CHATBOT IN NLP WITH PYTHON USING MACHINE LEARNING

BATCH MEMBER:

M.THARUN

R.RAMAPRABAKARAN

K.VIGNESH

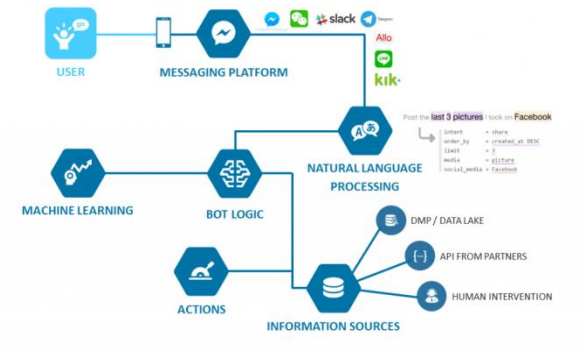
Introduction:

In this article, we will create an AI chatbot using Natural Language Processing (NLP) in Python. Our goal is to help you build a smart chatbot. First, we'll explain NLP, which helps computers understand human language. Then, we'll show you how to use AI to make a chatbot to have real conversations with people

Chatbot Machine Learning:

If you've heard about chatbots and how they work, then you probably know about machine learning as well. Quite precisely, machine learning is an integral part of what makes chatbot function 24/7/365. There is no wonder that machine learning is the future of chatbot.

But, the real question is, do you know anything about chatbot machine learning?



A chatbot developed using machine learning algorithms is called chatbot machine learning. In such a case, a chatbot learns everything from its data and human-to-human dialogues, the details of which are fed by machine learning codes.

Thanks to machine learning, chatbots can now be trained to develop their consciousness, and you can teach them to converse with people as well. One of the general reasons why chatbots have made such prominence in the market is because of their ability to drive a human to human conversations. However, all the tricks pulled up a chatbot depends on the datasets and algorithms used. The more datasets you have, the better is the effectiveness of machine learning and the more conversational chatbot you'll develop.

The concept behind developing a chatbot using machine learning

Before we get into how to develop a machine learning chatbot, we need to know a little about the design concepts.

Chatbots use data as fuel, which, in turn, is provided by machine learning.

However, feeding data to a chatbot isn't about gathering or downloading any large dataset; you can create your dataset to train the model. Now, to code such a chatbot, you need to understand what its intents are.

Wondering what does intent mean?

The intent is the intention of the user behind creating a chatbot. It denotes the idea behind each message that a chatbot receives from a particular user. So, when you know the group of customers you want the chatbot to interact with, you possess a clearer idea of how to develop a chatbot, the type of data that it encompasses, and code a chatbot solution that works. The intent of a chatbot varies from one solution to another.

That said, it is necessary to understand the intent behind your chatbot in relevance to the domain that you are creating it for.

For example, if you are building a Shopify chatbot you will intend to provide a seamless experience for all the customers visiting your website or app. By using correct machine learning for your chatbot will not only improve the customer experiences but will also enhance your sales.

Thinking about why intent is essential?

It is because intent answers questions, search for the customer base, and perform actions to continue conversations with the user. A chatbot needs to understand the intent behind a user's query. Once you know the idea behind a question, responding to it becomes easy. And that defines the gist of the human-human conversation.

Next question, how can you make your chatbot understand the intent of a user and provide accurate answers at the same time?

Well, you can define different intents and make training samples for those intents, and then train a chatbot based on that model using the particular training sample data.

How to prepare Chatbot Deep Learning?

Another pivotal question to address is how to develop a chatbot machine learning.

Or exactly how does it chatbot service work for your business?

Well, the answer is quite simple.

When you are creating a chatbot, your goal should be only towards building a product that requires minimal or no human interference.

The following steps detail the development process for a deep learning chatbot.

Accumulating data:

The first step to any machine learning related process is to prepare data. You can use thousands of existing interactions between customers and similarly train your chatbot. These data sets need to be detailed and varied, cover all the popular conversational topics, and include human interactions. The central idea, there need to be data points for your chatbot machine learning.

