COURSE : ARTIFICIAL INTELLIGENCE
TITLE: CREATE A CHATBOT IN PYTHON

Phase 5 submission document

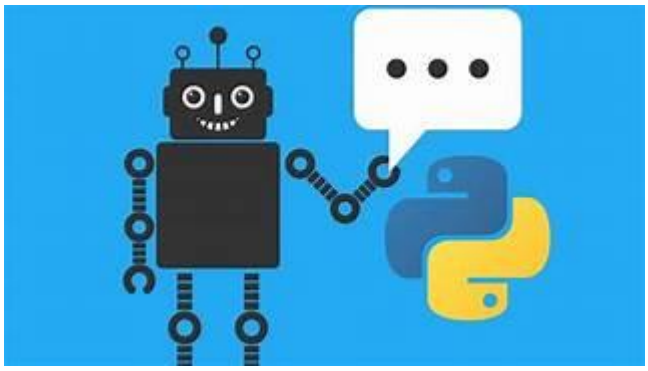
Phase 5: Project Documentation & Submission

Topic: In this section we will document the complete project and prepare it for submission.

TEAM MEMBERS:
SRIVALSAN. S (au311521106098)
VALLARASU . S (au3115211060107)

INTODUCTION:

In an increasingly digital world, the demand for intelligent and interactive conversational agents has surged. Chatbots, driven by the power of natural language processing and artificial intelligence, have become indispensable tools for businesses and individuals alike. This project aims to address this growing need by developing a Python-based chatbot capable of engaging users in text-based conversations on a wide array of topics. This chatbot is designed to provide information, answer questions, and offer assistance in a manner that mimics human interaction, making it a valuable asset for customer support, information retrieval, and general conversational purposes. By harnessing the capabilities of modern AI and natural language understanding, this chatbot will not only streamline communication but also enhance the user experience, ultimately becoming a valuable addition to any platform or service.



Here's an overall introduction to creating a chatbot using Python:

1. Define the Purpose:

First, you need to clearly define the purpose and goals of your chatbot. What do you want it to achieve? Who is your target audience? Understanding your objectives will guide the design and development process.

2. Choose a Development Approach:

There are several approaches to creating a chatbot, including rule-based chatbots and machine learning-based chatbots. Rule-based chatbots follow predefined rules and patterns, while machine learning-based chatbots use NLP models to understand and generate responses. Depending on your goals, you can choose one or a combination of these approaches.

3. Collect Data:

Data is crucial for training machine learning-based chatbots. You may need a dataset of conversational data or relevant text data to train your model. Preprocessing and cleaning the data is often necessary.

4. Select NLP Libraries and Frameworks:

Python offers various NLP libraries and frameworks that can be used to build chatbots. Some popular ones include:

- Natural Language Toolkit (NLTK)
- spaCy
- GPT-3 (OpenAI's model, which requires API access)
- Rasa (for creating conversational AI)

5. Preprocessing and Tokenization:

You'll need to preprocess and tokenize the text data to prepare it for analysis. This involves tasks like removing stop words, punctuation, and stemming/lemmatization.

6. Train Your Chatbot:

If you're using a machine learning-based approach, you'll need to train your model. This involves feeding it your dataset and fine-tuning the model to generate meaningful responses. Techniques like sequence-to-sequence models, transformers, and recurrent neural networks (RNNs) can be used.

7. Implement Conversation Flow:

Design the conversation flow and logic of your chatbot. This includes defining how the chatbot should respond to different user inputs and interactions. You can use if-else statements or decision trees for rule-based bots, or create a dialogue management system for more complex interactions.

8. Integration:

Integrate your chatbot into the desired platform or application. This could be a website, a messaging app, or any other medium where users will interact with the chatbot.

9. Testing and Feedback:

Thoroughly test your chatbot to ensure it functions as intended. Collect user feedback and continually improve the bot's responses and capabilities.

10. Deployment:

Once your chatbot is ready, deploy it to a server or hosting platform. Ensure it is accessible to users.

