

**GOVERNMENT COLLEGE OF ENGINEERING BARGUR**

**( AUTONOMOUS)**

**PROJECT TITLE: CHATBOT DEPLOYMENT WITH IBM CLOUD WATSON ASSISTANT**

**TEAM MEMBERS:**

**CHANDRU D**

**NAVINKUMAR N**

**MANIKANDAN V**

**BHARATH P**

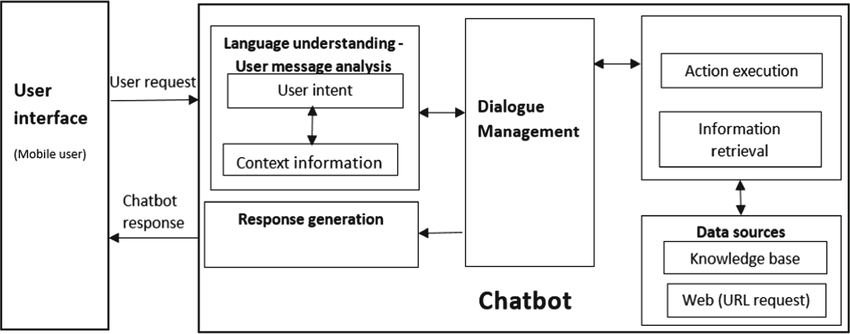
**JAISURYA G**

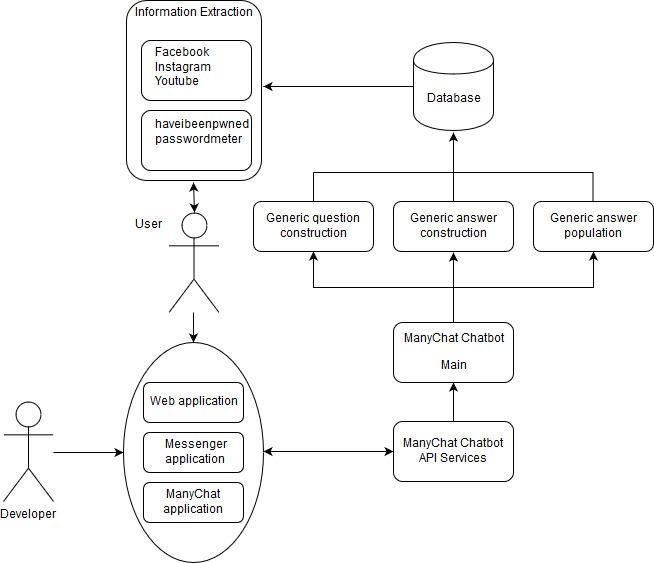
Design, architecture and code implementation:

Design:

There are two approaches that can be used to develop a chatbot depending on the algorithms and techniques adopted: rule-based approach and machine learning approach. Rule-based A rule-based chatbot processes information and provides responses based on a set of predefined rules with the use of pattern matching algorithms. Although the pattern matching techniques vary in complexity, the basic idea is the same. The user input is classified as a pattern, and the chatbot selects a predefined answer by matching the pattern with a set of stored responses. The pattern and response matching algorithms are handcrafted [65]. Pattern matching is adopted by many chatbots and is especially popular among the early chatbots like ELIZA, PARRY, and ALICE. The advantage of the rule-based approach is its speed as it does not require any deep analysis of the input text . However, the responses are repeated and lack flexibility and originality as the knowledge is set by the developer in advance [78]. The following paragraphs will provide an overview of the three most commonly used languages for the implementation of rule-based chatbots. Artificial Intelligence Mark-up Language (AIML) is a derivation of Extensible Mark-up Language (XML) . ALICE was the first chatbot implemented in the AIML language . AIML has a class of data object called an AIML objects, and these objects are responsible for modeling conversational pattern. Each object consists of two units called topics and categories. A topic is an optional top-level element that has a name attribute and a set of categories associated with it. Categories are the most basic unit of knowledge and are the rules of the chatbot. Each category consists of two elements called pattern and template. The pattern matches against the input from the user to the template that contains the response of the chatbot . AIML is simple, flexible, and highly maintainable, and thus is one of the most commonly adopted languages for chatbot development . The biggest disadvantages of AIML are that the developer must specify a pattern for every possible input of the user and that it is inefficient when the knowledge base is large . The structure of an AIML object is demonstrated as below: User Input Corresponding Response to Input . RiveScript is a line-based scripting language that can be used to implement the Knowledge Base . Compared to AIML, RiveScript has more built-in features and tags, which means that the writer does not need to specify information about the chatbot in the additional configuration files. ChatScript is an open-source language for developing rule-based chatbots. It matches user inputs to outputs using rules created by human writers in program scripts through a process called dialog flow scripting . ChatScript uses concepts that are set of words that have the same meaning. 8 It consists of 2000 predefined concepts and scripters can also write their own concepts easily . Compared to AIML and RiveScript, ChatScript is a harder language, but it allows developers to combine rules in more complicated ways.

