CREATE A CHATBOT IN PYTHON

Name:E. Sandhiya

Reg.No: 513521106031

Department: ECE

Year: III

NM ID: au513521106031

Email:sandhiya892004@gmail.com

PROBLEM STATEMENT:

Develop a chatbot using python that can effectively engage with users, provide helpful information, and perform tasks or answer questions across various domains. The chatbots should be able to understand natural language input and provide appropriate responses, ensuring a smooth and userfriendly conversational experience.

PROJECT OVERVIEW:

1. Natural Language Understanding: Implement natural language processing(NLP) techniques to enable the chatbot to understand and interpret user input effectively. This includes intent recognition, entity extraction, and context awareness.
2. Multi-Domain Support: Design the chatbot to handle a wide range of user queries across different domains, such ass customer support, general knowledge, e-commerce, and more.

3. Personalization: Incorporate features that allow the chatbot to remember user preferences and maintain context throughtout the conversation, providing a personalized user experience.

4. Task Automation: Enable the chatbot to perform tasks or actions on behalf of users, including making recommendations, scheduling appointments, or processing orders.

5. Integration: Integrate the chatbot with external data sources, APIs, or databases to fetch real-time information or execute specific actions. This may involve accessing weather data, retrieving product details, or connecting to booking systems.

DESIGN THINKING:

Design thinking is a human-centered approach to problem-solving and innovation. When designing a chatbot using Python, applying design thinking principles can help create a user-centric and effective conversational AI system.

Here's a design thinking process tailored for developing a chatbot using Python:

1. Empathize: Understand User Needs

2. Define: Clearly Define the Problem

3. Ideate: Brainstorm Solutions

4. Prototype: Build a Minimum Viable Product (MVP)

5. Test: Gather Feedback

6. Implement: Develop the Chatbot

7. Deploy: Make the Chatbot Accessible

8. Gather Feedback: Continuous Improvement

9. Iterate: Continuous Development

10. Scale: Handle Growth

11. Monitor: Maintain and Monitor

By following this design thinking process, you can develop a chatbot using Python that not only meets user needs but also evolves over time to provide an increasingly valuable conversational experience. The iterative nature of design thinking ensures that the chatbot remains relevant and effective in a dynamic environment.

PROBLEM DEFINTION:

Chatbots in customer support comes with interpreting the messages and understanding the user intention. Programming flexible algorithms for interpreting the intention of the message is a top priority upon making a chatbot.

1. Empathize: Understand User Needs
   User Research: Begin by conducting in-depth user research to understand the needs, pain points, and preferences of your target audience. Gather feedback through surveys, interviews, and user observations. User Personas: Create user personas to represent different user segments, each with unique characteristics and requirements for the chatbot.
2. Define: Clearly Define the Problem
   Problem Statement: Based on the insights gained during the empathize phase, define a clear problem statement that the

chatbot will address. Consider what specific tasks or issues the chatbot will help users with.

3. Ideate: Brainstorm Solutions
   Cross-Functional Team: Assemble a cross-functional team that includes developers, designers, NLP specialists, and domain experts to brainstorm creative solutions.

4. Prototype: Build a Minimum Viable Product (MVP)
   Develop MVP: Create a basic version of the chatbot using Python. Focus on core functionalities that address the defined problem. Use Python libraries or frameworks suitable for chatbot development.

5. Test: Gather Feedback
   Usability Testing: Conduct usability testing with real users to gather feedback on the chatbot's functionality, user interface, and overall user experience.

6. Implement: Develop the Chatbot Programming: Develop the chatbot using Python and relevant libraries or frameworks. Implement natural language understanding (NLU) and natural language generation (NLG) components for effective communication.

7. **Deploy**: Make

the Chatbot Accessible Hosting: Deploy the chatbot to a hosting environment that ensures availability and scalability. Consider cloudbased hosting solutions.

8. Gather feedback: Continuous Improvement

User Feedback: Encourage users to provide feedback on their interactions with the chatbot. Implement feedback mechanisms within the chatbot.

9. Iterate: Continuous Development

Iterative Development: Use an iterative development approach to make regular updates and enhancements to the chatbot based on user feedback, changing requirements, and emerging technologies.

10. Scale: Handle Growth

Scalability: Ensure that the chatbot can handle increased usage as its user base grows. Optimize performance and resource utilization. Security: Implement robust security measures to protect user data and ensure safe interactions.

11. Monitor: Maintain and Monitor

Ongoing Maintenance: Continuously monitor and maintain the chatbot to address any issues, bugs, or performance bottlenecks.

POSSIBLE FUTURE WORK:

There are several exciting and potentially transformative directions for future work on chatbots using Python or other bot development platforms. Here are some possible future avenues for chatbot development and research:

- Multilingual and Cross-Cultural Chatbots: Create chatbots that can understand and respond to users in multiple languages and adapt to cultural nuances. This will make chatbots more inclusive and globally accessible.

- Voice-Enabled Chatbots: Extend chatbot capabilities to support voice interactions, enabling users to have natural conversations without typing. Integrating speech recognition and synthesis technologies will be essential.

These future directions for chatbot development and research hold the potential to significantly enhance the capabilities, usability, and impact

of chatbots in various domains and applications. Chatbot developers and researchers can explore these areas to stay at the forefront .

GPT-3:

The Generative Pre-Trained Transformer (GPT) is an innovation in the Natural Language Processing (NLP) space developed by OpenAI. These models are known to be the most advanced of its kind and can even be dangerous in the wrong hands. It is an unsupervised generative model which means that it takes an input such as a sentence and tries to generate an appropriate response, and the data used for its training is not labelled.

GPT 3 Chatbot: A Magic Tool to Supersede Human Workforce:

The AI platform has grabbed gazillions of eyeballs, with people trying out the new label worldwide. Not only this, but the innovation has successfully entered the list of tech companies loved by the tech giant Microsoft. The news suggests that the company is all set to invest billions of dollars in the OpenAI GPT 3 chatbot.

Difference Between The ChatGPT and GPT-3:

People thought Google would always be the go-to source for all kinds of info. It was never capable of delivering everything. However, it was perhaps the only source that incrementally improved, eventually emerging with a fabulous array of tools and resources for individuals and businesses to benefit from. All that Google achieved now seems

like a proverbial stoneage in the face of the OpenAI algorithm and its different models. ChatGPT, for example, is an unprecedented worry for a robust entity.

GPT-3 and innovation in natural language processing (NLP):

- The OpenAI laboratory, based in California, launched this year a new model of artificial intelligence language called GPT-3, or Generative Pretrained Transformer. The new generation of the program is nothing more than a predictor of texts that performs a simple action – that of autocomplete -, but represents an important technical advance for the NLP area.

Natural Language Processing:

- Human natural language, by definition, is any means of communication developed by humans and without premeditation. For example, languages, which are made up of signs specific to the language of a location, require people to know how to “decode” the signs of a language in order to understand them. • Human-machine communication exists through the ability to process natural language, that is, the computer needs inputs – data and parameters – to process certain information, understand it, and eventually bring a coherent response.

LIMITATIONS OF GPT-3 CHATBOTS:

- 1. Natural language processing:

GPT-3 chatbots do not understand human-like conversational text or words. Instead, generating a correct answer like humans becomes difficult. It sometimes provides irrelevant output code or answers due to a pre-trained model. The output that it gives lacks the perfect understanding of the words. However, due to the language model used, the chatbot is restricted from answering questions correctly.