11. Maintenance and Updates:

Chatbots require ongoing maintenance and updates to adapt to changing user needs and to fix any issues that may arise.

12. User Experience (UX) Design:

Consider the user interface and experience when designing your chatbot. A well-designed chatbot will be more engaging and user-friendly.

DATA SET LINK: **https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot**

DATA SET:

```
hi, how are you doing?  i'm fine. how about yourself?
i'm fine. how about yourself?   i'm pretty good. thanks for asking.
i'm pretty good. thanks for asking.     no problem. so how have you been?
no problem. so how have you been?       i've been great. what about you?
i've been great. what about you?        i've been good. i'm in school right now.
i've been good. i'm in school right now.        what school do you go to?
what school do you go to?       i go to pcc.
i go to pcc.    do you like it there?
do you like it there?   it's okay. it's a really big campus.
it's okay. it's a really big campus.    good luck with school.
good luck with school.  thank you very much.
how's it going? i'm doing well. how about you?
i'm doing well. how about you?  never better, thanks.
never better, thanks.   so how have you been lately?
so how have you been lately?    i've actually been pretty good. you?
i've actually been pretty good. you?    i'm actually in school right now.
i'm actually in school right now.       which school do you attend?
which school do you attend?     i'm attending pcc right now.
i'm attending pcc right now.    are you enjoying it there?
are you enjoying it there?      it's not bad. there are a lot of people there.
it's not bad. there are a lot of people there.  good luck with that.
good luck with that.    thanks.
how are you doing today?        i'm doing great. what about you?
```

DESIGN THINKING:

Empathize:

Understand the perspective of the endusers or stakeholders by actively listening, observing, and engaging with them.

Develop empathy by conducting interviews, surveys, and observations to uncover their needs, desires, pain points, and challenges.

Define:

Clearly articulate and define the problem or challenge based on the insights gained during the empathize stage.

Create a problem statement or a usercentered point of view (POV) that frames the issue and provides a focus for the rest of the design process.

Ideate:

Encourage creative brainstorming sessions to generate a wide range of potential solutions without judgment.

Explore various ideas, concepts, and approaches to address the defined problem.

Use techniques like mind mapping, brainstorming, and ideation workshops to foster creativity.

Prototype:

Build lowfidelity prototypes or mockups of the proposed solutions. These can be sketches, paper prototypes, digital wireframes, or physical models.

Prototyping helps to visualize and communicate ideas and concepts, making them more tangible for evaluation and feedback.

Test:

Put the prototypes in front of users or stakeholders to gather feedback and insights.

Observe how users interact with the prototypes and note their reactions, preferences, and pain points.

Use feedback to refine and iterate on the prototypes, making improvements based on user input.

Implement (or Scale):

Once a viable solution has been identified and refined through testing, move forward with implementation.

Develop a plan for scaling and launching the solution, considering factors like technology, resources, and user adoption.

Continuously gather feedback and make adjustments as needed after implementation.

PROGRAM:

PROGRAM:

```
import numpy as np
import string
from nltk.corpus import stopwords
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction.text import TfidfTransformer,TfidfVectorizer
from sklearn.pipeline import Pipeline
df pd.read_csv('../input/simple-dialogs-for-chatbot/dialogs.txt',sep='\t')
a = pd.Series(df.columns)
df
a = a.rename({0: df.columns[0],1: df.columns[1]})
b = {'Questions':'Hi','Answers':'hello'}
c = {'Questions':'Hello','Answers':'hi'}
d= {'Questions':'how are you','Answers':"i'm fine. how about yourself?"}
e= {'Questions':'how are you doing','Answers':"i'm fine. how about yourself?"}
df = df.append(a,ignore_index=True)

df.columns=['Questions','Answers']
```

```
df = df.append([b,c,d,e],ignore_index=True)
Df
df = df.append(c,ignore_index=True)

 df = df.append(d,ignore_index=True)
df
def cleaner(x):
    return [a for a in (''.join([a for a in x if a not in string.punctuation])).lower().split()]

Pipe = Pipeline([
    ('bow',CountVectorizer(analyzer=cleaner)),
    ('tfidf',TfidfTransformer()),
    ('classifier',DecisionTreeClassifier())
])
Pipe.fit(df['Questions'],df['Answers'])
Pipe.fit(df['Questions'],df['Answers'])
Pipe.predict(['how are you'])[0]
Pipe.predict(['great'])[0]
Pipe.predict(['What are you doing'])[0]
```