This process is called data ontology creation, and your sole goal in this process is to collect as many interactions as you can.

Reforming the data:

Not a mandatory step, but depending on your data source, you might have to segregate your data and reshape it into single rows of insights and observations. These are called message-response pairs added to a classifier.

The central idea of this conversation is to set a response to a conversation. Post that, all of the incoming dialogues will be used as textual indicators, predicting the response of the chatbot in regards to a question.

Perhaps, you need to set certain restrictions when creating this message-to-message board, like the conversation has to be between two people only, individual messages sent can be assigned a collective unit, and the response to a message must come within 5 minutes.

Once you have reformed your message board, the conversation would look like a genuine conversation between two humans, nullifying the machine aspect of a chatbot.

All in all, post data collection, you need to refine it for text exchanges that can help you chatbot development process after removing URLs, image references, stop words, etc. Moreover, the conversation pattern you pick will define the chatbot's response system. So, you need to precise in what you want it to talk about and in what tone.

Adding language to your chatbot:

Pre-processing is a state in which you teach your chatbot the nuances of the English Language, keeping in mind the form of English practiced in the region while taking special note of the grammar, spelling, punctuations, notations, etc. So, the chatbot could respond to questions that might be grammatically incorrect by understanding the meaning behind the context.

This process involves several sub-processes such as tokenizing, stemming, and lemmatizing of the chats. The meaning of this process in layman's language is to refine the chatbots for their readability quotient through machine learning features. In this step, you need to employ several tools all to process the data collected, create parse trees of the chats, and improve its technical language through Machine Learning.

The type of chatbot you want to code:

Once you're collected, refined, and formatted the data, you need to brainstorm as to the type of chatbot you want to develop.

Broadly, there are two different types of chatbots:

Generative chatbot: Dependent on the Generative model, this chatbot type doesn't use any conversational repository. It is an advanced form of chatbot employing Machine Learning to respond to user queries. Most of the modern chatbots function as per the generative model. Therefore, they can answer almost all types of questions. Moreover, they cover the human quotient in their conversations. Therefore, a chatbot deep learning is more adaptable to the

queries of its customers but should not be mistaken to imitate humanness in their conversation patterns. For initial chatbot developers, perfecting their art of chatbot development using this model is a tedious task and requires years of Machine Learning studies.

Retrieval-based chatbot: This type of chatbot uses a pre-defined repository to solve queries. These can only answer some questions and can provide the same answer for two different questions. However, you need to select the response system for the chatbot to comply. A retrieval-model based chatbot seldom makes any mistakes since it consults a database for user questions and provides answers accordingly. That said, it has its limitations since it cannot answer rigid questions and might not seem human. Most of the websites disguise a retrieval chatbot as their live agent since they are simple to code and create. Although traditional, such a chatbot does keep track of a user's previous messages but can only answer questions that are straightforward and not complicated.

Creating a table of word vectors:

Word vectors include popular acronyms like LOL, LMAO, TTYL, etc. These are not a part of any conversation datasets but majorly used on social media and other personal forms of conversation.

You can create your list of word vectors or look for tools online that can do it for you. Developed chatbot using deep learning python use the programming language for these word vectors.

Tabulating a Seq2Seq model:

For this step, you need someone well-versed with Python and TensorFlow details. To create a seq2seq model, you need to code a Python script for your machine learning chatbot. You can even outsource Python development module to a company offering such services. Once you know how to do so, you can use the TensorFlow function.

Track the development of your chatbot:

Post developing a Seq2Seq model, track the training process of your chatbot. You can study your chatbot at different corners of the input string, test their outputs to specific questions about your business, and improve the structure of the chatbot in the process.

With time, chatbot deep learning will be able to complete the sentences while following the orders of spelling, grammar, and punctuation.

Incorporating a chatbot into the app or website:

When you've fed data to the chatbot, tested them as per the Seq2Seq model, you need to launch it at a location where it can interact with people.

