**CREATE A CHATBOT IN PYTHON**

**(PHASE 5 DOCUMENT)**

**Problem Statement :**

When using an app or website, customers expect outstanding service. They can become disinterested in the app if they can't locate the solution to a question they have. To avoid losing customers and having an adverse effect on your bottom line, you must provide the highest quality service possible while developing a website or application

**Project Definition and Design Thinking :**

Creating a chatbot using Python involves developing a computer program or script that can simulate human-like conversations with users. This process typically includes utilizing various Python libraries, frameworks, and natural language processing (NLP) techniques to enable the chatbot to understand user inputs, process them, and provide appropriate responses.



**General steps involved in creating a chatbot :**

To create a chatbot using Python, you can follow these general steps:

**1. Choose a Framework or Library**: There are several libraries and frameworks available in Python for building chatbots.

Some popular ones include:

* NLTK (Natural Language Toolkit): Good for basic chchatbot
* spaCy: Useful for NLP tasks.
* Rasa: An open-source framework for building conversational AI.
* ChatterBot: A simple library for creating chatbots.

**2. Install Necessary Libraries:** Depending on the framework or library you choose, install the required packages using pip, for example: pip install nltk

**3. Data Collection:** Gather or generate a dataset of questions and responses for your chatbot. This dataset will be used to train your chatbot to understand and respond to user inputs.

**4. Data Preprocessing:** Preprocess your dataset, which may involve tokenization, stemming, or lemmatization, to prepare the text data for training.

**5. Build and Train Your Model:** Use the chosen framework to build and train your chatbot model. This often involves creating intents, entities, and dialogue flows.

**6. Integration:** Integrate your chatbot into a platform or application. You can create a web-based chat interface or integrate it with messaging platforms like Slack or Facebook Messenger.

**7. Testing:** Thoroughly test your chatbot to ensure it responds correctly to user inputs. Make adjustments as needed.

**8. Deploy:** Once your chatbot works as expected, deploy it to a server or cloud platform so that it can be accessed by users.

**9. Continuously Improve**: Monitor the interactions and user feedback to improve your chatbot's performance. You can retrain your model with new data and make updates to its responses.

**10. Scale:** If your chatbot gains popularity, consider scaling its infrastructure to handle increased traffic.

**INNOVATION :**

consider exploring advanced techniques like using pre-trained language models (e.g., GPT-3) to enhance the quality of responses.



**Steps to create enhanced chatbot in python :**

Creating an advanced chatbot for solving customer queries using Python requires integrating natural language processing (NLP) techniques, a knowledge base, and potentially machine learning. Here's a high-level outline of how to build one:

**1. Data Collection and Preprocessing:**

Gather and preprocess the data you'll use to train your chatbot. This may include historical customer support interactions, FAQs, and other relevant documents. Tokenize and clean the text data.

**2. Choose a Framework or Library:**

Select a suitable NLP framework or library. For advanced chatbots, you might consider using Hugging Face Transformers for pre-trained models or custom models using spaCy or NLTK.

**3. Train an Intent Recognition Model:**

Use a machine learning model (e.g., SVM, Random Forest, or neural networks) to classify user queries into specific intents. This helps your chatbot understand the user's request better. Here's an example using scikit-learn:

**python**

**from sklearn.feature\_extraction.text import TfidfVectorizer**

**from sklearn.svm import SVC**

**# Train your intent recognition model with labeled data**

**vectorizer = TfidfVectorizer()**

**X\_train = vectorizer.fit\_transform(training\_data)**

**intent\_classifier = SVC(kernel='linear')**

**intent\_classifier.fit(X\_train, labels)**

**4. Create a Knowledge Base**:

Populate a knowledge base with answers to common customer queries. You can use a database, a structured file, or a dedicated knowledge management system.

**5. Build a Response Generator:**

Implement a response generator that, based on the user's intent and context, retrieves answers from the knowledge base or generates responses using a pre-trained language model. database, a structured file, or a dedicated knowledge management system.You can use Transformers for this purpose, similar to the previous example.

**6. Implement Dialog Management:**

Manage the conversation flow, including greeting, maintaining context, and handling multi-turn conversations. A dialogue manager ensures that the chatbot maintains coherent interactions.

**7. Connect to Communication Channels:**

Integrate your chatbot with communication channels like webchat, email, or messaging apps using APIs or SDKs.

**8. Testing and Evaluation:**

Thoroughly test your chatbot with real data to ensure it understands user queries and provides accurate responses. Use metrics like accuracy, precision, and recall for intent recognition and user satisfaction for response quality.

