International Journal for Multidisciplinary Research (IJFMR)

E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Design and Implementation of a Chatbot in Python

Samiksha Shrivastava

Student, Kalinga University, Raipur (C.G.)

Abstract

Chatbots have become increasingly popular in recent years, with applications spanning customer support, virtual assistants, and more. This research paper explores the design and implementation of a chatbot using Python. We delve into various aspects of chatbot development, including natural language processing (NLP), dialogue management, and user interface integration. The paper discusses key Python libraries and tools for building a chatbot and presents a case study demonstrating the creation of a simple chatbot.

Keywords: Introduction, Chatbot Architecture, Chatbot development process, Case study: Building a chatbot application, Evaluation and Performance metrics, Challenges and future directions, Conclusion

1. Introduction

Chatbots are automated conversational agents that use artificial intelligence (AI) and natural language processing (NLP) to engage in text or voice-based conversations with users. They find applications in customer service, information retrieval, task automation, and entertainment. This paper focuses on the development of a chatbot in Python, one of the most popular programming languages for AI and NLP tasks.

2. Chatbot Architecture

2.1. Natural Language Processing (NLP)

NLP plays a crucial role in chatbot development. Python offers various NLP libraries and tools such as NLTK, spaCy, and the Hugging Face Transformers library for tasks like tokenization, part-of-speech tagging, and sentiment analysis

2.2. Dialogue Management
Effective dialogue management is essential for a chatbot to understand and generate contextually relevant responses. This can be achieved using rule-based systems, finite-state machines, or more advanced techniques like reinforcement learning for dynamic conversations.

2.3. User Interface Integration
Chatbots need a user-friendly interface for interaction. Python has several libraries like Tkinter, Flask, and Django to create web-based or desktop chatbot interfaces.

3. Chatbot Development Process
3.1. Data Collection and Preprocessing
To train and fine-tune a chatbot's NLP models, you need a dataset of conversations. Tools like web scraping, API integration, or dataset creation can be used for data collection. Preprocessing includes cleaning, tokenization, and data augmentation.

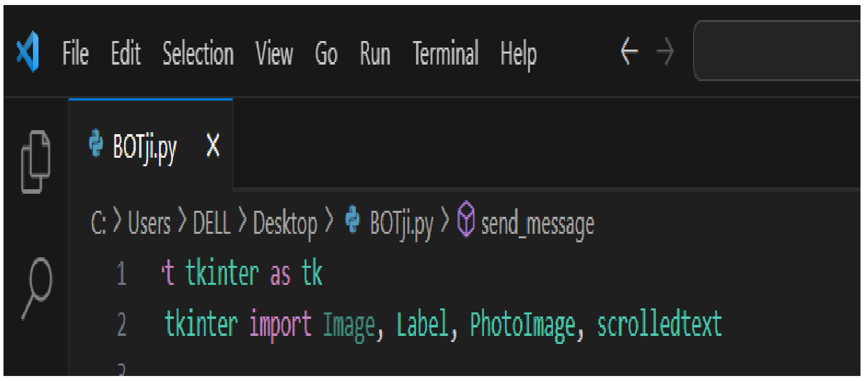
3.2. Model Selection and Training
Selecting the right NLP model is crucial. Popular choices include rule-based models, sequence-to-sequence models (e.g., LSTM or Transformers), or pre-trained language models like GPT-3. Fine-tuning is often necessary for domain-specific chatbots.

3.3. Dialogue Management
Implementing dialogue management involves creating a state machine or a set of rules to guide the conversation flow. Reinforcement learning techniques can optimize the dialogue policy over time.

3.4. User Interface Integration
Integrate the chatbot into a user interface, which can be a web application, a mobile app, or a standalone desktop application. Python libraries like Flask and HTML/CSS for web-based interfaces are commonly used.

4. Case Study: Building a Python Chatbot
In this section, we provide a step-by-step case study demonstrating the creation of a simple chatbot using Python. We cover data collection, preprocessing, model selection, and user interface integration.
Basic implementation of a chatbot using the Tkinter library in Python. It includes a simple user interface for chatting with the chatbot. Here are the steps to build this chatbot:

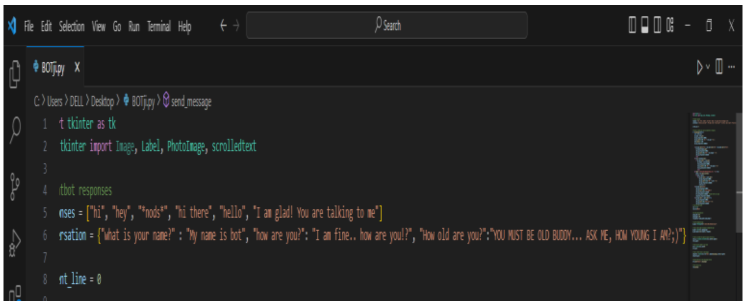
1. Import the necessary libraries:
- `tkinter` for creating the GUI.
- `scrolled text` for displaying the chat log.