• 2. Uncertainty of input leads to ambiguity :

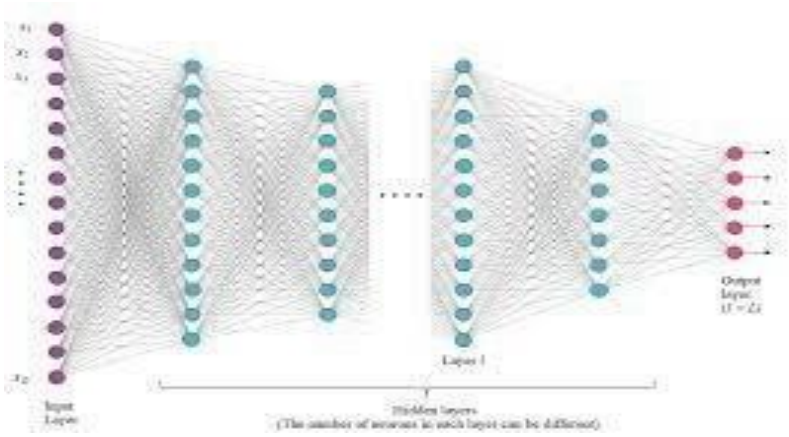
Chatbot is not similar to Google search engine, where you can throw any questions to get accurate answers. To generate text from the chatbot exactly the way you want is only possible if the language you use is certain. Be specific with the questions and language you use, as the chatbot will struggle with complex questions resulting in response errors.

NEURAL NETWORK:

It is a deep learning algorithm that resembles the way neurons in our brain process information (hence the name). It is widely used to realize

the pattern between the input features and the corresponding output in a dataset. Here is the basic neural network architecture. The purplecolored circles represent the input vector, xi where i =1, 2, ….., D which is nothing else but one feature of our dataset. The blue-colored circles are the neurons of hidden layers. These are the layers that will learn the math required to relate our input with the output. Finally, we have pink-colored circles which form the output layer. The dimension of the output layer depends on the number of different classes that we have. For example, let us say we have 5x4 sized dataset where we have 5 input vectors, each having some value for 4 features: A, B, C, and D. Assume that we want to classify each row as good or bad and we use the number 0 to represent good and 1 to represent bad. Then, the neural network is supposed to have 4 neurons at the input layers

neurons at the output.



Neural Network algorithm involves two steps:

1. Forward Pass through a Feed-Forward Neural Network
2. Backpropagation of Error to train Neural Network

NEURAL LANGUAGE PREPROCESSING:

At the beginning of the chatbot development process, however, you may face the lack of training data, which results in low accuracy in intent classification.

A few workarounds exist to solve this problem:

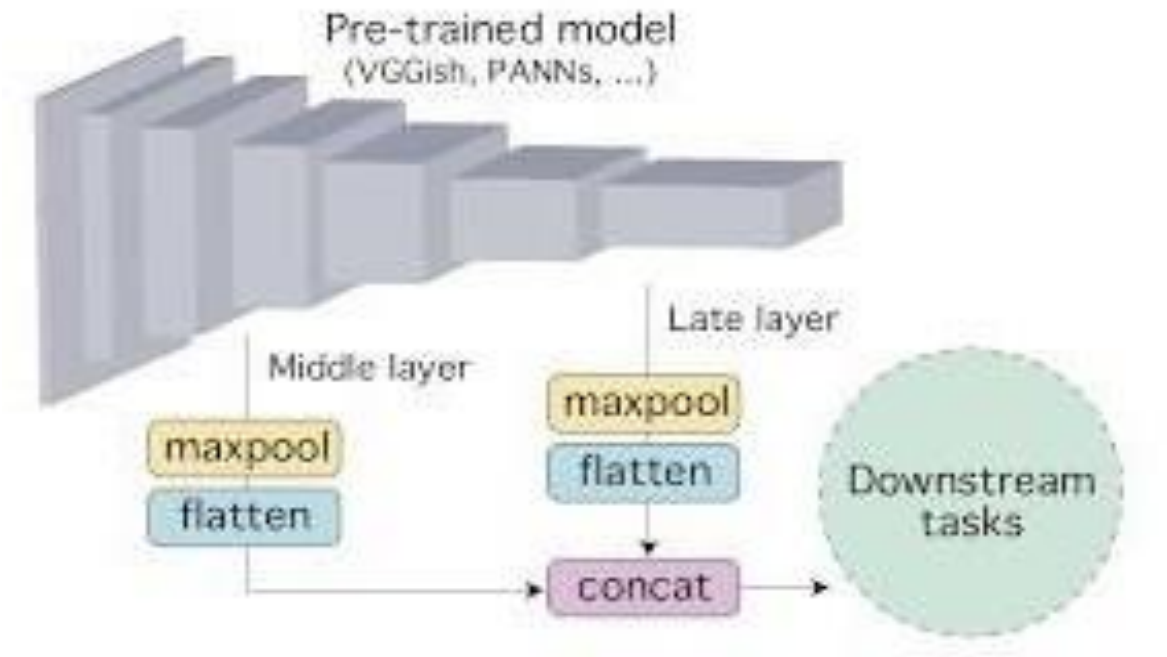
1. Pre-trained model

2. Training data generator 3. Crowdsource

1.Pre-Trained model:

The pre-trained language model can be used for NLU tasks without any task-specific change to the model architecture. Pre-trained models have an ability to continue pre-training on custom data, strarting from some checkpoint.

This process is compute-intensive and requires a massively parallel compute infrastructure. The training of such general models on taskspecific training data, called Fine-tuning, is far less computationally demanding and used more often. Well-known examples of similar models are Google’s BERT, XLNet and OpenAI’s GPT-2.



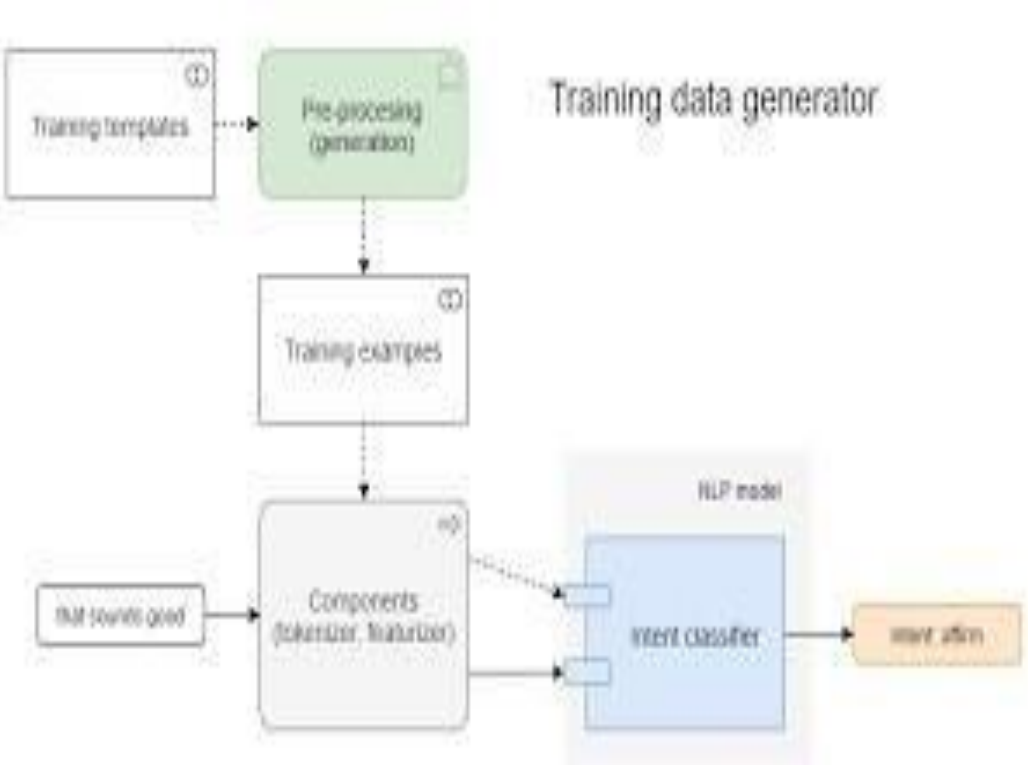
Top areas in which pretrained models are advancing AI include:

- **Natural language processing.** Pretrained models are used for translation, chatbots and other natural language processing applications. Large language models, often based on the transformer model architecture, are an extension of pretrained models. One example of a pretrained LLM is NVIDIA NeMo Megatron, one of the world's largest AI models.
- **Speech AI.** Pretrained models can help speech AI applications plug and play across different languages. Use cases include call center automation, AI assistants and voice-recognition technologies.
- **Computer vision.** Like in the unicorn example above, pretrained models can help AI quickly recognize creatures — or objects, places and people. In this way, pretrained models accelerate computer vision, giving applications human-like vision capabilities across sports, smart cities and more.
- **Healthcare.** For healthcare applications, pretrained AI models like MegaMolBART — part of the NVIDIA BioNeMo service and framework — can understand the language of chemistry and

learn the relationships between atoms in real-world molecules, giving the scientific community a powerful tool for faster drug discovery.

- **Cybersecurity.** Pretrained models provide a starting point to implement AI-based cybersecurity solutions and extend the capabilities of human security analysts to detect threats faster. Examples include digital fingerprinting of humans and machines, and detection of anomalies, sensitive information and phishing.
- **Art and creative workflows.** Bolstering the recent wave of AI art, pretrained models can help accelerate creative workflows through tools like GauGAN and NVIDIA Canvas.
- Pretrained AI models can be applied across industries beyond these, as their customization and fine-tuning can lead to infinite possibilities for use cases.

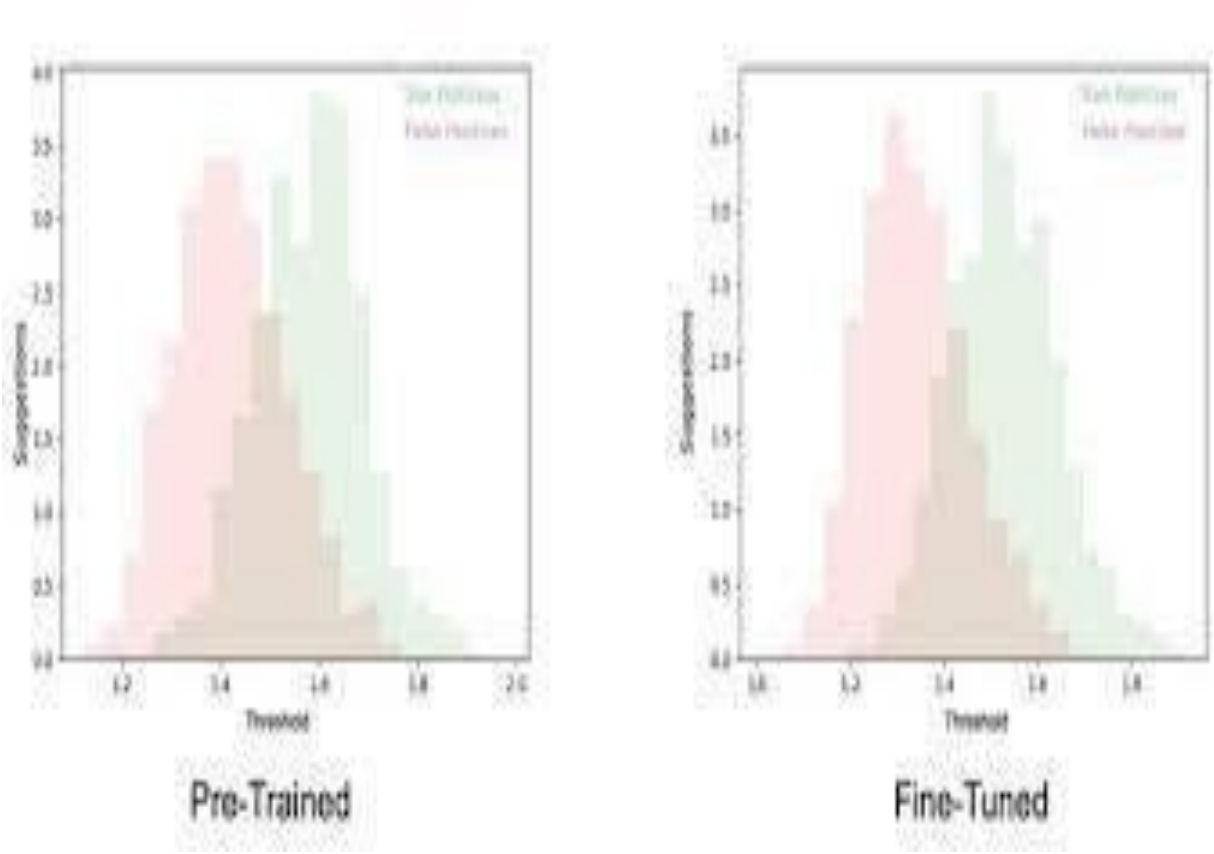
2. Training data generator:



1. *Import Necessary Libraries:* python import re import string from nltk.corpus import stopwords from nltk.tokenize import word_tokenize
2. *Lowercasing:*
   Convert all text to lowercase to ensure consistency in text analysis. python text = text.lower()
3. *Tokenization:*
   Tokenize the text into words or sentences. python words = word_tokenize(text)
4. *Remove Punctuation:*
   Remove punctuation marks from the text. python words = [word for word in words if word.isalnum()]
5. *Remove Stop Words:*
   Remove common stop words (e.g., "the," "and," "is") that don't contribute much to the meaning. python stop_words = set(stopwords.words('english')) words = [word for word in words if word not in stop_words]
6. *Stemming or Lemmatization (Optional):*
   Reduce words to their base or root form to improve text normalization. python # Using NLPK for stemming from nltk.stem import PorterStemmer stemmer = PorterStemmer() words = [stemmer.stem(word) for word in words] # Using NLTK for lemmatization from nltk.stem import WordNetLemmatizer lemmatizer = WordNetLemmatizer() words = [lemmatizer.lemmatize(word) for word in words]
7. *Custom Cleaning (if needed):*

Depending on your specific use case, you might want to perform additional cleaning steps like handling special characters or domain-specific terms.