**9. Iterate and Improve:**

Continuously gather user feedback and iterate on your chatbot's performance. Fine-tune the intent recognition model and improve the knowledge base.

**10. Deployment:**

Deploy your chatbot to a web server, cloud platform, or any suitable infrastructure for production use.

**11. Monitoring and Maintenance:**

Regularly monitor your chatbot's performance, address issues, and keep it up to date with new FAQs or knowledge.

**12. Privacy and Security:**

Ensure that your chatbot handles user data securely and complies with data privacy regulations.

**GPT - 3 BASED TRAINED CHATBOT :**

Enhancing a GPT-3-based chatbot involves improving its capabilities, making it more context-aware, and addressing specific use-case requirements. Here are some advanced enhancements you can implement:

**1. Fine-Tuning:**

Fine-tune the GPT-3 model on your specific domain or tasks. By providing custom training data, you can make the chatbot more specialized and accurate in understanding and generating content related to your industry or business.

**2. Custom Prompt Engineering:**

Craft prompts carefully to elicit more contextually relevant responses. Experiment with different input phrasings and instructions to get the desired output. You can guide the model by providing explicit context in the prompts.

**3. Response Post-Processing:**

After receiving a response from GPT-3, you can post-process it to ensure it meets specific requirements, such as content guidelines, formatting, or filtering out inappropriate content.

**4. Multi-turn Conversations:**

Implement multi-turn conversations by maintaining a conversation history and context. This allows your chatbot to engage in more extended and coherent interactions with users.

**5. User Profiling:**

Keep track of user preferences and historical interactions to personalize responses and provide recommendations. This can improve user engagement and satisfaction.

**6. Content Moderation:**

Implement content moderation and filtering mechanisms to prevent the chatbot from generating harmful or inappropriate content. This is crucial for maintaining a safe and responsible chatbot.

**7. Knowledge Integration:**

Combine GPT-3's capabilities with external knowledge bases or databases to provide accurate and up-to-date information. For example, you can integrate with APIs to fetch real-time data.

**8. Intent Recognition:**

Enhance your chatbot's understanding of user intents by using machine learning models to classify user queries into specific categories or actions. This can improve the relevance of responses.

**9. A/B Testing:**

Continuously experiment with different prompts, strategies, and models using A/B testing to measure the effectiveness of various enhancements and iterate on improvements.

**10. Feedback Loop:**

Implement a feedback mechanism where users can rate responses, report issues, or provide feedback. Use this feedback to refine and improve the chatbot's performance.

**11. Dynamic Generation Length:**

Adjust the length of responses dynamically based on the context and user input. For short queries, generate concise responses, and for more complex questions, allow longer responses.

**12. Error Handling:**  Implement robust error handling mechanisms to gracefully handle situations where the chatbot doesn't understand the user's query or encounters issues.

**13. Natural Language Understanding (NLU):**

Integrate NLU techniques to extract entities and key information from user queries, enabling the chatbot to provide more precise and relevant responses.

**14. Multimodal Capabilities:**

Explore incorporating other modalities like images, videos, or audio into the chatbot's responses if your use case requires it.

**15. Performance Optimization:**

Optimize the chatbot's performance by considering factors like response time, cost, and resource utilization, especially if you plan to deploy it at scale.

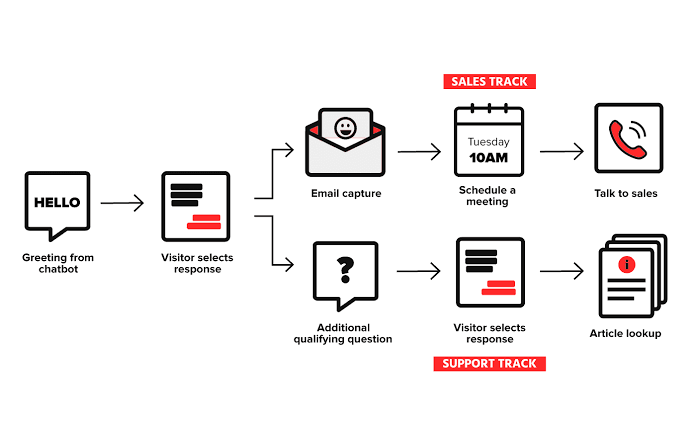
**16. Ethical Considerations:**

Be mindful of ethical and responsible AI practices, including bias mitigation, transparency, and user privacy.

