PHASE 4: DEVELOPMENT PART2

- **FEATURE ENGINEERING**
- **MODEL TRAINING**
- **EVALUATION**

Feature engineering is an essential step in building a chatbot. It involves transforming raw data into meaningful input features that can be used for training and improving the performance of your chatbot. Below are some feature engineering techniques for a chatbot, along with Python source code examples:

1.Text Preprocessing:

Tokenization: Split text into words or subword tokens.

Stopword Removal: Eliminate common words like "and," "the," etc.

Stemming or Lemmatization: Reduce words to their root form.

```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
def preprocess_text(text):
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word.lower() not in stopwords.words('english')]
    stemmer = PorterStemmer()
    tokens = [stemmer.stem(word) for word in tokens]
    return ' '.join(tokens)
```

2.Bag of Words (BoW):

Create a BoW representation of text data.

```
from sklearn.feature_extraction.text import CountVectorizer
corpus = ["This is a sample sentence.", "Another sentence."]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)
```

3.TF-IDF (Term Frequency-Inverse Document Frequency):

Assign weights to words based on their importance.

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)
```

4.Word Embeddings:

Utilize pre-trained word embeddings like Word2Vec, GloVe, or FastText.

```
from gensim.models import Word2Vec
# Train Word2Vec model
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, sg=0)
```

5.Dialogue History: The chat history as context for the chatbot.

```
context = ["Hi, how can I help you?", "I'm looking for a laptop."]
user_input = "What are the best laptops?"
context.append(user_input)
```

6.Intent Classification:

pre-trained NLP model should be used for intent classification.

```
from transformers import BertTokenizer, BertForSequenceClassification
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
```

```
model = BertForSequenceClassification.from_pretrained("bert-base-uncased")

input_text = "Tell me a joke"

inputs = tokenizer(input_text, return_tensors="pt")

outputs = model(**inputs)
```

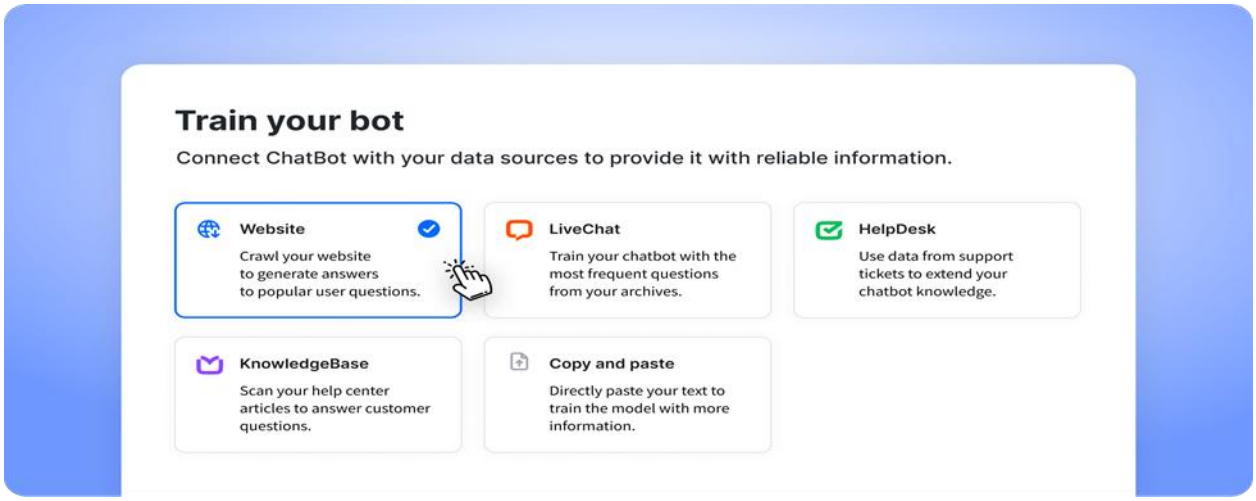
7.Named Entity Recognition (NER):

To Identify and tag entities in user messages.