OUTPUT:

```
Pipeline(steps=[('bow',
                 CountVectorizer(analyzer=<function cleaner at 0x7f5cfaae40e0>)),
                ('tfidf', TfidfTransformer()),
                ('classifier', DecisionTree
'hello'
"i'm fine. how about yourself?"
'i appreciate that.'
"i'm going to change the light bulb. it burnt
```

BUILD LOADING AND PREPROCESSING THE DATASET

Data preprocessing for a chatbot typically involves several steps to clean, format, and structure the data to make it suitable for training a chatbot model. Here's a general outline of the data preprocessing steps using Python:

1. Data Collection:
   - Collect the chatbot training data from various sources, such as chat logs, user interactions, or existing datasets.

2. Text Lowercasing:
   - Convert all text to lowercase to ensure consistency in the data. This helps the model generalize better.

3. Tokenization:
   - Tokenize the text into words or subword tokens. Common libraries for this include NLTK, spaCy, or the Hugging Face Transformers library for BERT-based tokenization.

4. Remove Special Characters:
   - Remove unnecessary special characters, punctuation, and symbols that don't convey meaningful information for the chatbot.

5. Handle Contractions:
   - Expand contractions to standardize the language (e.g., converting "can't" to "cannot").

6. Remove Stop Words:
   - Remove common stop words (e.g., "the," "and," "is") to reduce noise in the data.

7. Spelling Correction (optional):
   - Correct common spelling errors using libraries like TextBlob or autocorrect.

8. Remove Duplicates:
   - Eliminate duplicate or highly similar dialogues or responses to avoid bias in the training data.

9. Data Splitting:
   - Split the data into training, validation, and test sets to evaluate the chatbot's performance.

10. Text Encoding:
    - Convert the text into numerical representations that machine learning models can process. For instance, you can use word embeddings like Word2Vec, GloVe, or pre-trained models like BERT for contextual embeddings.

11. Padding and Truncating:
    - Ensure that all input sequences are of the same length by padding or truncating as necessary. This is important for batching during training.

12. Handling Out-of-Vocabulary (OOV) Words:
    - Implement an OOV strategy for words that are not in the vocabulary of your pre-trained embeddings, such as using a special token for OOV words.

13. Data Formatting:
    - Format the data into a format that your chatbot model can accept. This often involves creating input-output pairs for a seq2seq model or arranging data into a suitable format for a retrieval-based model.

14. Save Processed Data:
    - Save the preprocessed data in a format that's easy to load for model training and evaluation. This might involve saving it as CSV, JSON, or in a database.

15. Exploratory Data Analysis (EDA):
    - Perform some EDA to gain insights into your data. This can help you identify patterns, check for class imbalances, and make informed decisions about data augmentation or model selection.

TOOLS USED:

Here are some of the key tools and libraries commonly used in Python-based chatbot development:

1. Natural Language Processing (NLP) Libraries:
   - NLTK (Natural Language Toolkit): NLTK is a popular library for natural language processing. It provides tools for tokenization, stemming, lemmatization, parsing, and more.
   - spaCy: spaCy is a more modern NLP library known for its speed and accuracy in natural language processing tasks. It offers pre-trained models for various languages.
   - Gensim: Gensim is a library for topic modeling and document similarity analysis, useful for understanding and categorizing text data.
   - TextBlob: TextBlob is a simplified NLP library that can be used for tasks like part-of-speech tagging, sentiment analysis, and translation.