8. *Rejoin Text:*
   Rejoin the words into a cleaned text string. python cleaned_text=' '.join(words)



Chatbots can provide real-time customer support and are therefore a valuable asset in many industries. When you understand the basics of the ChatterBot library, you can build and train a selflearning chatbot with just a few lines of Python code.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	99 MB	0.749	0.921	25,636,712	168
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

Chatbots have become an integral part of modern applications, enhancing user engagement and providing instant support. In this tutorial, we'll walk through the process of creating a chatbot using the powerful GPT model from OpenAI and Python Flask, a micro web framework. By the end of this guide, you'll have a functional chatbot that can hold interactive conversations with users.

Basic understanding of python programming.

Familiarity with web development and API's.

DATASET:

1. *Conversation Dataset*:

- A dataset of conversations is fundamental. It should include both user messages and the corresponding chatbot responses. Each conversation typically consists of a series of message pairs.

2. *Intent Dataset*:

- If your chatbot is intent-based, you'll need a dataset that maps user messages to specific intents. This helps the chatbot understand what the user wants.

3. *Entity Recognition Dataset*:

- If your chatbot needs to extract specific information (entities) from user messages, you should have a dataset with annotated entities. For instance, in a restaurant chatbot, you'd annotate restaurant names, dates, and locations.

4. *User Feedback Dataset*:

- A dataset that includes user feedback on the chatbot's responses is valuable for improving the chatbot's performance. Users can rate responses as helpful or not and leave comments.

5. *Small Talk Dataset*:

- For general chatbots or social chatbots, a dataset of small talk or general knowledge questions and responses is helpful. This provides a basis for engaging users in casual conversations.

6. *Multilingual Dataset*:

- If your chatbot supports multiple languages, you'll need a dataset that covers a variety of languages and language-specific nuances.

7. *Custom Domain Dataset*:

- If your chatbot is tailored for a specific domain, gather or create a dataset relevant to that domain. For example, if it's a medical chatbot, you'd need a dataset of medical queries and responses.

8. *Contextual Dataset*:

- For chatbots that maintain context over multiple turns in a conversation, you need a dataset that reflects these extended interactions. This allows the chatbot to handle multi-turn conversations effectively.

9. *Real-World Data*:

- Whenever possible, use real-world data to make your chatbot more representative of actual user interactions. This can be obtained from customer support logs, chat transcripts, or user-generated content.

10. *Synthetic Data*:

- In cases where real data is limited, you can generate synthetic data. Tools like ChatGPT or data augmentation techniques can help create additional training data.

11. *Noisy Data*:

- Include data that reflects the natural variability and noise in user inputs. Real users may use typos, abbreviations, or unconventional language.

12. *Negative Examples*:

- Provide examples of what users should not do. For instance, if you're building a customer support chatbot, include examples of inappropriate or unhelpful user messages.

13. *Balanced Dataset*:

- Ensure a balanced distribution of intents or responses in your dataset to prevent biases and help your chatbot handle different user scenarios.

14. *Testing and Evaluation Data*:

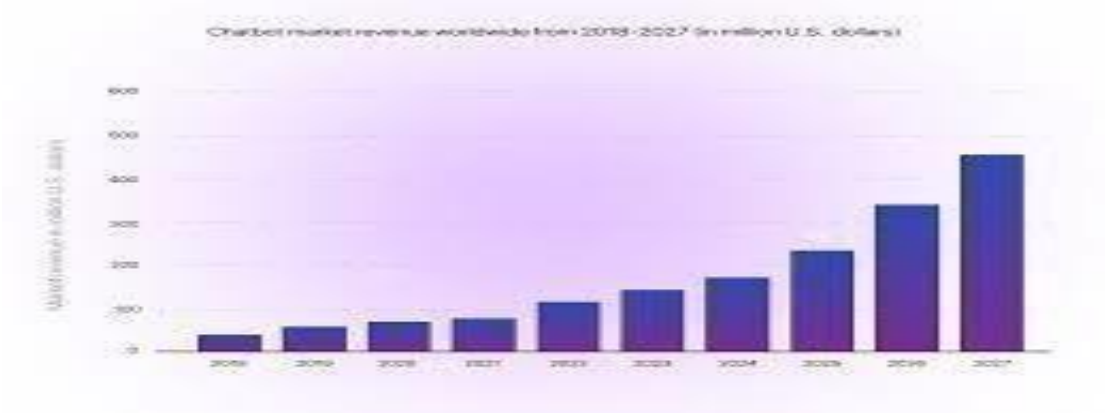
- Reserve a portion of your data for testing and evaluation. This allows you to assess your chatbot's performance without the risk of overfitting.

15. *Data Annotation Guidelines*:

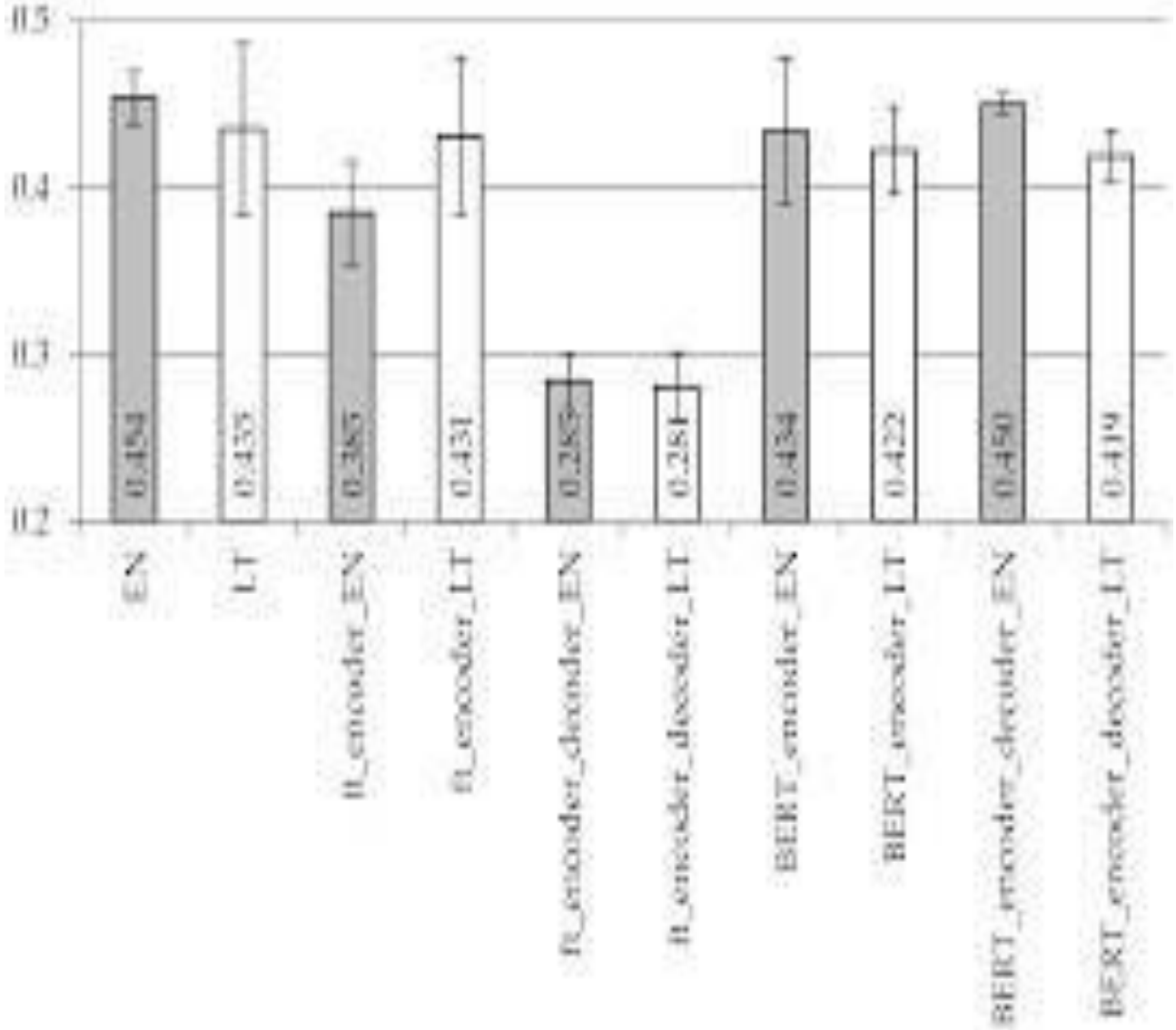
- Document clear annotation guidelines for human annotators to maintain consistency in the dataset annotations.