File Edit Selection View Go Run Terminal Help
BOTji.py X
C: › Users › DELL › Desktop › BOTji.py › send_message

```
t tkinter as tk
tkinter import Image, Label, PhotoImage, scrolledtext
```

2. Define the chatbot responses:

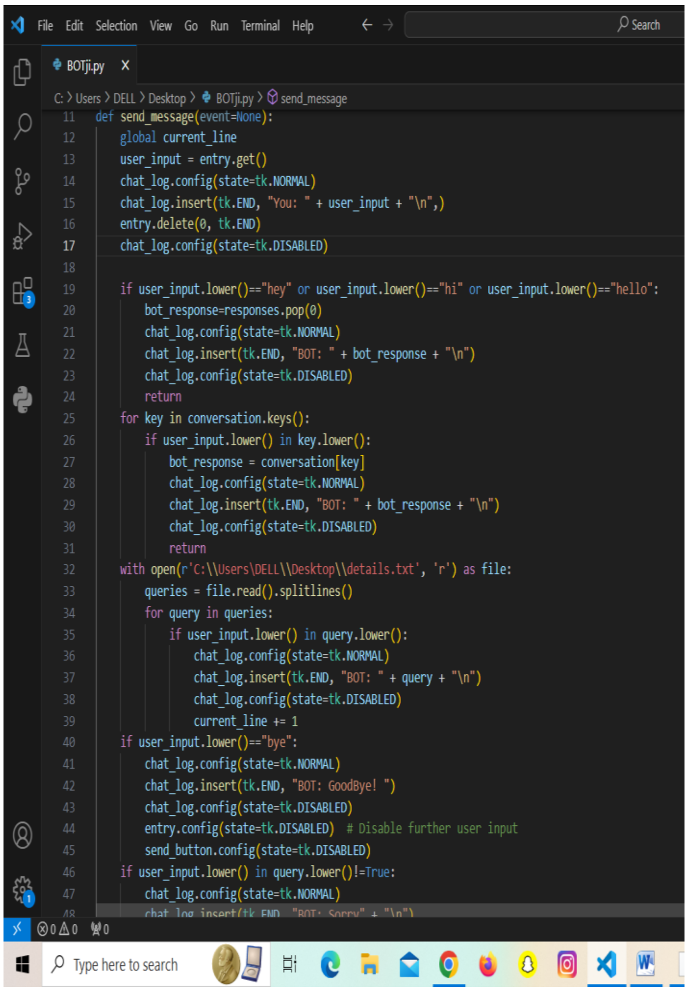
- Create a list of responses and a dictionary of conversation triggers and responses.


File Edit Selection View Go Run Terminal Help
Search
BOTji.py X
C: › Users › DELL › Desktop › BOTji.py › send_message

```
t tkinter as tk
tkinter import Image, Label, PhotoImage, scrolledtext

tbot responses
nses = ["hi", "hey", "*nods*", "hi there", "hello", "I am glad! You are talking to me"]
rsation = {"What is your name?" : "My name is bot", "how are you?": "I am fine.. how are you!?", "How old are you?":"YOU MUST BE OLD BUDDY... ASK ME, HOW YOUNG I AM?;)"}

nt_line = 0
```

3. Create a function to handle user input and generate bot responses:

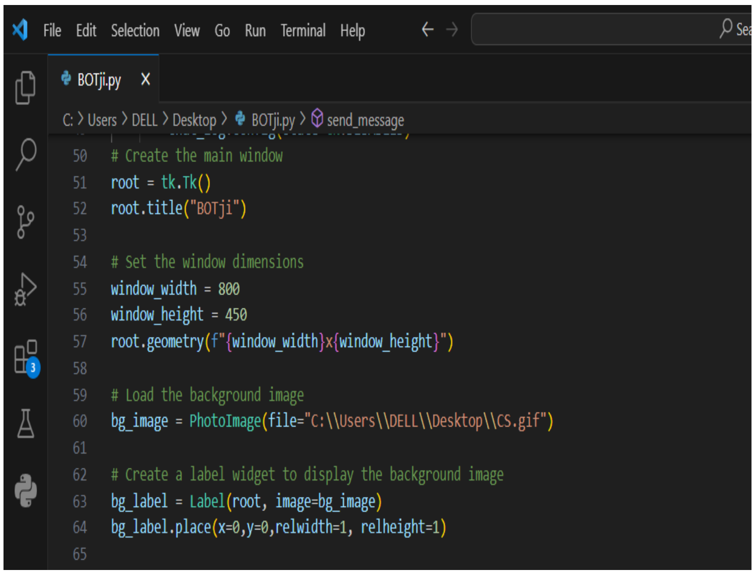
- The `send_message` function is called when the user sends a message.
- Get the user's input from the entry field.
- Display the user's message in the chat log.
- Check if the user input matches predefined greetings and respond accordingly.
- Check if the user input matches any conversation triggers and respond with the appropriate response.
- Check if the user input matches any queries from a file and respond with the corresponding query.
- If the user says "bye," end the conversation and disable further input.
- If none of the above conditions are met, respond with a default sorry message.



```
def send_message(event=None):
    global current_line
    user_input = entry.get()
    chat_log.config(state=tk.NORMAL)
    chat_log.insert(tk.END, "You: " + user_input + "\n",)
    entry.delete(0, tk.END)
    chat_log.config(state=tk.DISABLED)

    if user_input.lower()=="hey" or user_input.lower()=="hi" or user_input.lower()=="hello":
        bot_response=responses.pop(0)
        chat_log.config(state=tk.NORMAL)
        chat_log.insert(tk.END, "BOT: " + bot_response + "\n")
        chat_log.config(state=tk.DISABLED)
        return
    for key in conversation.keys():
        if user_input.lower() in key.lower():
            bot_response = conversation[key]
            chat_log.config(state=tk.NORMAL)
            chat_log.insert(tk.END, "BOT: " + bot_response + "\n")
            chat_log.config(state=tk.DISABLED)
            return
    with open(r'C:\\Users\DELL\\Desktop\\details.txt', 'r') as file:
        queries = file.read().splitlines()
        for query in queries:
            if user_input.lower() in query.lower():
                chat_log.config(state=tk.NORMAL)
                chat_log.insert(tk.END, "BOT: " + query + "\n")
                chat_log.config(state=tk.DISABLED)
                current_line += 1
    if user_input.lower()=="bye":
        chat_log.config(state=tk.NORMAL)
        chat_log.insert(tk.END, "BOT: GoodBye! ")
        chat_log.config(state=tk.DISABLED)
        entry.config(state=tk.DISABLED)  # Disable further user input
        send_button.config(state=tk.DISABLED)
    if user_input.lower() in query.lower()!=True:
        chat_log.config(state=tk.NORMAL)
        chat_log.insert(tk.END, "BOT: Sorry" + "\n")
```