```
from spacy import load

nlp = load("en_core_web_sm")

doc = nlp("I need a flight to New York on October 15th.")

entities = [(ent.text, ent.label_) for ent in doc.ents]
```

MODEL TRAINING:

Model training of a chatbot involves the process of training a machine learning model to understand and generate human-like responses in a conversational manner.



The training typically consists of two main steps: pre-training and fine-tuning.

During pre-training, the model is exposed to a large dataset containing a wide range of text from the internet. This helps the model learn grammar, facts, and some level of reasoning. However, it is important to note that the model does not have knowledge of specific sources or the ability to verify information.

After pre-training, the model goes through fine-tuning, where it is trained on a more specific dataset that is carefully generated with the help of human reviewers. These reviewers follow

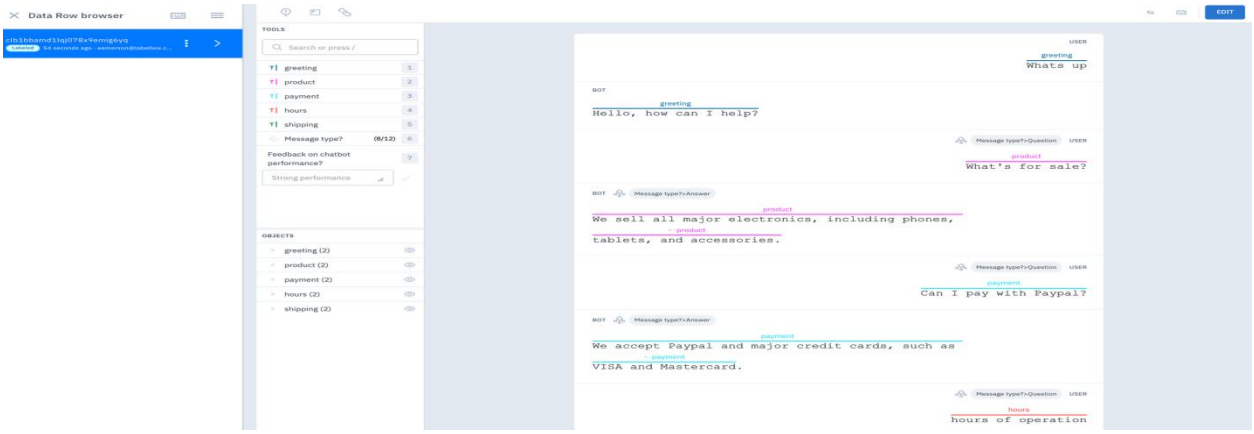
guidelines provided by the developers to review and rate possible model outputs for different inputs. This iterative feedback process helps the model improve its responses over time.

It is worth mentioning that the training process aims to strike a balance between generating creative and helpful responses while avoiding biased or inappropriate content. Developers continuously work on improving the model and addressing any limitations or biases that may arise.

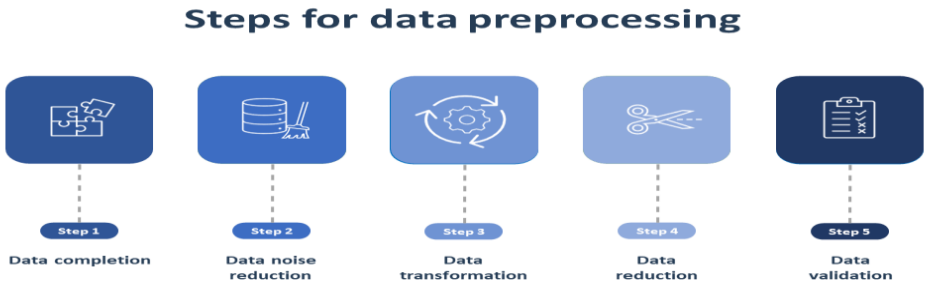
Overall, the model training of a chatbot involves a combination of pre-training on a large dataset and fine-tuning with human feedback to create a conversational AI that can generate human-like responses in multi-turn conversations.

Steps involved in training a chatbot