Now, you have two options over here: either you can launch your website on the app or website or soft launch it only to a group of people who will interact with the chatbot and test its answering and response abilities.

Set up a server, install Node, create a folder, and commence your new Node project. If required, install the other Node dependencies as well. The following code from HackerNoon will help you

to install the needed Node dependencies and parameters. Set up the chatbot as per the mentioned comments and customize it accordingly.

```
npm install express request body-parser –save
The next step is to create an index.js file and authenticate the bot by:
'use strict'
const express = require('express')
const bodyParser = require('body-parser')
const request = require('request')
const app = express()
app.set('port', (process.env.PORT || 5000))
// Process application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({extended: false}))
// Process application/json
app.use(bodyParser.json())
// Index route
app.get('/', function (req, res) {
        res.send('Hello world, I am a chat bot')
})
// for Facebook verification
app.get('/webhook/', function (req, res) {
        if (req.query['hub.verify_token'] === 'my_voice_is_my_password_verify_me') {
                res.send(req.query['hub.challenge'])
        }
        res.send('Error, wrong token')
})
// Spin up the server
app.listen(app.get('port'), function() {
        console.log('running on port', app.get('port'))
})
Create a file and name it Procfile. Paste the following in it:
web: node index.js
After committing all the code with Git, you can create a new Heroku instance by:
git init
git add .
git commit –message "hello world"
heroku create
git push heroku master
```

Test your machine learning chatbot:

Probably one of the tensest and stressful situations for a developer, the final step is to test your chatbot machine learning and **evaluate its success in the market. You need to assess its conversational model and check the chatbot on the following targets:**

- The accuracy and precision of the chatbot model
- The worth of a generative model based chatbot
- Testing the Seq2Seq evaluation sequence score for BLEU

You can test the chatbot's responses to the said target metrics and correlate with the human judgment of the appropriateness of the reply provided in a particular context. Wrong answers or unrelated responses receive a low score, thereby requesting the inclusion of new databases to the chatbot's training procedure.

Consistently improve the chatbot's database:

Once you have interacted with your chatbot machine learning, you will gain tremendous insights in terms of improvement, thereby rendering effective conversations. Adding more datasets to your chatbot is one way you can improve your conversational skills and provide a variety of answers in response to queries based on the scenarios.

A chatbot should be able to differentiate between conversations with the same user. For that, you need to take care of the encoder and the decoder messages and their correlation. Add hyperparameters like LSTM layers, LSTM units, training iterations, optimizer choice, etc., to it.

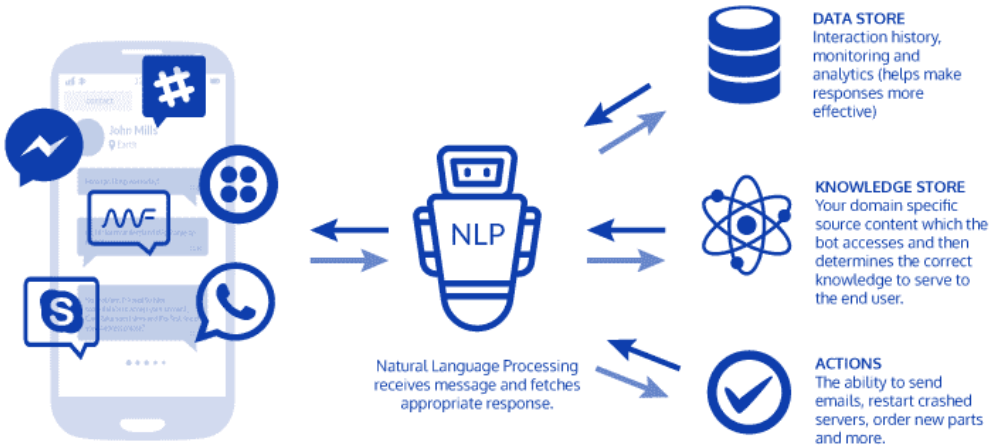
Challenges associated with chatbot machine learning

Many people agree that **chatbot machine learning** prepares the best bots that are useful in general and routine tasks. Moreover, since live agents aren't available all the time, these conversational agents can take up the lead and chat with people and perform all the actions you want them to.