4. Create the main window:

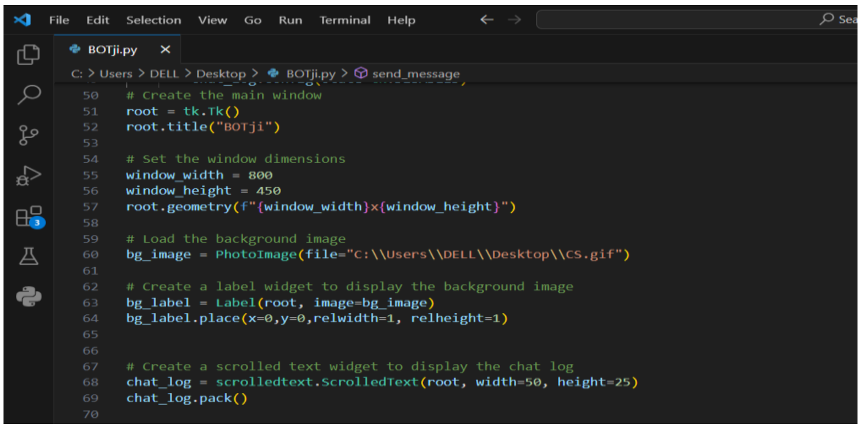
- Using tk.Tk(), create the main window and set the window title and window dimensions.
- Load a background image and display it using a `Label`.



```
# Create the main window
root = tk.Tk()
root.title("BOTji")

# Set the window dimensions
window_width = 800
window_height = 450
root.geometry(f"{window_width}x{window_height}")

# Load the background image
bg_image = PhotoImage(file="C:\\Users\\DELL\\Desktop\\CS.gif")

# Create a label widget to display the background image
bg_label = Label(root, image=bg_image)
bg_label.place(x=0,y=0,relwidth=1, relheight=1)
```

5. Create a scrolled text widget for the chat log:

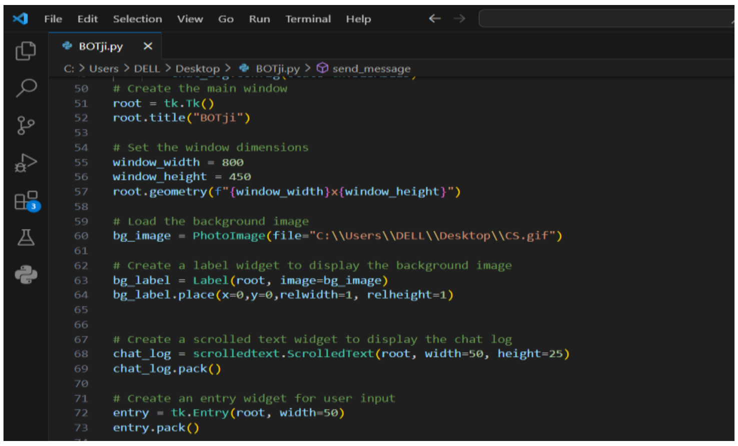
- Create a scrolled text widget to display the chat conversation.