Data Collection: Gathering a large dataset of text conversations or dialogues that will be used to train the chatbot. This dataset can be obtained from various sources, such as customer support chats, online forums, or existing chat logs.

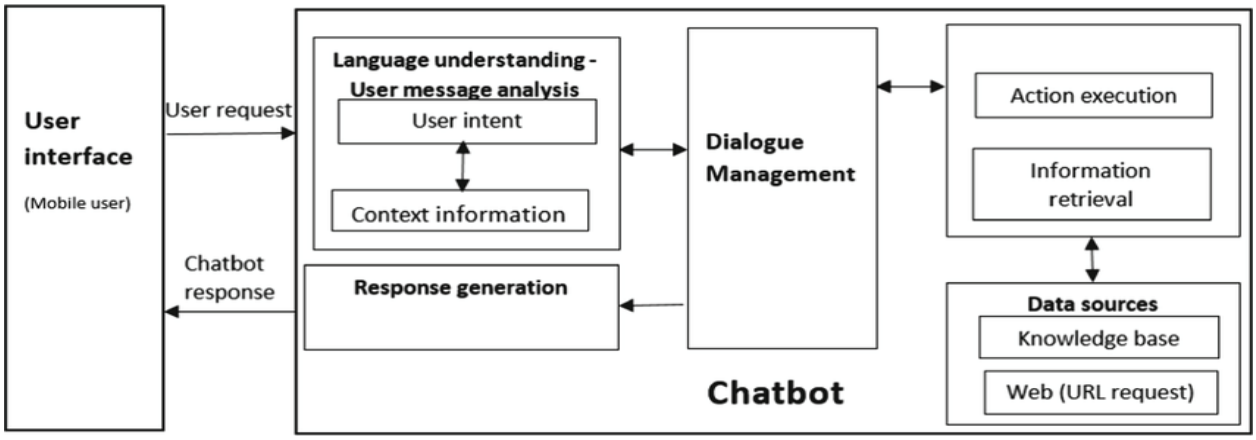


Data Preprocessing: Cleaning and preparing the collected data for training. This step involves removing irrelevant information, formatting the data into a suitable structure, and handling any inconsistencies or noise in the dataset.



Tokenization: Breaking down the text into smaller units called tokens, such as words or subwords. Tokenization helps the model understand the structure and meaning of the text.

Model Architecture Selection: Choosing an appropriate model architecture for the chatbot. This can include recurrent neural networks (RNNs), transformer models, or a combination of both. The architecture should be capable of handling multi-turn conversations.



Model Training: Training the selected model using the preprocessed data. This involves feeding the input text and training the model to predict the appropriate response. The training process typically involves optimizing model parameters using techniques like backpropagation and gradient descent.

Evaluation: Assessing the performance of the trained model. This can be done by using a separate validation dataset or by conducting human evaluations to measure the quality of the generated responses.

Fine-tuning: Iteratively refining the model by incorporating feedback from human reviewers. This step helps improve the model's responses, address biases, and ensure it aligns with desired guidelines and ethical considerations.

Deployment: Integrating the trained model into a chatbot application or platform, making it available for users to interact with.

Code

- Gather and label data needed to build a chatbot
- Download and import modules