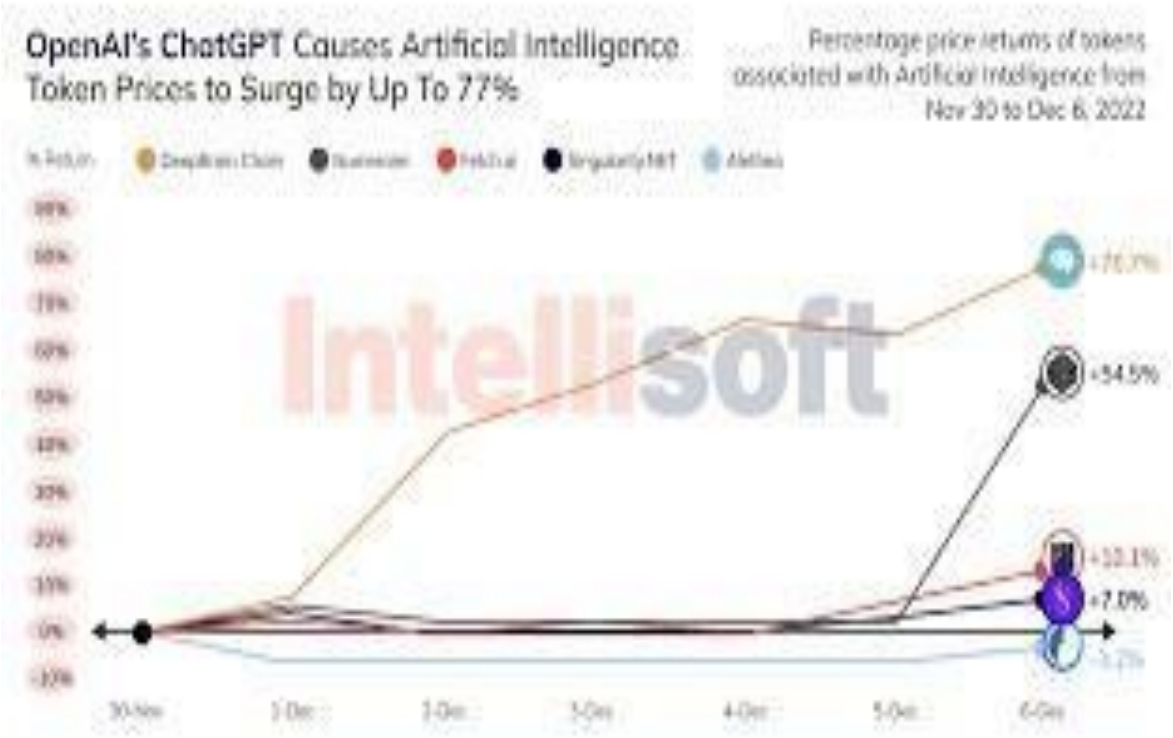
16. *Privacy and Compliance*:

- Be mindful of privacy regulations and ensure that the data you collect and use complies with relevant data protection laws.




Fine Tuning Vs. Training From Scratch
Accuracy [%]
Training samples [#]
BERT fine-tuning
Training from scratch


0.5
0.4
0.3
0.2
0.454
0.435
0.365
0.431
0.285
0.281
0.434
0.422
0.450
0.419
EN
LT
fr_encoder_EN
fr_encoder_LT
fr_encoder_decoder_EN
fr_encoder_decoder_LT
BERT_encoder_EN
BERT_encoder_LT
BERT_encoder_decoder_EN
BERT_encoder_decoder_LT



Initialize the flask:

#app.py

#import files from flask import Flask,

render_template, request app = Flask(__name__)

@app.route("/") def home(): return "Hello, This is

Flask Application" if__name___== "__main__":

app.run()

```
<!DOCTYPE html>
<html>
<head>
    <title>GenAI-Bot</title>
```

```
    <meta name="viewport"
content="width=devicewidth, initial-scale=1">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/
3.2.1/jquery.min.js"></script>
    <style>         * {
box-sizing: border-box
        }
/* Set height of body and the document to 100%
*/         body, html {              height:
100%;              margin: 0;              font-
family: Arial;
        }          #chatbox
{          margin-left:
auto;          margin-
right: auto;
width: 40%;
margin-top: 60px;
      }
      #userInput {
margin-left: auto;
margin-right: auto;
width: 40%;          margin-
top: 60px;
      }
      #textInput {          width: 90%;
border: none;          border-bottom:
3px solid black;          font-family:
monospace;          font-size: 17px;
      }       .userText {
color: white;          font-
```

```
family: monospace;
font-size: 17px;          text-
align: right;          line-
height: 30px;
      }
      .userText span {
background-color: #808080;
padding: 10px;          border-
radius: 2px;
      }       .botText {
color: white;          font-
family: monospace;
font-size: 17px;          text-
align: left;          line-
height: 30px;        }
      .botText span {
background-color: #4169e1;
padding: 10px;          border-
radius: 2px;
      }
      #tidbit  {
position: absolute;
bottom: 0;          right:
0;          width: 300px;
      }       .boxed {
margin-left: auto;
margin-right: auto;
width: 78%;          margin-top:
60px;          border: 1px solid
green;
      }
```

```
    </style>
</head>
<body>
<div>
        <h1 align="center"><b>AI-Gen
ChatBot</b></h1>
        <h4 align="center"><b>Please start your
personalized interaction with the
chatbot</b></h4>
        <p align="center"><img
src="static\\chatbot1.png" alt="Python-BOT"
height="210" width="220"></p>
        <div class="boxed">
            <div>
                <div id="chatbox">
                    <p class="botText">
                        <span>Hi! I'm your AI-
Generative Chatbot</span>
                    </p>
                </div>
                <div id="userInput">
<input id="textInput" type="text" name="msg"
placeholder="Message" />
                </div>
            </div>             <script>
function getBotResponse() {
var rawText =
$("#textInput").val();
var userHtml = '<p
class="userText"><span>' + rawText +
"</span></p>";
```