File Edit Selection View Go Run Terminal Help
BOTji.py
C: > Users > DELL > Desktop > BOTji.py > send_message

```python
# Create the main window
root = tk.Tk()
root.title("BOTji")

# Set the window dimensions
window_width = 800
window_height = 450
root.geometry(f"{window_width}x{window_height}")

# Load the background image
bg_image = PhotoImage(file="C:\\Users\\DELL\\Desktop\\CS.gif")

# Create a label widget to display the background image
bg_label = Label(root, image=bg_image)
bg_label.place(x=0,y=0,relwidth=1, relheight=1)


# Create a scrolled text widget to display the chat log
chat_log = scrolledtext.ScrolledText(root, width=50, height=25)
chat_log.pack()

```

6. Create an entry widget for user input:

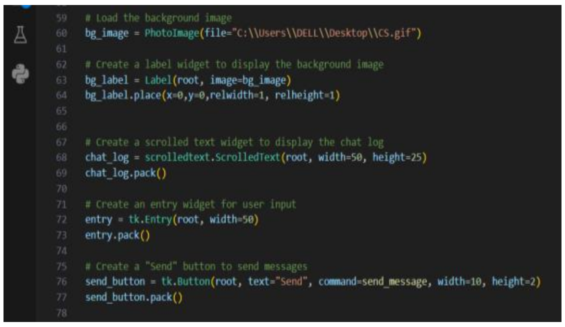
- Create an entry widget where the user can type messages.


File Edit Selection View Go Run Terminal Help
BOTji.py
C: > Users > DELL > Desktop > BOTji.py > send_message

```python
# Create the main window
root = tk.Tk()
root.title("BOTji")

# Set the window dimensions
window_width = 800
window_height = 450
root.geometry(f"{window_width}x{window_height}")

# Load the background image
bg_image = PhotoImage(file="C:\\Users\\DELL\\Desktop\\CS.gif")

# Create a label widget to display the background image
bg_label = Label(root, image=bg_image)
bg_label.place(x=0,y=0,relwidth=1, relheight=1)


# Create a scrolled text widget to display the chat log
chat_log = scrolledtext.ScrolledText(root, width=50, height=25)
chat_log.pack()

# Create an entry widget for user input
entry = tk.Entry(root, width=50)
entry.pack()
```

7. Create a "Send" button:

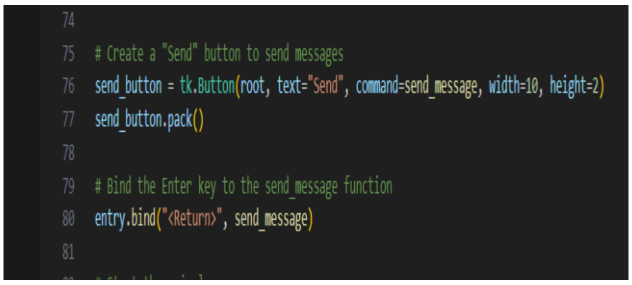
- Create a button labeled "Send" that triggers the `send_message` function when clicked.



```python
# Load the background image
bg_image = PhotoImage(file="C:\\Users\\DELL\\Desktop\\CS.gif")

# Create a label widget to display the background image
bg_label = Label(root, image=bg_image)
bg_label.place(x=0,y=0,relwidth=1, relheight=1)


# Create a scrolled text widget to display the chat log
chat_log = scrolledtext.ScrolledText(root, width=50, height=25)
chat_log.pack()

# Create an entry widget for user input
entry = tk.Entry(root, width=50)
entry.pack()

# Create a "Send" button to send messages
send_button = tk.Button(root, text="Send", command=send_message, width=10, height=2)
send_button.pack()

```

8. Bind the Enter key to the "Send" button:

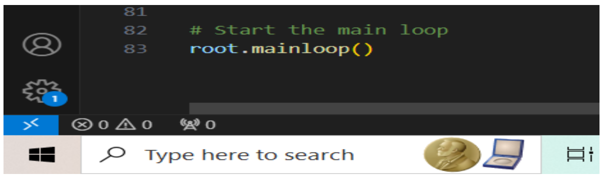
- Bind the Enter key to the `send_message` function so that users can send messages by pressing Enter.



```
# Create a "Send" button to send messages
send_button = tk.Button(root, text="Send", command=send_message, width=10, height=2)
send_button.pack()

# Bind the Enter key to the send_message function
entry.bind("<Return>", send_message)
```

9. Start the main loop:

- Start the main GUI event loop using `root.mainloop()`.



```
# Start the main loop
root.mainloop()
```

NOTE: Make sure you have the necessary image and text files in the specified paths for the chatbot to work correctly. Additionally, you may want to enhance the chatbot's capabilities and responses as per your requirements.

Combined INPUT CODE :

```
import tkinter as tk
from tkinter import Image, Label, PhotoImage, scrolledtext

# Chatbot responses
responses = ["hi", "hey", "*nods*", "hi there", "hello", "I am glad! You are talking to me"]
conversation = {"what is your name?" : "My name is bot",  "how are you?": "I am fine.. how are you!?", "How old are you?":"YOU MUST BE OLD BUDDY... ASK ME, HOW YOUNG I AM?;)"}

current_line = 0
def send_message(event=None):
   global current_line
   user_input = entry.get()
   chat_log.config(state=tk.NORMAL)
   chat_log.insert(tk.END, "You: " + user_input + "\n",)
   entry.delete(0, tk.END)
   chat_log.config(state=tk.DISABLED)

   if user_input.lower()=="hey" or user_input.lower()=="hi" or user_input.lower()=="hello":
      bot_response=responses.pop(0)
      chat_log.config(state=tk.NORMAL)
      chat_log.insert(tk.END, "BOT: " + bot_response + "\n")
      chat_log.config(state=tk.DISABLED)
      return
```