Architecture:





Implementation details:

The project consists of several parts, the most important of which are: 1- Actions section, which is the folder that contains 3 files in the Python programming language.The most important file is the Actions file, which contains custom Actions that are built according to the need and purpose of the chatbot. 2- The data section and contains 3 important files that cannot be dispensed with: the natural language understanding file, which contains the training data necessary for the bot, which it is expected to receive during its operation from the user, and the rules file, which contains a certain structure that makes the bot act obligatory according to what exists, regardless of the circumstances in terms of the received data, and the story file, which contains scenarios of conversations with users, and all conversations are recorded within this file in .yml extension. 3- There is also a very important file, which is the domain that defines the universe in which your assistant operates. It specifies the intents, entities, slots, responses, forms, and actions your bot should know about. It also defines a configuration for conversation sessions. 4- There are also configuration files, endpoints, and credentials that are responsible for the overall properties and some permissions allowed for the bot and linking with chat channels such as Slack and Facebook Messenger. 5- There is also the Models section, in which all models are stored after each bot training. Every model we can say is like the nucleus or brain of the bot. The bot cannot work and listen to the user’s messages and respond to them without the model. After each bot training process, and to get the latest results, you must choose the newest model. Older models can be selected so that they can be compared with the new model in terms of additions.

Code implementation:

# 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, "-----")

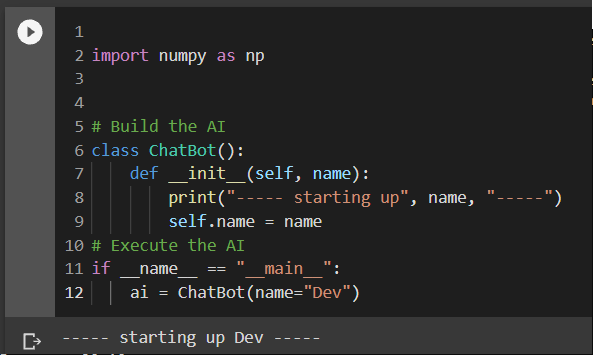
self.name = name

# Execute the AI

**if** \_\_name\_\_ == "\_\_main\_\_":

ai = **ChatBot**(name="Dev")

**Output :**

 **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.

Using Google APIs

To use popular Google APIs we will use the following code:

**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

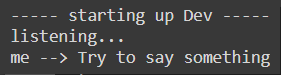
**if** \_\_name\_\_ == "\_\_main\_\_":

ai = ChatBot(name="Dev")

**while** True:

ai.speech\_to\_text()

**Output :**



**Note:** Here I am speaking and not typing

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()

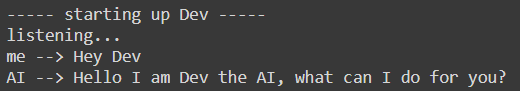
## wake up

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 :**



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(

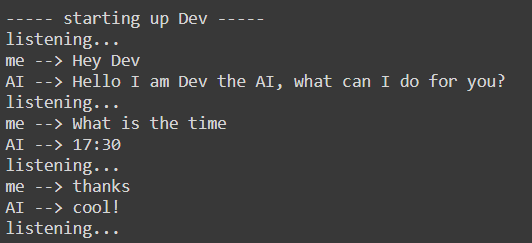
["you're welcome!","anytime!",

"no problem!","cool!",

"I'm here if you need me!","peace out!"])

ai.text\_to\_speech(res)

**Output :**



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](https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)) for our chatbot. This model was presented by Google and it replaced the earlier traditional sequence to sequence models with [attention mechanisms](https://en.wikipedia.org/wiki/Attention_(machine_learning)). 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](https://en.wikipedia.org/wiki/BERT_(language_model)) and OpenAI’s [GPT](https://en.wikipedia.org/wiki/GPT-3). These models have multidisciplinary functionalities and billions of parameters which helps to improve the chatbot and make it truly intelligent.