2. Machine Learning and Deep Learning Libraries:
   - TensorFlow: TensorFlow is a powerful library for building and training machine learning and deep learning models, which can be used for chatbot development, especially in the case of AI-driven chatbots.
   - PyTorch: PyTorch is another popular deep learning library that is commonly used for developing neural network-based chatbots.
   - scikit-learn: scikit-learn is a versatile library for traditional machine learning tasks, such as classification and clustering, which can be applied to chatbot projects.

3. Chatbot Frameworks:

- Rasa: Rasa is an open-source chatbot development framework that provides tools for natural language understanding, dialogue management, and integration with messaging platforms.
- ChatterBot: ChatterBot is a Python library for building chatbots based on machine learning and can be integrated with various platforms.

4. Web Frameworks:
- Flask: Flask is a lightweight web framework that can be used to create web-based chatbot interfaces or APIs.
- Django: Django is a more feature-rich web framework that can be used for creating complex chatbot applications with web interfaces.

5. Messaging Platform APIs:
- If you want your chatbot to interact with messaging platforms like Facebook Messenger, WhatsApp, or Slack, you'll need to use their respective APIs and libraries.

6. Database Management:
- Depending on your chatbot's requirements, you may need to integrate with databases. Python offers libraries like SQLAlchemy and Django ORM for database management.

7. Dependency Management:
- Use tools like `pip` for installing and managing Python packages and libraries. Consider using virtual environments to isolate project dependencies.

8. Development Environments:
- You can use integrated development environments (IDEs) like PyCharm, Visual Studio Code, or Jupyter Notebook for coding, debugging, and testing your chatbot.

9. Version Control:
- Version control systems like Git and platforms like GitHub or GitLab can help manage and collaborate on your chatbot project's source code.

10. Deployment and Hosting:
- To deploy your chatbot, you might use cloud platforms like AWS, Azure, or Heroku. Containerization tools like Docker can also simplify deployment.

11. Continuous Integration/Continuous Deployment (CI/CD):

- CI/CD tools like Jenkins, Travis CI, or GitHub Actions can automate the testing and deployment process for your chatbot.

PERFORMING DIFFERENT ACTIVITIES, THEY ARE:
FEATURE SELECTION:
Feature selection is a crucial step in building machine learning models, including chatbots. Selecting the right features can significantly impact the performance and efficiency of your chatbot. In this example, I'll provide you with a simple program to perform feature selection for a chatbot using Python and the scikit-learn library.

CODE

```
import numpy as np
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

# Sample chat data (replace with your own dataset)
corpus = [
    "Hello, how can I help you?",
    "Tell me about your products.",
    "How do I contact customer support?",
    "What is your return policy?",
    # Add more chat interactions
]

# Sample labels (replace with your labels)
labels = [0, 1, 2, 3]  # Example labels

# Convert text data to numerical features using CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2,
random_state=42)

# Perform feature selection using chi-squared test
k_best = SelectKBest(chi2, k=2)  # You can change 'k' to the desired number of
features
```

```
X_new = k_best.fit_transform(X_train, y_train)

# Get the selected feature indices
selected_feature_indices = k_best.get_support(indices=True)

# Print the selected feature indices
print("Selected feature indices:", selected_feature_indices)

# Transform the test data using the selected features
X_test_new = X_test[:, selected_feature_indices]

# Now, you can use X_new and X_test_new as your selected features for
training your chatbot model.
```

In this example:

1. We have a sample corpus of chat interactions and labels. You should replace these with your actual dataset.

2. We use the `CountVectorizer` to convert text data into numerical features.

3. We split the data into training and testing sets.

4. Feature selection is performed using the chi-squared test through the `SelectKBest` method. You can change the 'k' parameter to specify the number of features you want to select.