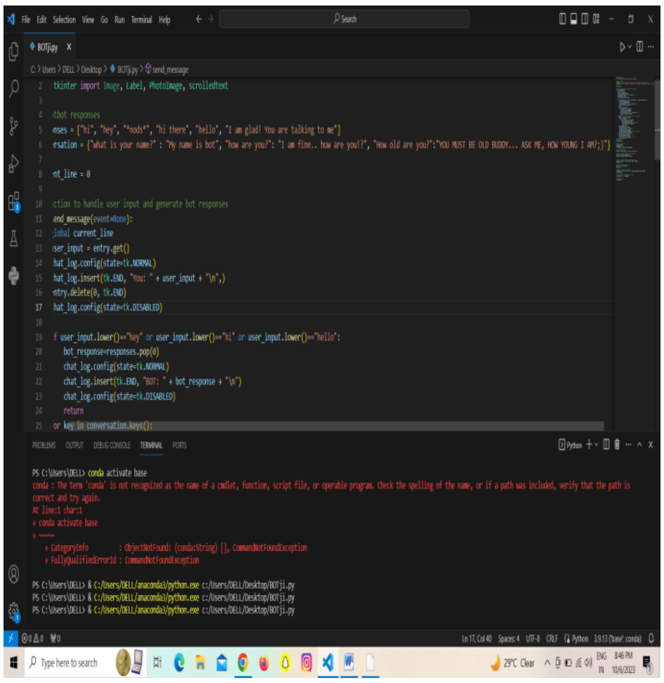
```
    for key in conversation.keys():
        if user_input.lower() in key.lower():
            bot_response = conversation[key]
            chat_log.config(state=tk.NORMAL)
            chat_log.insert(tk.END, "BOT: " + bot_response + "\n")
            chat_log.config(state=tk.DISABLED)
            return
    with open(r'C:\\Users\DELL\\Desktop\\details.txt', 'r') as file:
        queries = file.read().splitlines()
        for query in queries:
            if user_input.lower() in query.lower():
                chat_log.config(state=tk.NORMAL)
                chat_log.insert(tk.END, "BOT: " + query + "\n")
                chat_log.config(state=tk.DISABLED)
                current_line += 1
    if user_input.lower()=="bye":
        chat_log.config(state=tk.NORMAL)
        chat_log.insert(tk.END, "BOT: GoodBye! ")
        chat_log.config(state=tk.DISABLED)
        entry.config(state=tk.DISABLED)  # Disable further user input
        send_button.config(state=tk.DISABLED)
    if user_input.lower() in query.lower()!=True:
        chat_log.config(state=tk.NORMAL)
        chat_log.insert(tk.END, "BOT: Sorry" + "\n")
        chat_log.config(state=tk.DISABLED)
# Create the main window
root = tk.Tk()
root.title("BOTji")

# Set the window dimensions
window_width = 800
window_height = 450
root.geometry(f"{window_width}x{window_height}")

# Load the background image
bg_image = PhotoImage(file="C:\\Users\\DELL\\Desktop\\CS.gif")

# Create a label widget to display the background image
bg_label = Label(root, image=bg_image)
bg_label.place(x=0,y=0,relwidth=1, relheight=1)


# Create a scrolled text widget to display the chat log
```

```
chat_log = scrolledtext.ScrolledText(root, width=50, height=25)
chat_log.pack()

# Create an entry widget for user input
entry = tk.Entry(root, width=50)
entry.pack()

# Create a "Send" button to send messages
send_button = tk.Button(root, text="Send", command=send_message, width=10, height=2)
send_button.pack()

# Bind the Enter key to the send_message function
entry.bind("<Return>", send_message)

# Start the main loop
root.mainloop()
```

5. Evaluation and Performance Metrics

Evaluating a chatbot's performance is essential to measure its effectiveness. Metrics like accuracy, response time, user satisfaction, and conversation quality can be used. Conducting user surveys and A/B testing can help gather feedback.

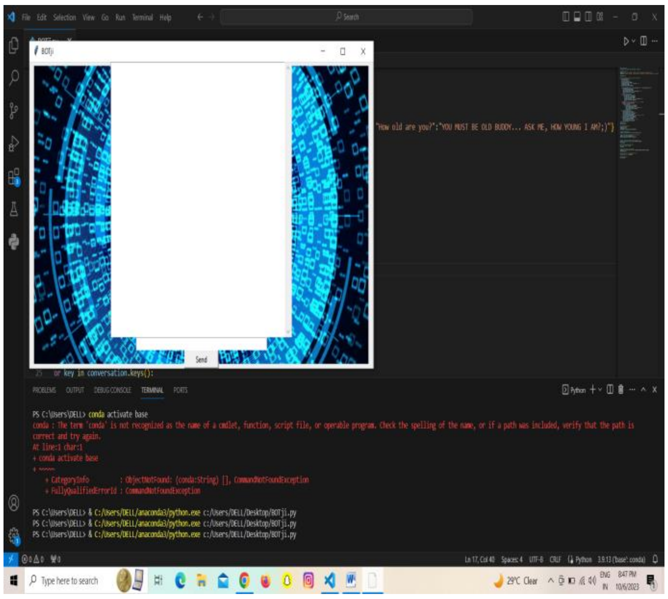
Here’s the final result :