Introduction to AI Chatbot

Were you ever curious as to how to build a talking ChatBot with Python and also have a conversation with your own personal AI?

As the topic suggests we are here to help you have a conversation with your AI today. To have a conversation with your AI, you need a few pre-trained tools which can help you build an AI chatbot system. In this article, we will guide you to combine speech recognition processes with an artificial intelligence algorithm.



Natural Language Processing or NLP is a prerequisite for our project. NLP allows computers and algorithms to understand human interactions via various languages. In order to process a large amount of natural language data, an AI will definitely need NLP or Natural Language Processing. Currently, we have a number of NLP research ongoing in order to improve the AI chatbots and help them understand the complicated nuances and undertones of human conversations.

Chatbots are applications that businesses and other entities employ to facilitate automated conversations between AI and humans. These conversations can occur through text or speech. Chatbots must comprehend and imitate human conversation when engaging with people worldwide. Chatbots have made significant progress from ELIZA, the first-ever chatbot, to today's Amazon ALEXA. This tutorial will cover all the fundamental concepts necessary for creating a basic chatbot that can comprehend and respond to human interactions. We will utilize speech recognition APIs and pre-trained Transformer models.

What is NLP?

NLP, or Natural Language Processing, stands for teaching machines to understand human speech and spoken words. NLP combines computational linguistics, which involves rule-based modeling of human language, with intelligent algorithms like statistical, machine, and deep learning algorithms. Together, these technologies create the smart voice assistants and chatbots we use daily.

In human speech, there are various errors, differences, and unique intonations. NLP technology empowers machines to rapidly understand, process, and respond to large volumes of text in real-time. You've likely encountered NLP in voice-guided GPS apps, virtual assistants, speech-to-text note creation apps, and other chatbots that offer app support in your everyday life. In the business world, NLP is instrumental in streamlining processes, monitoring employee productivity, and enhancing sales and after-sales efficiency.

Tasks in NLP

Interpreting and responding to human speech presents numerous challenges, as discussed in this article. Humans take years to conquer these challenges when learning a new language from scratch. Programmers have integrated various functions into NLP technology to tackle these hurdles and create practical tools for understanding human speech, processing it, and generating suitable responses.

NLP tasks involve breaking down human text and audio signals from voice data in ways that computers can analyze and convert into comprehensible data. Some of the tasks in NLP data ingestion include:

1. **Speech Recognition:** This process involves converting speech into text, a crucial step in speech analysis. Within speech recognition, there is a subprocess called speech tagging, which allows a computer to break down speech and add context, accents, or other speech attributes.
2. **Word Sense Disambiguation:** In human speech, a word can have multiple meanings. Word sense disambiguation is a semantic analysis that selects the most appropriate meaning for a word based on its context. For instance, it helps determine whether a word functions as a verb or a pronoun.
3. **Named Entity Recognition (NER):** NER identifies words and phrases as specific entities, such as recognizing “Dev” as a person’s name or “America” as the name of a country.
4. **Sentiment Analysis:** Human speech often contains sentiments and undertones. Extracting these nuances and hidden emotions, like attitude, sarcasm, fear, or joy, is one of the most challenging tasks undertaken by NLP processes.

Types of AI Chatbots

Chatbots are a relatively recent concept and despite having a huge number of programs and NLP tools, we basically have just two different categories of chatbots based on the NLP technology that they utilize. These two types of chatbots are as follows:

Scripted Chatbots

Scripted chatbots are chatbots that operate based on pre-determined scripts stored in their library. When a user inputs a query, or in the case of chatbots with speech-to-text conversion modules, speaks a query, the chatbot replies according to the predefined script within its library. One drawback of this type of chatbot is that users must structure their queries very precisely, using comma-separated commands or other regular expressions, to facilitate string analysis and understanding. This makes it challenging to integrate these chatbots with NLP-supported speech-to-text conversion modules, and they are rarely suitable for conversion into intelligent virtual assistants.