5. The program prints the selected feature indices, which correspond to the columns in your feature matrix.

6. Finally, we transform the test data using the selected features for evaluation.

MODEL TRAINING:

Model training for a chatbot typically involves training a machine learning or deep learning model, depending on your chatbot's architecture and requirements. In this example, I'll provide a simple way to train a rule-based chatbot using a Python program. This chatbot won't involve machine learning

but instead uses a set of predefined rules to respond to user inputs. You can extend and improve this rule-based chatbot as needed.

Here's a Python program for training a rule-based chatbot:

CODE

```
class RuleBasedChatbot:
    def __init__(self):
        self.rules = {
            "hello": "Hello! How can I assist you?",
            "products": "Our products include item A, item B, and item C.",
            "contact": "You can contact our customer support team at
support@chatbot.com.",
            "return": "Our return policy allows for returns within 30 days of
purchase. Please visit our website for more details.",
        }

    def respond(self, user_input):
        user_input = user_input.lower()
        for keyword, response in self.rules.items():
            if keyword in user_input:
                return response
        return "I'm sorry, I couldn't understand your question. Please try again."

# Instantiate the chatbot
chatbot = RuleBasedChatbot()

# Main interaction loop
while True:
    user_input = input("You: ")
    if user_input.lower() == "exit":
        print("Chatbot: Goodbye!")
        break
    response = chatbot.respond(user_input)
    print("Chatbot:", response)
```

This program creates a simple rule-based chatbot:

1. The `RuleBasedChatbot` class defines a set of rules and responses. When a user inputs a message, the chatbot checks for keywords in the input and responds based on these predefined rules.

2. The `respond` method takes the user's input, converts it to lowercase for case-insensitive matching, and looks for keywords in the input to provide an appropriate response.

3. The main interaction loop allows users to input messages and receive responses from the chatbot. Typing "exit" will exit the chat.

This is a very basic example, and in practice, you'd want to improve and expand the rules and responses to make the chatbot more useful. For more advanced chatbots that use machine learning, you'll need to follow the specific training procedures for the chosen machine learning or deep learning framework, as mentioned in your earlier question. Training models for machine learning-based chatbots often involve preprocessing data, feature extraction, and training the model on a labeled dataset. The choice of model architecture will depend on the complexity of your chatbot's tasks and the size of your dataset.
To create a machine learning-based chatbot, you'll need a labeled dataset for training and another dataset for testing or making predictions. The labeled dataset consists of input messages and corresponding labels or intents (i.e., what the user intends with their message), and the test dataset includes input messages for which you want the chatbot to predict the corresponding intents.

Here's an example of how your labeled training dataset might look:

plaintext

Input Message	Intent
Hello, how are you?	Greeting
Tell me about your products.	ProductInfo
How do I contact support?	ContactInfo
What's your return policy?	ReturnPolicy
Where are you located?	Location

...

And here's an example of a test dataset:

plaintext
Input Message

What are your hours?
Can you help me with billing?
How does your service work?
...

In this example, the "Intent" column in the training dataset represents the intent or action the chatbot should take in response to the corresponding input message. This is the target variable you'll train your machine learning model to predict.

Feature Engineering:
Feature engineering is a crucial part of building a machine learning-based chatbot. It involves extracting relevant information from the text input to represent it in a numerical format that the model can understand. Here are some common features and feature engineering techniques:

1. Bag of Words (BoW): Representing each input message as a vector of word frequencies. You can use tools like CountVectorizer or TfidfVectorizer from scikit-learn.

2. Word Embeddings: Using pre-trained word embeddings like Word2Vec, GloVe, or fastText to convert words into dense vectors.

3. Text Preprocessing:
   - Tokenization: Splitting the text into words or subword tokens.
   - Lowercasing: Converting all text to lowercase for consistency.
   - Removing Stopwords: Eliminating common words (e.g., "and," "the") that don't carry much information.
   - Lemmatization or Stemming: Reducing words to their root form (e.g., "running" to "run").