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](https://huggingface.co/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](https://huggingface.co/transformers/main_classes/pipelines.html) tasks. In the code below, we have specifically used the [DialogGPT](https://huggingface.co/transformers/model_doc/dialogpt.html) 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.

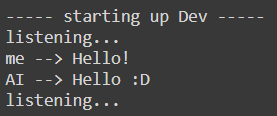
**Code:**

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:



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)

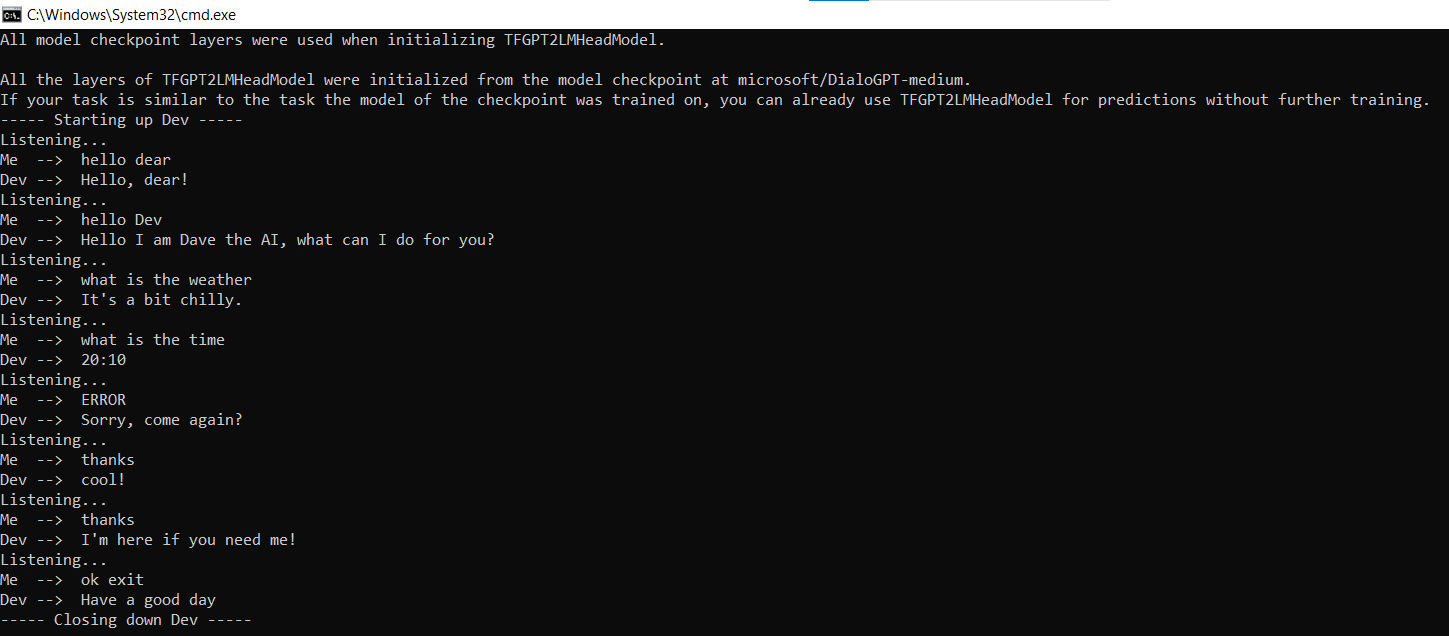
res = str(chat)

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.

The link to the full code can be found [here](https://github.com/arnabm14/Dev_AIChatbot_NLP).

**Bonus tips:** Feel free to drop a star if you liked this tutorial or bot and feel free to fork and create your own AI chatbot and call it whatever you want!

**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.