Artificially Intelligent Chatbots

Artificially intelligent chatbots, as the name suggests, are designed to mimic human-like traits and responses. NLP (Natural Language Processing) plays a significant role in enabling these chatbots to understand the nuances and subtleties of human conversation. When NLP is combined with artificial intelligence, it results in truly intelligent chatbots capable of responding to nuanced questions and learning from each interaction to provide improved responses in the future. AI chatbots find applications in various platforms, including automated chat support and virtual assistants designed to assist with tasks like recommending songs or restaurants.

In the current world, computers are not just machines celebrated for their calculation powers. Today, the need of the hour is interactive and intelligent machines that can be used by all human beings alike. For this, computers need to be able to understand human speech and its differences.

NLP technologies have made it possible for machines to intelligently decipher human text and actually respond to it as well. However, communication amongst humans is not a simple affair. There are a lot of undertones dialects and complicated wording that makes it difficult to create a perfect chatbot or virtual assistant that can understand and respond to every human.

To overcome the problem of chaotic speech, developers have had to face a few key hurdles which need to be explored in order to keep improving these chatbots. To understand these hurdles or problems we need to under how NLP works to convert human speech into something an algorithm or AI understands. Here’s a list of snags that a chatbot hits whenever users are trying to interact with it:

1. Synonyms, homonyms, slang
2. Misspellings
3. Abbreviations
4. Complex punctuation rules
5. Accents, dialects and speech differences with the age and other issues of humans. (for eg. lisps, drawls, etc)

To a human brain, all of this seems really simple as we have grown and developed in the presence of all of these speech modulations and rules. However, the process of training an AI chatbot is similar to a human trying to learn an entirely new language from scratch. The different meanings tagged with intonation, context, voice modulation, etc are difficult for a machine or algorithm to process and then respond to. NLP technologies are constantly evolving to create the best tech to help machines understand these differences and nuances better.

Installing Packages required to Build AI Chatbot

We will begin by installing a few libraries which are as follows :

Code:

```
# To be able to convert text to Speech
! pip install SpeechRecognition  #(3.8.1)
#To convey the Speech to text and also speak it out
!pip install gTTS  #(2.2.3)
# To install our language model
!pip install transformers  #(4.11.3)
!pip install tensorflow #(2.6.0, or pytorch)
```

We will start by importing some basic functions:

```
import numpy as np
```

We will begin by creating an empty class which we will build step by step. To build the chatbot, we would need to execute the full script. The name of the bot will be " Dev"

```
# Beginning of the AI
class ChatBot():
    def __init__(self, name):
        print("----- starting up", name, "-----")
        self.name = name
# Execute the AI
if __name__ == "__main__":
    ai = ChatBot(name="Dev")
```

Output :

What is Speech Recognition?

NLP or Natural Language Processing has a number of subfields as conversation and speech are tough for computers to interpret and respond to. One such subfield of NLP is Speech Recognition. Speech Recognition works with methods and technologies to enable recognition and translation of human spoken languages into something that the computer or AI can understand and respond to.

For computers, understanding numbers is easier than understanding words and speech. When the first few speech recognition systems were being created, IBM Shoebox was the first to get decent success with understanding and responding to a select few English words. Today, we have a number of successful examples which understand myriad languages and respond in the correct dialect and language as the human interacting with it. Most of this success is through the SpeechRecognition library.

:

Code:

```
import speech_recognition as sr
def speech_to_text(self):
    recognizer = sr.Recognizer()
    with sr.Microphone() as mic:
         print("listening...")
         audio = recognizer.listen(mic)
    try:
         self.text = recognizer.recognize_google(audio)
         print("me --> ", self.text)
    except:
         print("me -->  ERROR")
```

Note: The first task that our chatbot must work for is the speech to text conversion. Basically, this involves converting the voice or audio signals into text data. In summary, the chatbot actually ‘listens’ to your speech and compiles a text file containing everything it could decipher from your speech. You can test the codes by running them and trying to say something aloud. It should optimally capture your audio signals and convert them into text.