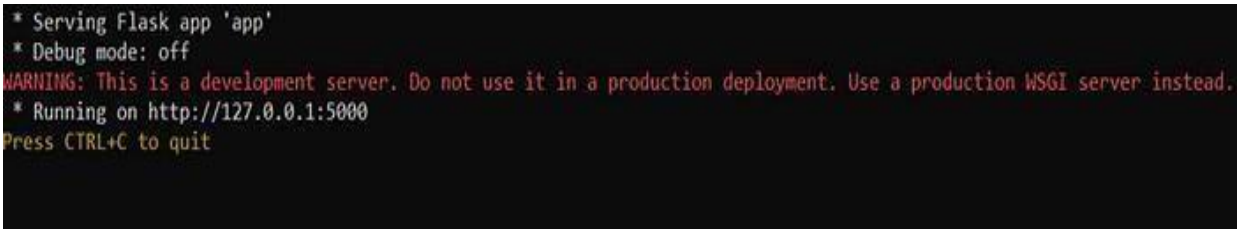
```
                        $("#textInput").val("");

$("#chatbox").append(userHtml)
;                         document
.getElementById("userInput")
                            .scrollIntoView({ block:
"start", behavior: "smooth" });
                        $.get("/get", { msg: rawText
}).done(function (data) {
var botHtml = '<p class="botText"><span>' +
data + "</span></p>";

$("#chatbox").append(botHtml);
document
.getElementById("userInput")
                                  .scrollIntoView({
block: "start", behavior: "smooth" });
                        });
                    }

$("#textInput").keypress(function
(e) {                         if (e.which ==
13) {
getBotResponse();
                        }
});
            </script>
        </div>
    </div>
</body>
</html>
```

```
#Final app.py #import files from flask import
Flask, render_template, request import openai
app = Flask(__name__) openai.api_key  = "<place
your openai_api_key>"
 def get_completion(prompt, model="gpt-3.5-
turbo"):     messages = [{"role": "user",
"content": prompt}]     response =
openai.ChatCompletion.create(
model=model,          messages=messages,
temperature=0, # this is the degree of
randomness of the model's output
    )
    return
response.choices[0].message["content"]
@app.route("/") def home():
return render_template("index.html")
@app.route("/get") def
get_bot_response():
    userText = request.args.get('msg')
response = get_completion(userText)
#return str(bot.get_response(userText))
return response if __name__ ==
"__main__":
```



FEATURE ENGINEERING:

AI-Based Chatbots:

With the rise in the use of machine learning in recent years, a new approach to building chatbots has emerged. Using artificial intelligence, it has become possible to create extremely intuitive and precise chatbots tailored to specific purposes. Unlike their rule-based kin, AI based chatbots are based on complex machine learning models that enable them to self-learn. Now that we're familiar with how chatbots work, we'll be looking at the libraries that will be used to build our simple Rule-based Chatbot.

Natural Language Toolkit(NLTK):

Natural Language Toolkit is a Python library that makes it easy to process human language data. It provides easy-to-use interfaces to many language-based resources such as the Open Multilingual Wordnet, as well as access to a variety of text-processing libraries.

Regular Expression (RegEx) in Python:

A regular expression is a special sequence of characters that helps you search for and find patterns of words/sentences/sequence of letters in sets of strings, using a specialized syntax. They are widely used for text searching and matching in UNIX. Python includes support for regular expression through the re package.

Building a chatbot:

This very simple rule based chatbot will work by searching for specific keywords in inputs given by a user. The keywords will be used to

understand what action the user wants to take (user's intent). Once the intent is identified, the bot will then pick out a response appropriate to the intent.

The list of keywords the bot will be searching for and the dictionary of responses will be built up manually based on the specific use case for the chatbot.

We'll be designing a very simple chatbot for a Bank. The bot will be able to respond to greetings (Hi, Hello etc.) and will be able to answer questions about the bank's hours of operation

Flow of how the chatbot will process

We will be following the steps below to build our chatbot

1. Importing Dependencies
2. Building the Keyword List
3. Building a dictionary of Intents
4. Defining a dictionary of responses
5. Matching Intents and Generating Responses

Importing dependencies:

The first thing we'll need to do is import the packages/libraries we'll be using. re is the package that handles regular expression in Python. We'll also be using WordNet from NLTK. WordNet is a lexical database that defines semantical relationships between words. We'll be using WordNet to build up a dictionary of synonyms to our keywords. This will help us expand our list of keywords without manually having to introduce every possible word a user could use.

```
# Importing modules   import re

from nltk.corpus import wordnet
```

Building a List of Keywords:

hince we have imported our libraries, we'll need to build up a list of keywords that our chatbot will look for. This list can be as exhaustive as you want. The more keywords you have, the better your chatbot will perform.

As discussed previously, we'll be using WordNet to build up a dictionary of synonyms to our keywords.

Code:

```
 # Building a list of Keywords

list_words=['hello','timings']

list_syn={} for word in

list_words:

synonyms=[]

 for syn in wordnet.synsets(word):  for

lem in syn.lemmas():

# Remove any special characters from synonym strings

lem_name = re.sub('[^a-zA-Z0-9 \n\.]', ' ',  lem.name())

synonyms.append(lem_name)

 list_syn[word]=set(synonyms)
```

```
print (list_syn)
```

OUTPUT:

```
Hello

{'hello', 'howdy', 'hi', 'hullo', 'how do you do'}

timings

{'time', 'clock', 'timing'}
```

PYTHON:

```
# Import "chatbot" from

# chatterbot package.

from chatterbot  import ChatBot



# Inorder to train our bot, we have

# to import a trainer package #

"ChatterBotCorpusTrainer"

from chatterbot.trainers import
ChatterBotCorpusTrainer




# Give a name to the chatbot “corona bot”

# and assign a trainer component.
```

```
chatbot=ChatBot('corona bot')


# Create a new trainer for the chatbot trainer

= ChatterBotCorpusTrainer(chatbot)


# Now let us train our bot with multiple corpus

trainer.train("chatterbot.corpus.english.greetin

gs",

              "chatterbot.corpus.english.convers

ations" )


response = chatbot.get_response('What is your

Number') print(response)


response = chatbot.get_response('Who are you?')
print(response)
```

Output:

```
Training greetings.yml: [####################] 100%
Training conversations.yml: [#######             ] 33%
[nltk_data] Downloading package stopwords to /home/nikhil/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/nikhil/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
Training conversations.yml: [####################] 100%
I don't have any number
I am just an artificial intelligence.
```

Creating a Chatbot using Python:

We will follow a step-by-step approach and break down the procedure of creating a Python chat.

We will begin building a Python chatbot by importing all the required packages and modules necessary for the project. We will also initialize different variables that we want to use in it. Moreover, we will also be dealing with text data, so we have to perform data preprocessing on the dataset before designing an ML model.

This is where tokenizing supports text data - it converts the large text dataset into smaller, readable chunks (such as words). Once this process is complete, we can go for lemmatization to transform a word into its lemma form. Then it generates a pickle file in order to store the objects of Python that are utilized to predict the responses of the bot.

Understanding the ChatterBot Library

ChatterBot is a Python library that is developed to provide automated responses to user inputs. It makes utilization of a combination of Machine Learning algorithms in order to generate multiple types of responses. This feature enables developers to construct chatbots using Python that can communicate with humans and provide relevant and appropriate responses. Moreover, the ML algorithms support the bot to improve its performance with experience.

Another amazing feature of the **ChatterBot** library is its language independence. The library is developed in such a manner that makes it possible to train the bot in more than one programming language.

