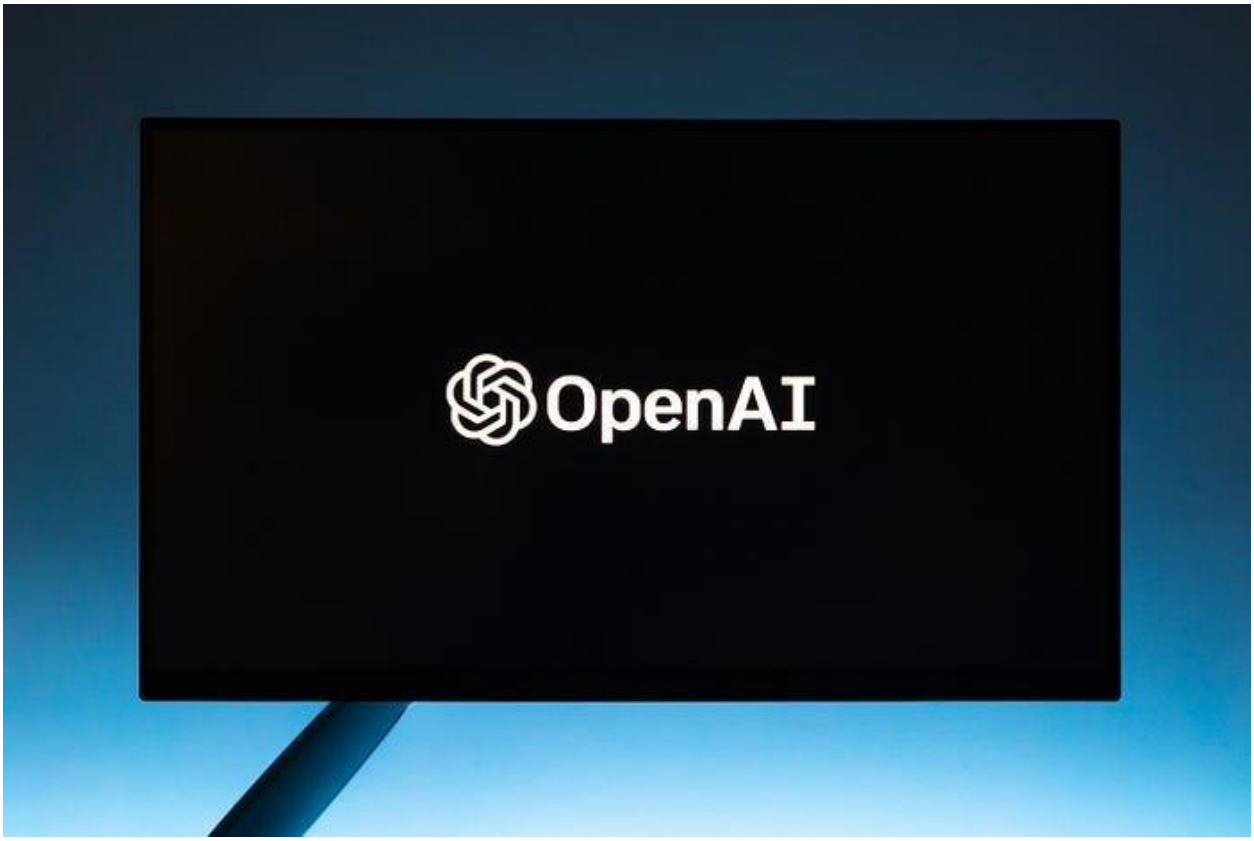
CREATE A CHATBOT IN PYTHON

Name: KEERTHANA KUMARI. V
Reg.No: 513521106013
Department: ECE
Year: III
NM ID: au513521106013
Email:keerthanakumari424@gmail.com



INTRODUCTION:

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.

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()
```

HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>GenAI-Bot</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></s
cri pt>    <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>
```

Backend Logic:

```
#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'}
```

CONCLUSION:

This blog was a hands-on introduction to building a very simple rule-based 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.