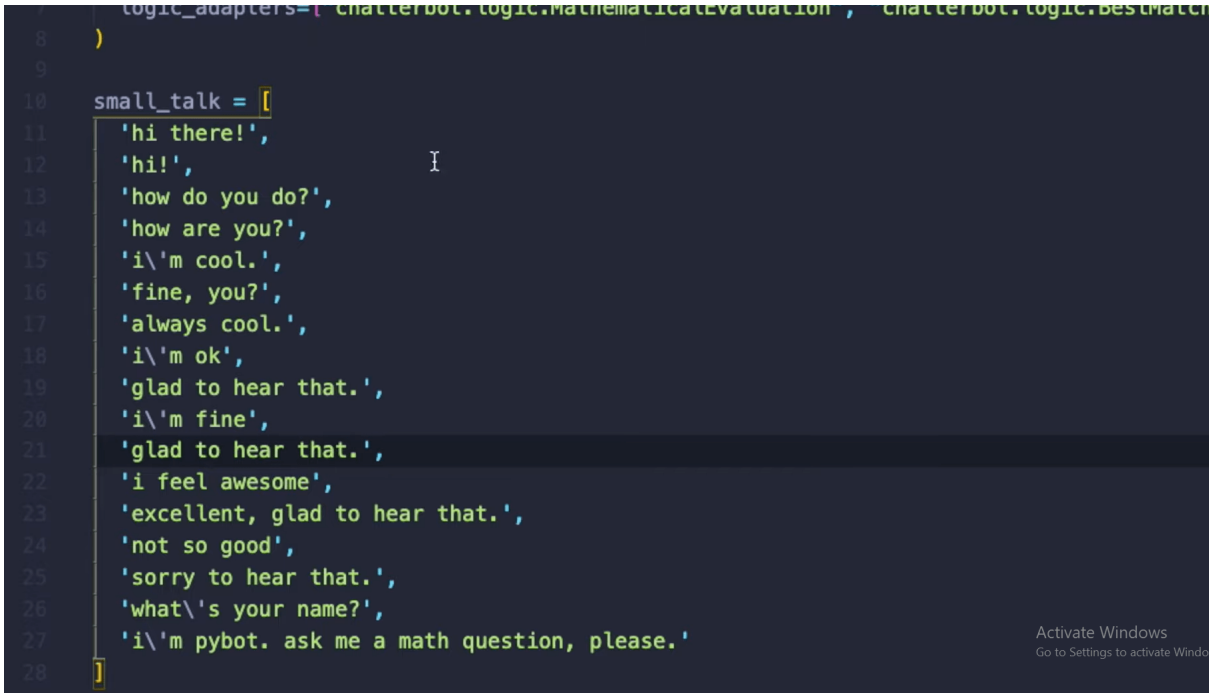
4. TF-IDF Features: Term Frequency-Inverse Document Frequency (TF-IDF) is a way to represent the importance of words in a document relative to a corpus of documents.

5. N-grams: Capturing word sequences, not just individual words, by using n-grams (e.g., bigrams or trigrams).

6. Sentiment Analysis: Analyzing the sentiment of the input message (positive, negative, neutral) using sentiment analysis tools.

7. Named Entity Recognition (NER): Identifying and categorizing named entities in the text (e.g., names of people, organizations, locations).

8. Part-of-Speech (POS) Tagging: Identifying the grammatical structure of sentences.



Model Selection:
You can choose from various machine learning or deep learning models for your chatbot, including:

1. Rule-Based Models: These are simple and based on predefined rules and patterns, as shown in the earlier examples.

2. Naive Bayes Classifier: A probabilistic model for text classification.

3. Support Vector Machines (SVM): Effective for text classification tasks.

4. Recurrent Neural Networks (RNNs): Useful for sequence-to-sequence tasks.

5. Convolutional Neural Networks (CNNs): Good for text classification.

6. Transformer-based Models: Such as BERT, GPT-3, etc., which are state-of-the-art for natural language understanding tasks.

