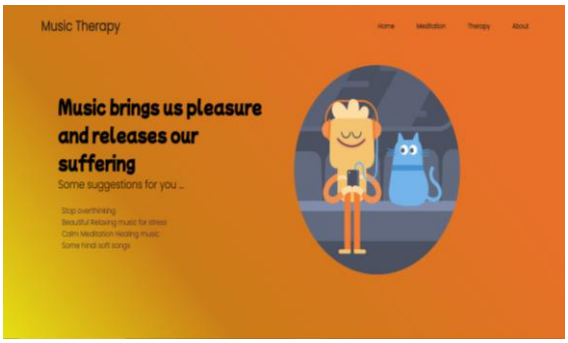
[0 0 0 0 0 0 0 0]]

Implementations: We have implemented our chatbot in web application as well as telegram. That means we have created a telegram bot which can be more user friendly. We have connected our chatbot to web through socket and to telegram through ngrok. Both are working on localhost.









Conclusion & Future Scope: Crypto helps support users and gain mental stability. Human connection is not something that can be achieved with a chatbot but helping people who don't have the resources to treat themselves. Even though chatbots can make a conversation they only mimic understanding but they don't exactly understand. This can cause resistance as chatbots are prone to mistakes. These mistakes are something that can be avoided and worked upon in the future. One of the most important aspects when dealing with users is to keep a conversation going. Understanding every detail the user chats about and is where the future of chatbots relies upon. Keeping an option of interacting with an actual therapist is also feasible. The user's privacy has to be taken into consideration. The information that the user shares with the bot has to be kept confidential.

References:

[1] https://rasa.com/docs/rasa/

[2] https://rasa.com/docs/rasa/nlu-training-data

[3] https://rasa.com/docs/rasa/components

[4] https://www.kaggle.com/datasets/elvis23/mental-health-conversational-data

[5] https://www.researchgate.net/figure/Rasa-NLU-and-Rasa-Core-Flowchart_fig4_359382833

[6] https://medium.com/@Zack.hardtoname/rasa-mechanism-work-flow-simply-explained-b44e85d5a6f1

[7] https://intellifysolutions.com/blog/know-open-source-conversational-ai-rasa/

[8] https://rasahq.github.io/rasa-nlu-examples/docs/classifier/naive-bayes/

[9] https://bhashkarkunal.medium.com/conversational-ai-chatbot-using-deep-learning-how-bi-directional-lstm-machine-reading-38dc5cf5a5a3

[10] https://botfront.io/blog/introducing-a-flow-chart-editor-for-rasa-forms