Chatbot in Python:

In the past few years, chatbots in the Python programming language have become enthusiastically admired in the sectors of technology and business. These intelligent bots are so adept at imitating natural human languages and chatting with humans that companies across different industrial sectors are accepting them. From e-commerce industries to healthcare institutions, everyone appears to be leveraging this nifty utility to drive business advantages. In the following tutorial, we will understand the chatbot with the help of the Python programming language and discuss the steps to create a chatbot in Python.

Understanding the Chatbot:

A **Chatbot** is an Artificial Intelligence-based software developed to interact with humans in their natural languages. These chatbots are generally converse through auditory or textual methods, and they can effortlessly mimic human languages to communicate with human beings in a human-like way. A chatbot is considered one of the best applications of natural languages processing.

We can categorize the Chatbots into two primary variants: **RuleBased Chatbots** and **Self-Learning Chatbots**.

1. **Rule-based Chatbots:**The Rule-based approach trains a chatbot to answer questions based on a list of predetermined rules on which it was primarily trained. These set rules can either be pretty simple or quite complex, and we can use these rule-based chatbots to handle simple queries but not process more complicated requests or queries.
2. **Self-learning Chatbots:**Self-learning chatbots are chatbots that can learn on their own. These leverage advanced technologies such as Artificial Intelligence (AI) and Machine Learning (ML) to train themselves from behaviours and instances. Generally, these chatbots are quite smarter than rule-based bots. We can classify the Self-learning chatbots furtherly into two categories - **Retrieval-based Chatbots** and **Generative Chatbots**.
    a. **Retrieval-based Chatbots:**A retrieval-based chatbot works on pre-defined input patterns and sets responses. Once the question or pattern is inserted, the chatbot utilizes a heuristic approach to deliver the relevant response. The model based on retrieval is extensively

utilized to design and develop goal-oriented chatbots using customized features such as the flow and tone of the bot in order to enhance the experience of the customer.

b. **Generative Chatbots:**Unlike retrieval-based chatbots, generative chatbots are not based on pre-defined responses - they leverage seq2seq neural networks. This is constructed on the concept of machine translation, where the source code is converted from one language to another language. In the seq2seq approach, the input is changed into an output.

The first chatbot named **ELIZA** was designed and developed by Joseph Weizenbaum in 1966 that could imitate the language of a psychotherapist in only 200 lines of code. But as the technology gets more advance, we have come a long way from scripted chatbots to chatbots in Python today.

Chatbot in present Generation:

Today, we have smart Chatbots powered by Artificial Intelligence that utilize natural language processing (NLP) in order to understand the commands from humans (text and voice) and learn from experience. Chatbots have become a staple customer interaction utility for companies and brands that have an active online existence (website and social network platforms).

With the help of Python, Chatbots are considered a nifty utility as they facilitate rapid messaging between the brand and the customer. Let us think about Microsoft's Cortana, Amazon's Alexa, and Apple's Siri. Aren't these chatbots wonderful? It becomes quite

interesting to learn how to create a chatbot using the Python programming language.

Fundamentally, the chatbot utilizing Python is designed and programmed to take in the data we provide and then analyze it using the complex algorithms for Artificial Intelligence. It then delivers us either a written response or a verbal one. Since these bots can learn from experiences and behavior, they can respond to a large variety of queries and commands.

Although chatbot in Python has already started to rule the tech scenario at present, chatbots had handled approximately 85% of the customer-brand interactions by 2020 as per the prediction of Gartner.

In light of the increasing popularity and adoption of chatbots in the industry, we can increase the market value by learning how to create a chatbot in Python - among the most extensively utilized programming languages globally.

Syntax:

1. $ pip install chatterbot_corpus
2. $ pip install chatterbot

Syntax:

1. $ pip install chatterbot
2. $ pip install chatterbot_corpus

Syntax:

```
1. $ pip install --upgrade chatterbot_corpus   2.
$ pip install --upgrade chatterbot
```

my_chatbot.py

```
1. # importing the required modules
2. from chatterbot import ChatBot
3. from chatterbot.trainers import ListTrainer
```

Creating and Training the Chatbot:

The next step is to create a chatbot using an instance of the class "**ChatBot**" and train the bot in order to improve its performance. Training the bot ensures that it has enough knowledge, to begin with, particular replies to particular input statements.

Let us consider the following snippet of code for the same.

my_chatbot.py

```
1.      # creating a chatbot
2.      myBot = ChatBot(
3.      name = 'Sakura',
4.      read_only = True,
5.      logic_adapters = [
6.      'chatterbot.logic.MathematicalEvaluation',
7.      'chatterbot.logic.BestMatch'
8.      ]
9.      )
```

PROGRAM:

```
1.          # training the chatbot
2.          small_convo = [
```

```
3.          'Hi there!',
4.          'Hi',
5.          'How do you do?',
6.          'How are you?',
7.          'I\'m cool.',
8.          'Always cool.',
9.          'I\'m Okay',
10.         'Glad to hear that.',
11.         'I\'m fine',
12.         'I feel awesome',
13.         'Excellent, glad to hear that.',
14.         'Not so good',
15.         'Sorry to hear that.',
16.         'What\'s your name?',
17.         ' I\'m Sakura. Ask me a math question, please.'
18.         ]   19.
20.         math_convo_1 = [
21.         'Pythagorean theorem',
22.         'a squared plus b squared equals c squared.'
23.         ]
24.
25.         math_convo_2 = [
26.         'Law of Cosines',
27.         'c**2 = a**2 + b**2 - 2*a*b*cos(gamma)'
28.         ]
```

```
# using the ListTrainer class
list_trainee = ListTrainer(myBot)
for i in (small_convo, math_convo_1, math_convo_2):
list_trainee.train(i)
```

Output:

```
# starting a conversation
>>> print(myBot.get_response("Hi, there!"))
Hi
>>>     print(myBot.get_response("What's  your
name?"))
I'm Sakura. Ask me a math question, please.
>>> print(myBot.get_response("Do you know
Pythagorean theorem")) a squared plus b squared
equals c squared.
>>> print(myBot.get_response("Tell me the
formula of law of cosines")) c**2 =
a**2 + b**2 - 2*a*b*cos(gamma)
```

Training the Python Chatbot using a Corpus of Data: **my_chatbot.py**

```
from chatterbot.trainers import ChatterBotCorpusTrainer
corpus_trainee = ChatterBotCorpusTrainer(myBot)
corpus_trainee.train('chatterbot.corpus.english')
```

Complete Project Code

my_chatbot.py

```
# importing the required modules
from chatterbot import ChatBot
from chatterbot.trainers import ListTrainer
from chatterbot.trainers import ChatterBotCorpusTrainer

        # creating a chatbot
        myBot = ChatBot(
        name = 'Sakura',
        read_only = True,
        logic_adapters = [
        'chatterbot.logic.MathematicalEvaluation',
        'chatterbot.logic.BestMatch'
        ]
        )

    # training the chatbot
    small_convo = [
    'Hi there!',
    'Hi',
    'How do you do?',
    'How are you?',
    'I\'m cool.',
    'Always cool.',
    'I\'m Okay',
    'Glad to hear that.',
    'I\'m fine',
    'I feel awesome',
```

```
28.     'Excellent, glad to hear that.',
29.     'Not so good',
30.     'Sorry to hear that.',
31.     'What\'s your name?',
32.     ' I\'m Sakura. Ask me a math question, please.'
33.     ]
34.
35.     math_convo_1 = [
36.     'Pythagorean theorem',
37.     'a squared plus b squared equals c squared.'
38.     ]
39.
40.     math_convo_2 = [
41.     'Law of Cosines',
42.     'c**2 = a**2 + b**2 - 2*a*b*cos(gamma)'
43.     ]   44.
45.     # using the ListTrainer class
46.     list_trainee = ListTrainer(myBot)
47.     for i in (small_convo, math_convo_1, math_convo_2):
48.     list_trainee.train(i)
49.
50.  # using the ChatterBotCorpusTrainer class
51.  corpus_trainee = ChatterBotCorpusTrainer(myBot)
52.  corpus_trainee.train('chatterbot.corpus.english')
```

Natural Language Processing (NLP) in Python:

Natural Language Processing, often abbreviated as NLP, is the cornerstone of any intelligent chatbot. NLP is a subfield of AI that focuses on the interaction between humans and computers using natural language. The ultimate objective of NLP is to read, decipher, understand, and make sense of human language in a valuable way.

