# CREATING A CHATBOT USING PYTHON

## PHASE 3 : **DEVELOPMENT PART 1**



Developing a Truly Conversational AI agent…

✝ Conversational AI has been a field of intense research for past few years, and the vast amount of convenience it promises, definitely labels it as the **NEXT BIG THING!**

✝ Much of the progress in the Natural Language Understanding can be attributed to advancements in deep learning and NLP, and convenient access to high-end computational resources.

✝ Being a Language Understanding company, building multilingual conversational agents to serve a simple motto —

As an AI language model, I can provide a general project outline for creating a chatbot in Python using AI, DAC, and IOT.

Project: Building an AI-powered Chatbot for a Smart Home using DAC and IOT

Technologies and Tools:

- Python

- TensorFlow/Keras

- Dialogflow API

- Raspberry Pi

- IOT devices (e.g., smart thermostat, security system)

- Blockchain technology

1) **Design and Planning**:

- Define the purpose and scope of the chatbot: Determine what task or problem the chatbot will address and what type of interactions it will have with users in the smart home environment.
- Identify the required hardware and software.
- Establish the infrastructure for the chatbot and configure Raspberry Pi and IOT devices.
- Design the dialogue flow and create intents and entities in Dialogflow to handle user requests.

2) **Implementing AI and DAC**:

- Implement NLP algorithms using TensorFlow/Keras.
- Train the chatbot using the collected data.
- Integrate the chatbot with DAC using blockchain technology.
- Test the chatbot's accuracy and efficiency.

3) **Implementing IOT Integration:**

2. Create suitable IOT devices that can connect with the chatbot.

3. Configure the devices to identify various commands.

4. Use Python libraries to establish a connection between the IOT, Raspberry Pi, and the chatbot.

5. **Testing and Deployment**:

- Test the chatbot for accuracy, efficiency, and usability with the integrated IOT devices.
- Deploy the chatbot in the smart home environment.
- Train the chatbot on a larger dataset to improve its accuracy and functionality.

Once the project is complete, the chatbot can interact with users in the smart home environment using voice or text and perform various tasks based on the user's request. The integration with DAC enables the chatbot to perform autonomous transactions based on predefined rules and conditions.

# DIMINISHING THE INTELLIGENCE-GAP BETWEEN HUMANS AND CONVERSATIONAL AGENTS:

○ This series of articles aims at helping you understand the underlying elements involved in chatbot design, and the process of chatbot development, by implementing a bot in python.

## Outline

- *How does a Bot think?*
- *Domain Identification*
- *Training Data for NLU*
- *Training and Evaluation*
- *Setting up NLU server*
- *How to make NLU better?*

## 1. How does a Bot think?

○ For building any conversational agent, the most crucial part is understanding what the user is conveying. *Natural Language Understanding* (**NLU**) plays an important role in this aspect. After this, to actually build a bot, a *Dialogue Framework* needs to be chosen.
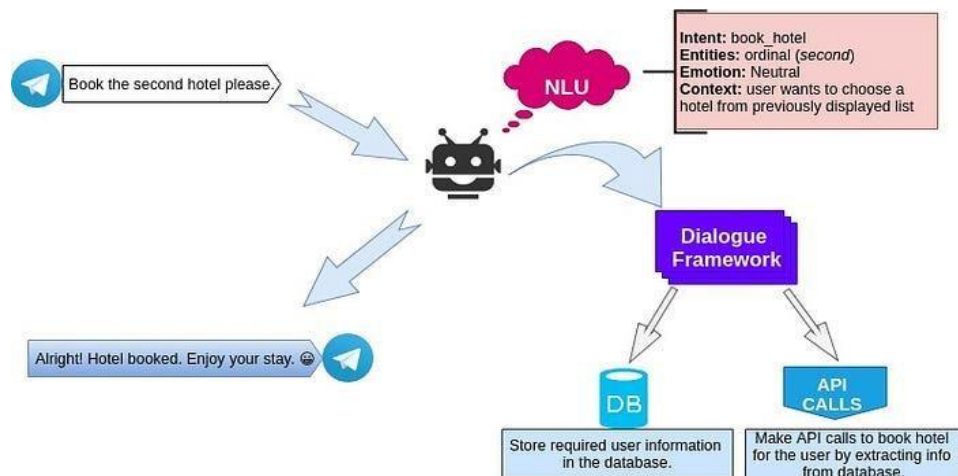
The flow of information through a Natural Language Understanding Engine and Dialog Framework

- ○ Whenever a user utters queries to our conversational agent, the first task is to classify the intention of the user utterance (*intent*) and alongside extract out important information present in the utterance (*entity*). A lot of additional tasks can be carried out in parallel, such as **emotion classification** (*to check the mood of the user*), **order request classification** and many more.