File Edit Selection View Go Run Terminal Help
Search
BOTji.py
C: > Users > DELL > Desktop > BOTji.py > send_message
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
Ln 17, Col 40 Spaces: 4 UTF-8 CRLF Python 3.9.13 ('base': conda)
Type here to search
29°C Clear
ENG IN 8:46 PM 10/6/2023

```
tkinter import Image, Label, PhotoImage, scrolledtext

tbot responses
nses = ["hi", "hey", "*nods*", "hi there", "hello", "I am glad! You are talking to me"]
rsation = {"what is your name?" : "My name is bot", "how are you?": "I am fine.. how are you!?", "How old are you?":"YOU MUST BE OLD BUDDY... ASK ME, HOW YOUNG I AM?;)"}

nt_line = 0

ction to handle user input and generate bot responses
end_message(event=None):
lobal current_line
ser_input = entry.get()
hat_log.config(state=tk.NORMAL)
hat_log.insert(tk.END, "You: " + user_input + "\n",)
ntry.delete(0, tk.END)
hat_log.config(state=tk.DISABLED)

f user_input.lower()=="hey" or user_input.lower()=="hi" or user_input.lower()=="hello":
   bot_response=responses.pop(0)
   chat_log.config(state=tk.NORMAL)
   chat_log.insert(tk.END, "BOT: " + bot_response + "\n")
   chat_log.config(state=tk.DISABLED)
   return
or key in conversation.keys():
```

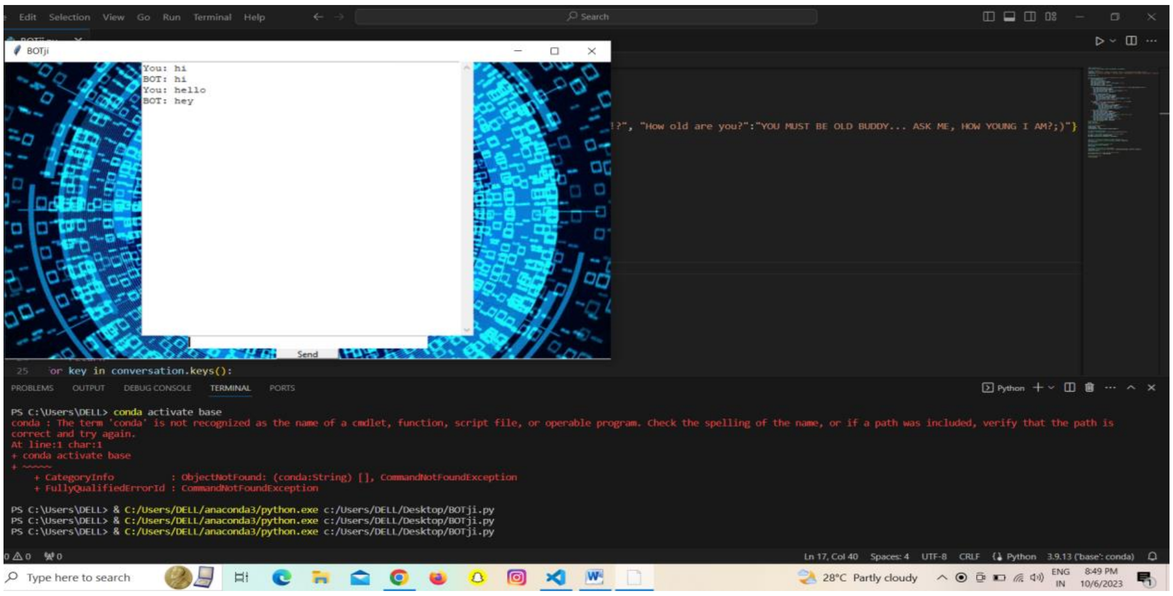
```
PS C:\Users\DELL> conda activate base
conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is
correct and try again.
At line:1 char:1
+ conda activate base
+ ~~~~~
    + CategoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
```



```
"How old are you?":"YOU MUST BE OLD BUDDY... ASK ME, HOW YOUNG I AM?;)"}
```

```
or key in conversation.keys():
```

```
PS C:\Users\DELL> conda activate base
conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is
correct and try again.
At line:1 char:1
+ conda activate base
+ ~~~~~
    + CategoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
```