In the realm of chatbots, NLP comes into play to enable bots to understand and respond to user queries in human language. But how does Python contribute to NLP? Well, Python, with its extensive array of libraries like NLTK (Natural Language Toolkit), SpaCy, and TextBlob, makes NLP tasks much more manageable. These libraries contain packages to perform tasks from basic text processing to more complex language understanding tasks.

For instance, Python's NLTK library helps with everything from splitting sentences and words to recognizing parts of speech (POS). On the other hand, SpaCy excels in tasks that require deep learning, like understanding sentence context and parsing.

In summary, understanding NLP and how it is implemented in Python is crucial in your journey to creating a Python AI chatbot. It equips you with the tools to ensure that your chatbot can understand and respond to your users in a way that is both efficient and human-like.

Types of Chatbots:

- **Rule-Based Chatbots:** These chatbots operate based on pre-determined rules on which they are initially programmed. They are best for scenarios that require simple query-response conversations. Their downside is they can't handle complex queries because their intelligence is limited to their programmed rules.
- **Self-Learning Chatbots:** Powered by Machine Learning and Artificial Intelligence, these chatbots learn from their mistakes and the inputs they receive. The more data they are exposed to, the better their responses

become. These chatbots are suited for complex tasks but their implementation is more challenging.

- **Hybrid Chatbots:** As the name suggests, these chatbots combine the best of both worlds. They operate on pre-defined rules for simple queries and use machine learning capabilities for complex queries. Hybrid chatbots offer flexibility and can adapt to a wide array of situations, making them a popular choice.

Let‘s consider a practical scenario. Suppose you run an ecommerce website. A rule-based chatbot might suffice if you want to answer FAQs. But, if you want the chatbot to recommend products based on customers’ past purchases or preferences, a self-learning or hybrid chatbot would be more suitable.

Understanding the types of chatbots and their uses helps you determine the best fit for your needs. The choice ultimately depends on your chatbot’s purpose, the complexity of tasks it needs to perform, and the resources at your disposal.

NLTK library to preprocess the data:

```
import nltk from nltk.stem import
WordNetLemmatizer from nltk.corpus
import stopwords import string

# Download NLTK data
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

# Load data with open('data.txt', 'r',
encoding='utf-8') as f:     raw_data = f.read()

# Preprocess data
def
preprocess(data):
```

```
# Tokenize data
tokens =
nltk.word_tokenize(d
ata)

    # Lowercase all words
    tokens = [word.lower() for word in tokens]

    # Remove stopwords and punctuation
stop_words = set(stopwords.words('english'))
tokens = [word for word in tokens if word not
in stop_words and word not in
string.punctuation]

    # Lemmatize words
    lemmatizer = WordNetLemmatizer()      tokens
= [lemmatizer.lemmatize(word) for word in
tokens]
    return tokens

# Preprocess data
processed_data = [preprocess(qa) for qa in
raw_data.split('\n')]
```

Train a machine learning model:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text
import Tokenizer from
tensorflow.keras.preprocessing.sequence
import pad_sequences

# Set parameters
vocab_size = 5000
embedding_dim = 64
```

```
max_length = 100
trunc_type='post'
padding_type='post'
oov_tok = "<OOV>"
training_size =
len(processed_data)

# Create tokenizer tokenizer =
Tokenizer(num_words=vocab_size,
oov_token=oov_tok)
tokenizer.fit_on_texts(processed_data)
word_index = tokenizer.word_index

# Create sequences sequences =
tokenizer.texts_to_sequences(processed_dat
a) padded_sequences =
pad_sequences(sequences,
maxlen=max_length, padding=padding_type,
truncating=trunc_type)
# Create training data training_data =
padded_sequences[:training_size] training_labels
= padded_sequences[:training_size]

# Build model model = tf.keras.Sequential([
tf.keras.layers.Embedding(vocab_size,
embedding_dim, input_length=max_length),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Conv1D(64, 5,
activation='relu'),
tf.keras.layers.MaxPooling1D(pool_size=4),
tf.keras.layers.LSTM(64),
tf.keras.layers.Dense(64, activation='relu'),
```

```
tf.keras.layers.Dense(vocab_size,
activation='softmax')
])

# Compile model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train model num_epochs = 50 history = model.fit(training_data, training_labels, epochs=num_epochs, verbose=2)
```

Build the chatbot interface:

```
# Define function to predict answer def predict_answer(model, tokenizer, question):
    # Preprocess question question = preprocess(question) # Convert question to sequence sequence = tokenizer.texts_to_sequences([question])
    # Pad sequence padded_sequence = pad_sequences(sequence, maxlen=max_length, padding=padding_type, truncating=trunc_type) # Predict answer
pred = model.predict(padded_sequence)[0]
# Get index of highest probability idx = np.argmax(pred)
    # Get answer
    answer = tokenizer.index_word[idx]
return answer
```

```
# Start chatbot
while True:
    question = input('You: ')     answer =
predict_answer(model, tokenizer, question)
    print('Chatbot:', answer)
```

CONCLUSION:

We started by gathering and preprocessing data, then we built a neural network model using the Keras Sequential API. We then created a simple command-line interface for the chatbot and tested it with some example conversations.

This is just a basic example of a chatbot, and there are many ways to improve it. With more advanced techniques and tools, you can build chatbots that can understand natural language, generate human-like responses, and even learn from user interactions to improve over time.

Building a chatbot using Python code can be a simple process, as long as you have the right tools and knowledge. In this article, I've provided you with a basic guide to get started. Once you have your chatbot up and running, it'll be able to handle simple tasks and conversations. If you want to take your chatbot to the next level, you can consider adding more features or connecting it to other services.

This blog was a hands-on introduction to building a very simple rulebased chatbot in python. We only worked with 2 intents in this

tutorial for simplicity. You can easily expand the functionality of this chatbot by adding more keywords, intents and responses.

As we saw, building a rule-based chatbot is a laborious process. In a business environment, a chatbot could be required to have a lot more intent depending on the tasks it is supposed to undertake.

In such a situation, rule-based chatbots become very impractical as maintaining a rule base would become extremely complex. In addition, the chatbot would severely be limited in terms of its conversational capabilities as it is near impossible to describe exactly how a user will interact with the bot.

AI-based Chatbots are a much more practical solution for real-world scenarios. In the next blog in the series, we'll be looking at how to build a simple AI-based Chatbot in Python.