Speech to Text Conversion

```
# Execute the AI
if __name__ == "__main__":
     ai = ChatBot(name="Dev")
     while True:
         ai.speech_to_text()
```

Output :

```
----- starting up Dev -----
listening...
me --> Try to say something
```

Processing Suitable Responses

Next, our AI needs to be able to respond to the audio signals that you gave to it. In simpler words, our chatbot has received the input. Now, it must process it and come up with suitable responses and be able to give output or response to the human speech interaction. To follow along, please add the following function as shown below. This method ensures that the chatbot will be activated by speaking its name. When you say “Hey Dev” or “Hello Dev” the bot will become active.

Code:

```
def wake_up(self, text):
    return True if self.name in text.lower() else False
```

As a cue, we give the chatbot the ability to recognize its name and use that as a marker to capture the following speech and respond to it accordingly. This is done to make sure that the chatbot doesn’t respond to everything that the humans are saying within its ‘hearing’ range. In simpler words, you wouldn’t want your chatbot to always listen in and partake in every single conversation. Hence, we create a function that allows the chatbot to recognize its name and respond to any speech that follows after its name is called.

Fine-tuning Bot Responses

After the chatbot hears its name, it will formulate a response accordingly and say something back. For this, the chatbot requires a text-to-speech module as well. Here, we will be using GTTS or Google Text to Speech library to save mp3 files on the file system which can be easily played back.

The following functionality needs to be added to our class so that the bot can respond back

Code:

```
from gtts import gTTS
import os
@staticmethod
def text_to_speech(text):
    print("AI --> ", text)
    speaker = gTTS(text=text, lang="en", slow=False)
    speaker.save("res.mp3")
    os.system("start res.mp3")  #if you have a macbook->afplay or for windows use-
>start
    os.remove("res.mp3")
```

Code :

```
#Those two functions can be used like this
# Execute the AI
if __name__ == "__main__":
     ai = ChatBot(name="Dev")
     while True:
         ai.speech_to_text()
         ## wake up
         if ai.wake_up(ai.text) is True:
             res = "Hello I am Dev the AI, what can I do for you?"
         ai.text_to_speech(res)
```

Output :

```
----- starting up Dev -----
listening...
me --> Hey Dev
AI --> Hello I am Dev the AI, what can I do for you?
```

Next, we can consider upgrading our chatbot to do simple commands like some o the virtual assistants help you to do. An example of such a task would be to equip the chatbot to be able to answer correctly whenever the user asks for the current time. To add this function to the chatbot class, follow along with the code given below:

Code:

```
import datetime
@staticmethod
def action_time():
    return datetime.datetime.now().time().strftime('%H:%M')
#and run the script after adding the above function to the AI class
# Run the AI
if __name__ == "__main__":
ai = ChatBot(name="Dev")
while True:
```

```
        ai.speech_to_text()
        ## waking up
        if ai.wake_up(ai.text) is True:
            res = "Hello I am Dev the AI, what can I do for you?"
        ## do any action
        elif "time" in ai.text:
            res = ai.action_time()
        ## respond politely
        elif any(i in ai.text for i in ["thank","thanks"]):
            res = np.random.choice(
                ["you're welcome!","anytime!",
                 "no problem!","cool!",
                 "I'm here if you need me!","peace out!"])
        ai.text_to_speech(res)
```

Output :

```
----- starting up Dev -----
listening...
me --> Hey Dev
AI --> Hello I am Dev the AI, what can I do for you?
listening...
me --> What is the time
AI --> 17:30
listening...
me --> thanks
AI --> cool!
listening...
```

After all of the functions that we have added to our chatbot, it can now use speech recognition techniques to respond to speech cues and reply with predetermined responses. However, our chatbot is still not very intelligent in terms of responding to anything that is not predetermined or preset. It is now time to incorporate artificial intelligence into our chatbot to create intelligent responses to human speech interactions with the chatbot or the ML model trained using NLP or Natural Language Processing.