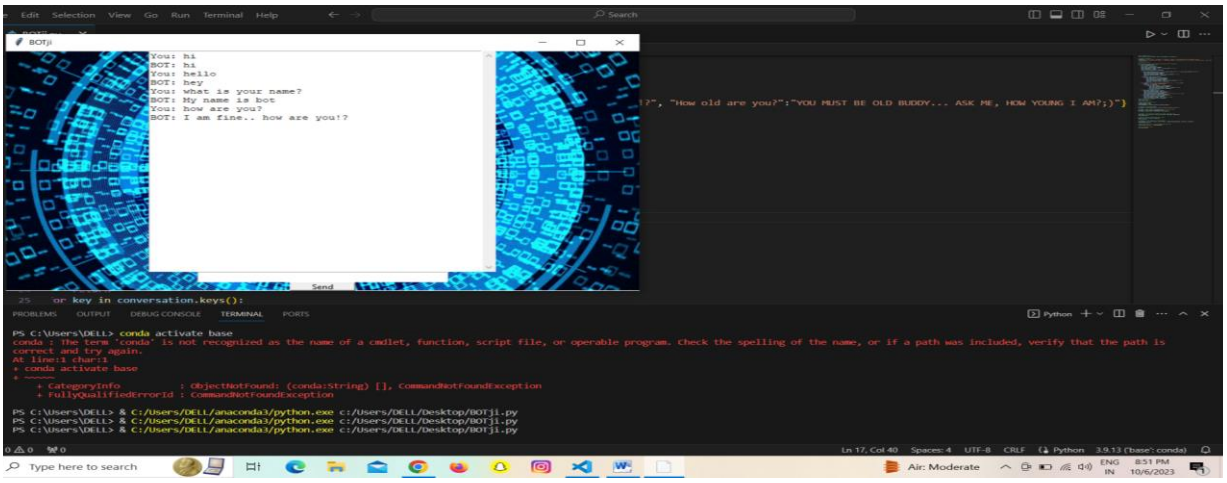


```
You: hi
BOT: hi
You: hello
BOT: hey
```

```
?", "How old are you?":"YOU MUST BE OLD BUDDY... ASK ME, HOW YOUNG I AM?;)"}
```

```
or key in conversation.keys():
```

```
PS C:\Users\DELL> conda activate base
conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is
correct and try again.
At line:1 char:1
+ conda activate base
+ ~~~~~
    + CategoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
```

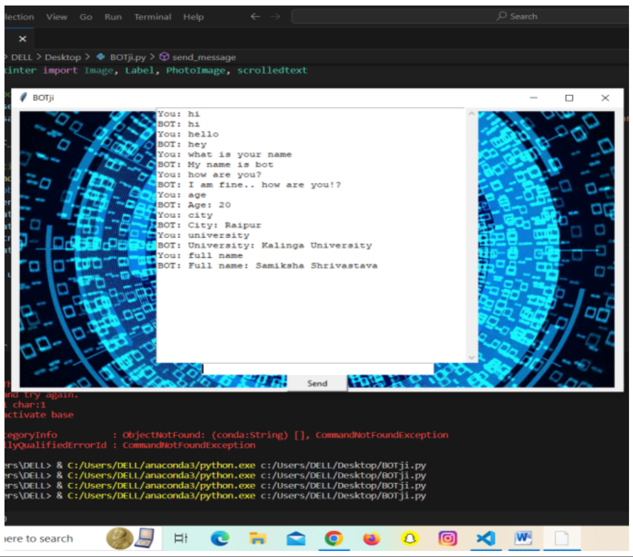


```
You: hi
BOT: hi
You: hello
BOT: hey
You: what is your name?
BOT: My name is bot
You: how are you?
BOT: I am fine.. how are you!?
```

```
?", "How old are you?":"YOU MUST BE OLD BUDDY... ASK ME, HOW YOUNG I AM?;)"}
```

```
or key in conversation.keys():
```

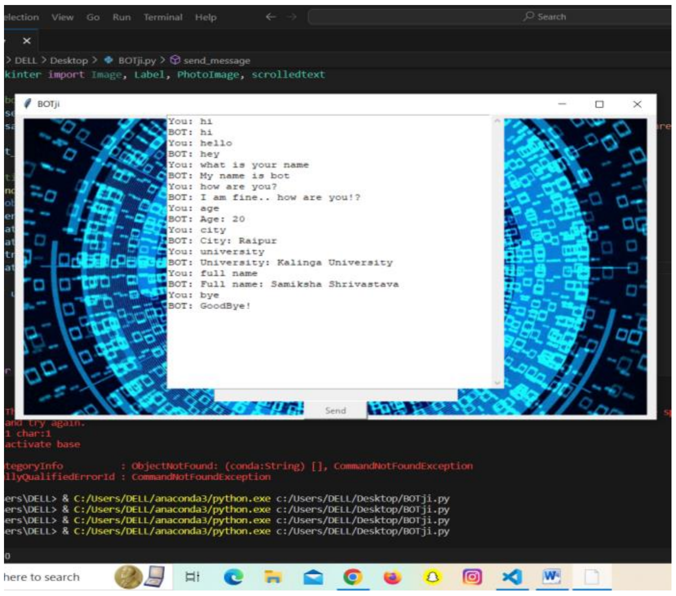
```
PS C:\Users\DELL> conda activate base
conda : The term 'conda' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is
correct and try again.
At line:1 char:1
+ conda activate base
+ ~~~~~
    + CategoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
PS C:\Users\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
```