Predicting Data:
To make predictions on the test dataset, you need to preprocess the test data using the same techniques you applied to the training data. Then, you can use your trained model to predict the intent of each input message in the test dataset. The model will output the predicted intent or response for each message.

ADVANTAGES:
1. Wide Adoption and Community Support: Python is one of the most widely used programming languages, and it has a large and active community of developers. This means you'll have access to a wealth of resources, libraries, and frameworks that can help you build and maintain your chatbot.

2. Versatility: Python is a versatile language that can be used for a wide range of applications, including chatbots. You can integrate your chatbot with various platforms, databases, APIs, and other technologies using Python.

3. Rich Ecosystem: Python offers numerous libraries and frameworks that can simplify chatbot development. Popular libraries like NLTK, spaCy, and TensorFlow can be used for natural language processing (NLP) and machine learning, which are essential for chatbots.

4. NLP Capabilities: Python has excellent NLP libraries and tools, making it well-suited for building chatbots that can understand and generate human language. You can use tools like spaCy, NLTK, and Gensim to perform tasks like text analysis, sentiment analysis, and named entity recognition.

5. Machine Learning and AI Integration: Python is widely used in the field of machine learning and artificial intelligence. You can leverage libraries like TensorFlow, PyTorch, and scikit-learn to build and train chatbots that can learn from user interactions and improve over time.

6. Integration with Web Technologies: Python can be used to create web-based chatbots that can be easily integrated into websites and web applications. Frameworks like Flask and Django allow you to build web interfaces for your chatbot.

7. Cross-Platform Compatibility: Python is a cross-platform language, meaning you can develop chatbots that work on various operating systems, including Windows, macOS, and Linux.

8. Rapid Development: Python's clean and readable syntax allows for faster development and easier debugging, which can be crucial when iterating on your chatbot.

DISADVANTAGES:

1. Performance: Python is an interpreted language, which can make it slower compared to compiled languages like C++ or Java. This may not be a significant issue for many chatbots, but for extremely high-performance applications, it could be a concern.

2. Resource Intensive: Python can be relatively resource-intensive in terms of memory and CPU usage, which could be a limitation for chatbots deployed on resource-constrained environments, such as embedded systems.

3. Global Interpreter Lock (GIL): Python has a Global Interpreter Lock, which can restrict the execution of multiple threads simultaneously. This can impact the performance of multi-threaded chatbot applications, as it may not fully utilize multi-core processors.

4. Limited Mobile Development: While Python can be used for mobile app development, it may not be the first choice for creating chatbots on mobile platforms, as mobile operating systems often favor languages like Swift (iOS) and Kotlin (Android).

5. Limited Access to Low-Level System Features: Python abstracts many low-level system operations, which can be a limitation when you need to interact closely with hardware or low-level system features in your chatbot.

6. Dependency Management: Python's dependency management can be challenging, especially when working on projects with numerous external libraries and dependencies. Tools like virtual environments can help manage dependencies, but it can still be a complex task.

7. Less Suitable for Real-Time Applications: For applications requiring real-time performance and low-latency responses, Python might not be the ideal choice due to its inherent overhead and slower execution speed.

CONCLUSION:

In conclusion, developing a chatbot using Python offers several advantages, including a vast community, rich ecosystem of libraries, natural language processing capabilities, machine learning integration, cross-platform compatibility, and cost-efficiency. Python's versatility and ease of use make it a popular choice for chatbot development, and it's suitable for a wide range of applications.

However, it's essential to consider the disadvantages, such as potential performance limitations, resource intensity, the Global Interpreter Lock (GIL), and the need for careful dependency management. Python may not be the best choice for real-time, resource-constrained, or extremely high-performance chatbot applications.

Ultimately, the decision to use Python for chatbot development should be based on your project's specific requirements, performance considerations, and the trade-offs you're willing to make. By carefully assessing your needs and planning your chatbot development, you can leverage Python's strengths while mitigating its limitations to create an effective and efficient chatbot solution.