The Language Model for AI Chatbot

Here, we will use a Transformer Language Model for our chatbot. This model was presented by Google and it replaced the earlier traditional sequence to sequence models with attention mechanisms. This language model dynamically understands speech and its undertones. Hence, the model easily performs NLP tasks. Some of the

most popularly used language models are Google's BERT and OpenAI's GPT. These models have multidisciplinary functionalities and billions of parameters which helps to improve the chatbot and make it truly intelligent.

Image 5

This is where the chatbot becomes intelligent and not just a scripted bot that will be ready to handle any test thrown at them. The main package that we will be using in our code here is the Transformers package provided by HuggingFace. This tool is popular amongst developers as it provides tools that are pre-trained and ready to work with a variety of NLP tasks. In the code below, we have specifically used the DialogGPT trained and created by Microsoft based on millions of conversations and ongoing chats on the Reddit platform in a given interval of time.

Code:

```
import transformers
nlp = transformers.pipeline("conversational",
                            model="microsoft/DialoGPT-medium")
#Time to try it out
input_text = "hello!"
nlp(transformers.Conversation(input_text), pad_token_id=50256)
```

Reminder: Don't forget to provide the pad_token_id as the current version of the library we are using in our code raises a warning when this is not specified. What you can do to avoid this warning is to add this as a parameter.

nlp(transformers.Conversation(input_text), pad_token_id=50256)

You will get a whole conversation as the pipeline output and hence you need to extract only the response of the chatbot here.

Code:

```
chat = nlp(transformers.Conversation(ai.text), pad_token_id=50256)
res = str(chat)
res = res[res.find("bot >> ")+6:].strip()
```

Finally, we're ready to run the Chatbot and have a fun conversation with our AI. Here's the full code:

```
----- starting up Dev -----
listening...
me --> Hello!
AI --> Hello :D
listening...
```

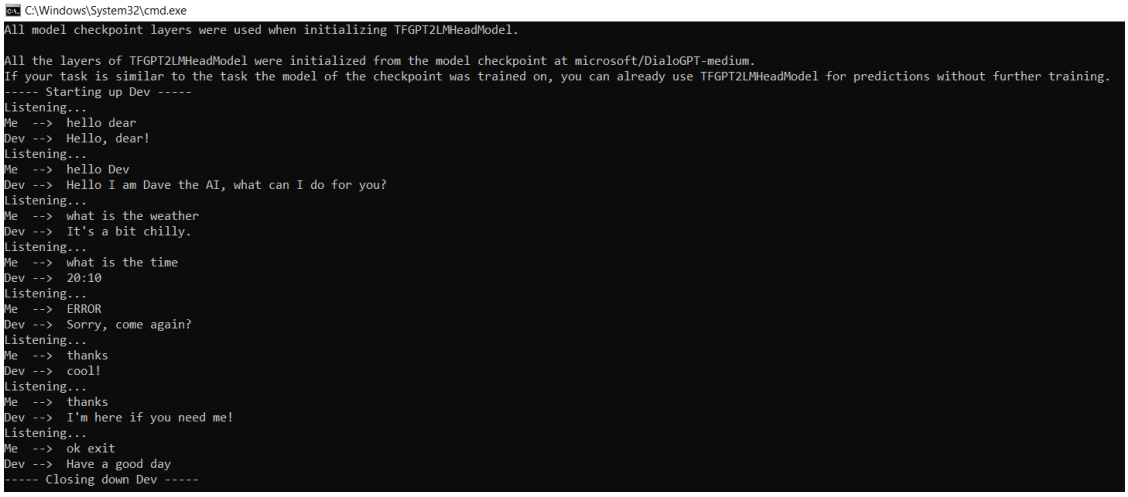
Great! The bot can both perform some specific tasks like a virtual assistant (i.e. saying the time when asked) and have casual conversations. And if you think that Artificial Intelligence is here to stay, she agrees:

Final Code:

```
# for speech-to-text
import speech_recognition as sr
# for text-to-speech
from gtts import gTTS
# for language model
import transformers
import os
import time
# for data
import os
import datetime
import numpy as np
# Building the AI
class ChatBot():
    def __init__(self, name):
        print("----- Starting up", name, "-----")
        self.name = name
    def speech_to_text(self):
        recognizer = sr.Recognizer()
        with sr.Microphone() as mic:
            print("Listening...")
            audio = recognizer.listen(mic)
            self.text="ERROR"
        try:
            self.text = recognizer.recognize_google(audio)
            print("Me  --> ", self.text)
        except:
            print("Me  -->  ERROR")
    @staticmethod
    def text_to_speech(text):
        print("Dev --> ", text)
        speaker = gTTS(text=text, lang="en", slow=False)
        speaker.save("res.mp3")
        statbuf = os.stat("res.mp3")
       mbytes = statbuf.st_size / 1024
```

```
        duration = mbytes / 200
        os.system('start res.mp3')  #if you are using mac->afplay or else for
windows->start
        # os.system("close res.mp3")
        time.sleep(int(50*duration))
        os.remove("res.mp3")
    def wake_up(self, text):
        return True if self.name in text.lower() else False
    @staticmethod
    def action_time():
        return datetime.datetime.now().time().strftime('%H:%M')
# Running the AI
if __name__ == "__main__":
    ai = ChatBot(name="dev")
    nlp = transformers.pipeline("conversational", model="microsoft/DialoGPT-medium")
    os.environ["TOKENIZERS_PARALLELISM"] = "true"
    ex=True
    while ex:
        ai.speech_to_text()
        ## wake up
        if ai.wake_up(ai.text) is True:
            res = "Hello I am Dave the AI, what can I do for you?"
        ## action time
        elif "time" in ai.text:
            res = ai.action_time()
        ## respond politely
        elif any(i in ai.text for i in ["thank","thanks"]):
            res = np.random.choice(["you're welcome!","anytime!","no
problem!","cool!","I'm here if you need me!","mention not"])
        elif any(i in ai.text for i in ["exit","close"]):
            res = np.random.choice(["Tata","Have a good day","Bye","Goodbye","Hope to
meet soon","peace out!"])
            ex=False
        ## conversation
        else:
            if ai.text=="ERROR":
                res="Sorry, come again?"
            else:
                chat = nlp(transformers.Conversation(ai.text), pad_token_id=50256)
                res = str(chat)
                res = res[res.find("bot >> ")+6:].strip()
        ai.text_to_speech(res)
    print("----- Closing down Dev -----")
```

Output:



Note: I had later switched from google collab to my local machine due to some module issues which I faced during implementation and hence I am sharing my experience here so that if any of you also face the same issue can solve it. Obviously, Google is also there but the following lines will explain the issue. I used Python 3.9 as it had all the modules necessary and Python 3.6 and older versions will also work. Python 3.8 or the latest version might not have all the modules ported to match the version and hence I would suggest using Python 3.9 or older versions than 3.6.

To run a file and install the module, use the command “python3.9” and “pip3.9” respectively if you have more than one version of python for development purposes. “PyAudio” is another troublesome module and you need to manually google and find the correct “.whl” file for your version of Python and install it using pip.

Conclusion:

In this guide, we've provided a step-by-step tutorial for creating a conversational chatbot. You can use this chatbot as a foundation for developing one that communicates like a human. The code samples we've shared are versatile and can serve as building blocks for similar chatbot projects.

It's a great way to enhance your data science expertise and broaden your capabilities. With the help of speech recognition tools and NLP technology, we've covered the processes of converting text to speech and vice versa. We've also demonstrated using pre-trained Transformers language models to make your chatbot intelligent rather than scripted.