lection View Go Run Terminal Help
Search
DELL > Desktop > BOTji.py > send_message
BOTji
Send
ere to search

```
kinter import Image, Label, PhotoImage, scrolledtext
```

```
You: hi
BOT: hi
You: hello
BOT: hey
You: what is your name
BOT: My name is bot
You: how are you?
BOT: I am fine.. how are you!?
You: age
BOT: Age: 20
You: city
BOT: City: Raipur
You: university
BOT: University: Kalinga University
You: full name
BOT: Full name: Samiksha Shrivastava
```

```
nd try again.
 char:1
ctivate base

tegoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
llyQualifiedErrorId : CommandNotFoundException

rs\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
rs\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
rs\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
rs\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
```


election View Go Run Terminal Help
Search
DELL > Desktop > BOTji.py > send_message
BOTji
Send
here to search

```
kinter import Image, Label, PhotoImage, scrolledtext
```

```
You: hi
BOT: hi
You: hello
BOT: hey
You: what is your name
BOT: My name is bot
You: how are you?
BOT: I am fine.. how are you!?
You: age
BOT: Age: 20
You: city
BOT: City: Raipur
You: university
BOT: University: Kalinga University
You: full name
BOT: Full name: Samiksha Shrivastava
You: bye
BOT: GoodBye!
```

```
and try again.
1 char:1
activate base

tegoryInfo          : ObjectNotFound: (conda:String) [], CommandNotFoundException
llyQualifiedErrorId : CommandNotFoundException

ers\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
ers\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
ers\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
ers\DELL> & C:/Users/DELL/anaconda3/python.exe c:/Users/DELL/Desktop/BOTji.py
```

6. Challenges and Future Directions

Chatbot development in Python faces several challenges, including handling multi-turn conversations, improving context awareness, and ethical concerns surrounding AI chatbots. Future directions may involve more advanced NLP models, multi-modal interactions (text and voice), and addressing biases in chatbot responses.

Chatbots have a wide range of potential future uses across various industries and domains. Here are some of the potential future use cases of chatbots:

1. Customer Support and Service:

- Chatbots can provide 24/7 customer support, handling routine queries and issues.
- They can assist customers with product information, troubleshooting, and order tracking.

2. E-commerce:

- Chatbots can help users find products, provide recommendations, and facilitate the shopping process.
- They can assist with order placement, payment processing, and returns.

3. Healthcare:

- Chatbots can offer medical advice, symptom assessment, and medication reminders.
- They can schedule appointments, assist with insurance queries, and provide health-related information.

4. Finance and Banking:
- Chatbots can help users check account balances, transfer funds, and manage investments.
- They can provide financial advice and answer questions about banking services.

5. HR and Recruitment:
- Chatbots can assist with job applications, answer questions about company policies, and schedule interviews.
- They can automate the initial screening of job candidates.

6. Education:
- Chatbots can act as virtual tutors, answering students' questions and providing explanations.
- They can assist with homework, offer language learning support, and provide study resources.

7. Travel and Tourism:
- Chatbots can provide travel recommendations, weather updates, and local information.

8. Legal Services:
- Chatbots can offer basic legal advice and document preparation services.
- They can guide users through legal processes and answer legal questions.

9. Entertainment and Gaming:
- Chatbots can enhance gaming experiences by providing in-game assistance and hints.
- They can engage users in interactive storytelling and role-playing.

10. Government and Public Services:
- Chatbots can assist citizens with government services, such as renewing licenses and accessing public information.
- They can answer frequently asked questions about government programs.

11. Language Translation:
- Chatbots can translate languages in real-time, enabling cross-cultural communication.
- They can be used for multilingual customer support.

12. Mental Health and Well-being:
- Chatbots can provide emotional support and resources for mental health.
- They can offer relaxation techniques and mindfulness exercises.

13. IoT (Internet of Things):
- Chatbots can control and manage smart home devices and appliances.
- They can provide status updates and receive voice commands.

14. Automated Content Creation:
- Chatbots can generate content like news articles, reports, and marketing materials.
- They can assist writers and content creators with ideas and drafts.

15. Personal Assistants:
- Chatbots can act as virtual personal assistants, managing schedules, setting reminders, and answering questions.

16. Accessibility and Inclusion:
- Chatbots can assist individuals with disabilities by providing text-to-speech and speech-to-text capabilities.

The future of chatbots is likely to involve increased integration with artificial intelligence and machine learning technologies, making them more intelligent, context-aware, and capable of handling complex tasks. As technology continues to advance, chatbots will play an increasingly significant role in improving efficiency and enhancing user experiences across a wide range of industries.

7. Conclusion

This research paper explored the design and implementation of a chatbot in Python, highlighting key components such as NLP, dialogue management, and user interface integration. Python's rich ecosystem of libraries and tools makes it a suitable choice for chatbot development. With further advancements in AI and NLP, chatbots are expected to become more sophisticated and capable of handling a wider range of tasks.

References

1. Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd ed.). O'Reilly Media.
2. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention Is All You Need. Advances in Neural Information Processing Systems (NeurIPS).
3. Jurafsky, D., Martin, J. H. (2020). "Speech and Language Processing."