Enhancing a GPT-3-based chatbot is an ongoing process that involves experimentation, feedback analysis, and continuous improvement. Keep iterating and refining your chatbot to meet the evolving needs of your users and business.

**Implementation**

V*irtual Environment Creation*



Before starting the coding part of our chatbot development, let’s create a virtual environment for the chatbot. The python library that we are using to create the virtual environment is “virtualenv”.

So first of all let’s install virtualenv(In the command prompt)

Now we can create our virtual environment named my\_env, so take the terminal in the vscode or any code editor and write the below code.

**virtualenv my\_env**

**Next is to activate our virtual environment.**

**Activation in windows power shell**

**my\_envScriptsactivate.ps1**

**Activation in command prompt**

**my\_envScriptsactivate.bat**

The virtual environment is activated.

**Installation of Libraries**

To install the libraries required for this project separately in this enenvironment

pip install keras nltk tensorflow

**Creating Intents File**

First of all, let’s look into our intents\_file.json file. This intents file contains the different patterns of the question that the user might enquire and the possible output for the specific question and a tag for that type of question

**Code** :

{"intents": [

{"tag": "greetings",

"patterns": ["Hello there", "Hey, How are you", "Hey", "Hi", "Hello", "Anybody", "Hey there"],

"responses": ["Hello, I'm your helping bot", "Hey it's good to see you", "Hi there, how can I help you?"],

"context": [""]

},

{"tag": "thanks",

"patterns": ["Thanks for your quick response", "Thank you for providing the valuable information", "Awesome, thanks for helping"],

"responses": ["Happy to help you", "Thanks for reaching out to me", "It's My pleasure to help you"],

"context": [""]

},

{"tag": "no\_answer",

"patterns": [],

"responses": ["Sorry, Could you repeat again", "provide me more info", "can't understand you"],

"context": [""]

},

{"tag": "support",

"patterns": ["What help you can do?", "What are the helps you provide?", "How you could help me", "What support is offered by you"],

"responses": [ "ticket booking for airline", "I can help you to book flight tickets easily"],

"context": [""]

},

{"tag": "goodbye",

"patterns": ["bye bye", "Nice to chat with you", "Bye", "See you later buddy", "Goodbye"],

"responses": [ "bye bye, thanks for reaching", "Have a nice day there", "See you later"],

"context": [""]

}

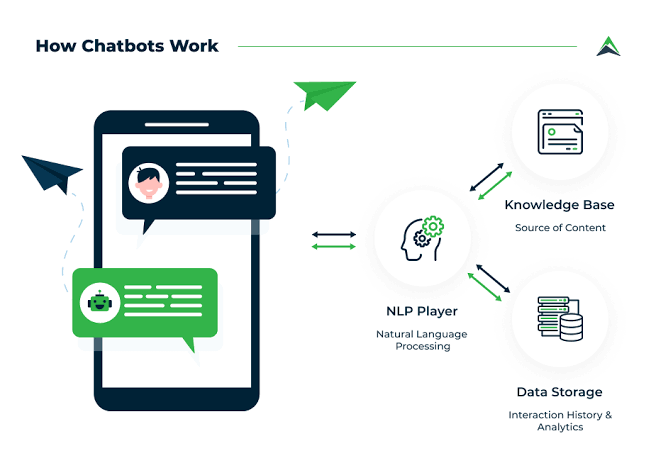
]

}

The above shows the intents file that we are going to use in our project

**Project solution :**

To build a chat bot by selecting machine learning algorithms and training model and perform different analysis also evaluating it's performance by Artifical intelligence.



**Steps involved in building machine learning algorithms for chat bot** :

Creating a chatbot with a machine learning model in Python typically involves several steps:

**1. Choose a Framework or Library:**

You can use popular libraries like TensorFlow, PyTorch, or Hugging Face's Transformers library for natural language processing tasks.

**2. Data Collection and Preprocessing:**

- Gather a dataset for training your chatbot. This dataset should consist of pairs of input (user message) and output (bot response).

- Preprocess the data, which may include tasks like tokenization, removing special characters, and converting text to lowercase.

**3.choose architecture model:**

- Depending on your preference, you can use recurrent neural networks (RNNs), transformers, or other deep learning architectures for this task.

**4. Model Training:**

- Use the preprocessed data to train your chosen model. This involves adjusting the model's parameters to minimize the loss function.

**5. Model Evaluation:**

- Assess the performance of your model using validation data. Common metrics include perplexity, BLEU score, or other relevant evaluation metrics for chatbot tasks.