```
import nltk
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()
nltk.download ('punkt')
from nltk.tokenize import word_tokenize
import numpy as np
import tflearn
import tensorflow as tf
import random
import json
import urllib3
```

Modules for data

```
[ ] http = urllib3.PoolManager()
    r = http.request ('GET', url)
    data = json.loads (r.data.decode('utf-8'))
```

- Pre-processing the data
- Tockenization

```
[ ] for intent in data["intents"]:
        for question in intent["questions"]:
          tokens = nltk.word_tokenize(question)
          words.extend(tokens)
          docs_questions.append(tokens)
          docs_intents.append(intent['tag'])

        if intent["tag"] not in labels:
            labels.append(intent["tag"])
```

- Stemming

```
words = [stemmer.stem(token.lower()) for token in wo

words = sorted(list(set(words)))
labels = sorted(labels)
```

- Set up training and test the output

- Create a bag-of-words (BoW)

```
[ ] for intent in data["intents"]:
        for question in intent["questions"]:
          tokens = nltk.word_tokenize(question)
          words.extend(tokens)
          docs_questions.append(tokens)
          docs_intents.append(intent['tag'])

        if intent["tag"] not in labels:
            labels.append(intent["tag"])
```

- Convert BoWs into numPy arrays

```
input = np.array(input)
```

```
bag_of_words("Hey how are you", words)

array([0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0
```

- Build the model for the chatbot

```
#resetting default settings
tf.compat.v1.reset_default_graph()
```

```
net = tflearn.input_data(shape=[None, len(training[0
```

```
net_h1 = tflearn.fully_connected(net, 8)
net_h2 = tflearn.fully_connected(net, 8)
```

```
net = tflearn.fully_connected(net, len(output[0], ac
net = tflearn.regression(net)
```

```
model = tflearn.DNN(net)
```

- Model predictions for the chatbot

```
def bag_of_words(sentence, words):
    bag = np.zeros(len(words))
    sentence_words = nltk.word_tokenize(sentence)
    sentence_words = [stemmer.stem(wod.lower()) for w

    for sw in sentence_words:
        for i,word in enumerate(words):
            if word==sw:
                bag[i] +=1

    return np.array(bag)
```

- Create a chat function for the chatbot

```
input = input("You: ")
```

- Classifying incoming questions for the chatbot

```
results_index = np.argmax(results)
```

```
#adding a confidence threshold
    if results[results_index] > 0.7:
        print(random.choice(responses))
    else:
        print("I don't quite understand. Try again o
```

- Customize your chatbot

EVALUATION:

Evaluating a chatbot involves assessing its performance, understanding user interactions, and measuring its effectiveness.There are several steps involved which is listed below.

1.Intent Recognition Accuracy:

Assess how accurately the chatbot recognizes user intents. Calculating the intent recognition rate using test data.

```
from sklearn.metrics import accuracy_score

    predicted_intents = chatbot.recognize_intents(test_user_inputs)
```

```
accuracy = accuracy_score(test_true_intents, predicted_intents)
```

2.Entity Recognition Accuracy:

Evaluating the chatbot's ability to correctly extract entities from user inputs.

```
from sklearn.metrics import classification_report

entity_results = chatbot.extract_entities(test_user_inputs)

print(classification_report(test_true_entities, entity_results))
```

3. User Satisfaction Surveys:

Collecting user feedback through surveys or feedback forms to understand user satisfaction and identify areas for improvement.

```
def collect_user_feedback():
    feedback = input("Please rate your experience from 1 (poor) to 5 (excellent): ")
    # Store feedback in a database or log file
```

4. Response Quality:

Chatbot should be reviewed responses for accuracy, relevance, and clarity

```
user_input = "Tell me a joke"

response = chatbot.generate_response(user_input)

print("User: ", user_input)

print("Bot: ", response)
```

5. Error Handling Assessment:

Testing the chatbot's ability to handle unexpected or unclear user inputs.

```
unclear_input = "jshsdadg&3£!?"

response = chatbot.generate_response(unclear_input)

print("User: ", unclear_input)

print("Bot: ", response)
```

6.Task Completion Rate:

Tracking the chatbot's ability is needed to help users accomplish their tasks.

```
def check_task_completion(user_input, expected_result):
  response = chatbot.generate_response(user_input)
  if response == expected_result:
     print("Task completed successfully.")
```