**6. Inference:**

- Use the trained model to generate responses for user input. This is the actual interaction stage where users interact with the chatbot.

**7. Iterate and Improve:**

- Depending on performance, you may need to iterate on the model, data, or preprocessing steps to improve the chatbot's performance.

**Steps to evaluate performance of training model by python :**

When evaluating and analyzing a machine learning model for a chatbot, you'll want to consider a variety of metrics and techniques. Here are some common steps and methods:

**1. Performance Metrics:**

**Perplexity**: For language models, perplexity measures how well the model predicts the next word in a sequence. Lower perplexity indicates better performance.

**BLEU Score:** Commonly used to evaluate the quality of machine-translated text. It can also be adapted for chatbot responses.

**ROUGE Score**: Evaluates the overlap between model-generated text and reference text.

**F1 Score:** Measures the balance between precision and recall in multi-class classification tasks.

**Accuracy:** Appropriate for classification tasks where you have labeled classes for responses.

**2. Human Evaluation:**

User Studies: Conduct surveys or tests with real users to assess the quality of the bot's responses. This provides subjective feedback on user satisfaction.

A/B Testing : Compare the performance of different versions of your chatbot with real users to see which one is preferred.

**3. Contextual Understanding:**

- Evaluate the model's ability to understand and maintain context in conversations. For example, ensure that it can refer back to earlier parts of the conversation appropriately.

**4. Error Analysis:**

Identify common types of errors the model makes. Is it often failing to understand the user's intent, or is it generating grammatically incorrect responses?

**5. Response Coherence:**

- Ensure that the bot's responses are logically connected to the user's input and previous messages in the conversation.

**6. Handling Out-of-Domain Inputs:**

Creating a chatbot with a machine learning model in Python typically involves several steps:

**1. Choose a Framework or Library:**

- You can use popular libraries like TensorFlow, PyTorch, or Hugging Face's Transformers library for natural language processing tasks.

**2. Data Collection and Preprocessing**:

- Gather a dataset for training your chatbot. This dataset should consist of pairs of input (user message) and output (bot response).

- Preprocess the data, which may include tasks like tokenization, removing special characters, and converting text to lowercase.

**3. Choose a Model Architecture:**

- Depending on your preference, you can use recurrent neural networks (RNNs), transformers, or other deep learning architectures for this task.

**4. Model Training:**

- Use the preprocessed data to train your chosen model. This involves adjusting the model's parameters to minimize the loss function.

**5. Model Evaluation**:

- Assess the performance of your model using validation data. Common metrics include perplexity, BLEU score, or other relevant evaluation metrics for chatbot tasks.

**6. Inference:**

- Use the trained model to generate responses for user input. This is the actual interaction stage where users interact with the chatbot.

**7. Iterate and Improve**

- Depending on performance, you may need to iterate on the model, data, or preprocessing steps to improve the chatbot's performance.

Here's a very simplified example using Hugging Face's Transformers library with a pre-trained model:

**python**

**# Install the transformers library**

**!pip install transformers**

**# Import necessary libraries**

**from transformers import GPT2Tokenizer, GPT2LMHeadModel**

**# Load pre-trained model and tokenizer**

**tokenizer = GPT2Tokenizer.from\_pretrained("gpt2")**

**model = GPT2LMHeadModel.from\_pretrained("gpt2")**

**# Encode user input and generate a response**

**user\_input = "How are you?"**

**input\_ids = tokenizer.encode(user\_input, return\_tensors="pt")**

**output = model.generate(input\_ids, max\_length=50, num\_return\_sequences=1)**

**# Decode the generated output**

**bot\_response = tokenizer.decode(output[0], skip\_special\_tokens=True)**

**print(bot\_response)**

Please note that this is a basic example. For a more sophisticated chatbot, especially one that handles complex conversations,

**7. Handling Ambiguity and Open-Ended Questions:**

Evaluate the model's ability to respond appropriately to ambiguous or open-ended questions.

**8. Comparative Analysis:**

- Compare the performance of your machine learning-based chatbot with other baselines or commercially available chatbots.

**9. Fine-Tuning and Iteration:**

- If the model's performance is unsatisfactory, consider fine-tuning on additional data or modifying the architecture.

**10. Scalability and Latency:**

Assess the performance of the chatbot under different loads to ensure it can handle real-world usage.

Remember, the choice of metrics and evaluation methods should align with the specific goals and requirements of your chatbot project. It's also important to continuously monitor and refine the chatbot as it interacts with real users.