- ○ Once the conversational agent understands complete detail about the user utterance, it uses a Dialogue Framework to **predict the actions** to be carried out next. And then, that particular action is executed (may involve database fetch or API calls) to provide the desired response back to the user.

## 2. DOMAIN IDENTIFICATION:

Let us dive right into what we are eagerly waiting for…

*Let's build a bot*

*Highly caffeinated fingers ready to roll*

○ We will start with a basic version of **Payment Bot** in **English**.

○ The first step is to identify the domain of the bot. As discussed, two important aspects of NLU are *Intents* and *Entities.*

○ **Intents** are the different categories of utterances to classify what the user could mean while conversing with the bot. Whereas, **Entities** are important information present in an utterance from user.

**For example**, let the user utterance be — *"I*

*have to book a flight to Delhi."*

○ Clearly, the user intends to *book a flight*

and also has provided the *destination* which

is *Delhi.* Therefore, the following can be concluded.

*"intent"*: *"book_flight"*
*"entities"*: *{"destination" : "Delhi"}*

⭘ Let us narrow down our domain primarily focusing on a bot that can validate user's mobile number, and show the current wallet balance. There are obviously some *general intents* needed for every domain.

| General Intents | Domain Intents | Entities |
|---|---|---|
| greet | inform | mobile |
| bye | show_wallet_balance | otp |
| affirm | | |
| deny | | |
| thankyou | | |
| restart | | |
| revert | | |
| repeat | | |

**Building domain level understanding for Chatbot Development**

⭘ *At [Saarthi.ai](), we use our chatbot development platform to train the NLU models as per the required configurations. There is also a*

*data tagging tool which facilitates multiple members in the Data Team to write intent utterances conveniently tagging the entities. But, since the work is at pre-release stage, let me explain using another commonly used open-source chatbot framework 'RasaNLU' , to walk you through the process of building a bot.*

⊙ *We shall learn how to tweak some configurations in RasaNLU to make it perform better for our use case.*

## INSTALLATION:

⊙ We will be installing RasaNLU open-source framework for NLU <u>training</u>.

⊙ First make sure, your system has <u>Anaconda</u> setup with the latest stable version.

```
## create a new conda environment
$ conda create -n rasanlu python==3.6.1
$ conda activate rasanlu$ git clone https://github.com/RasaHQ/rasa_nlu.git $ cd rasa_nlu
$ pip install -r requirements.txt
$ pip install -e .$ pip install rasa_nlu[spacy]
$ python -m spacy download en_core_web_md
$ python -m spacy link en_core_web_md en $ pip install rasa_nlu[tensorflow]
```

## Always prefer cloning the projects and installing from your local file system˙
## 3. TRAINING DATA FOR NLU:
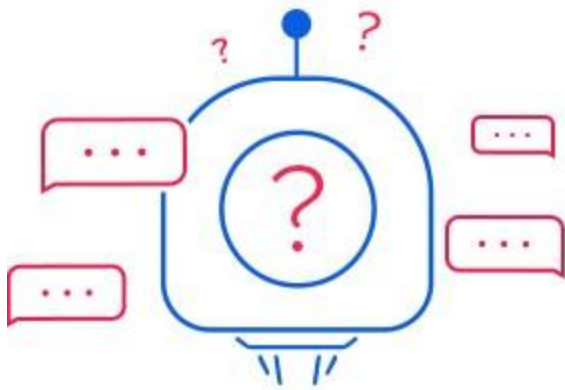
**Preparing training data for your chatbot usecase**

- After the NLU domain is finalized, enters the cumbersome (as any developer might think ) task of preparing data for training the NLU. But, once the NLU server is up and running with acceptable accuracy for classification task, a huge hindrance is avoided , which is usually coding the actions for the bot using any Dialogue Framework.
- The training data needs to be prepared in Rasa format since we are using RasaNLU here. Markdown (md) format is shown as an example.
- Please do have a look at [training data format](#) used in Rasa.

```
## intent:inform
- my mobile number [9412312345](mobile) is
- [70014512345](mobile)
- O.T.P. [2343](otp)
...## intent:show_wallet_balance
- what is my balance
- how much money is in my paytm wallet
...
```

# While writing the utterances, it is a good practice to write them as an end-user of the bot.

- Similarly, write the utterances for each intent present in the domain.

  Since we are going to use "*tensorflow embedding*" pipeline of Rasa, it is important to have **distinct utterances** of each intent in the final training data. Also, there must be at least **40 utterance examples** to make sure we have enough test data to evaluate our model.

- With these points in mind, once the dataset is finalized, You have overcome the initial hurdle. Although there will be iterations on the NLU training data, for now you are all set to move towards training and evaluating the NLU.

# 4. TRAINING AND EVALUATION:
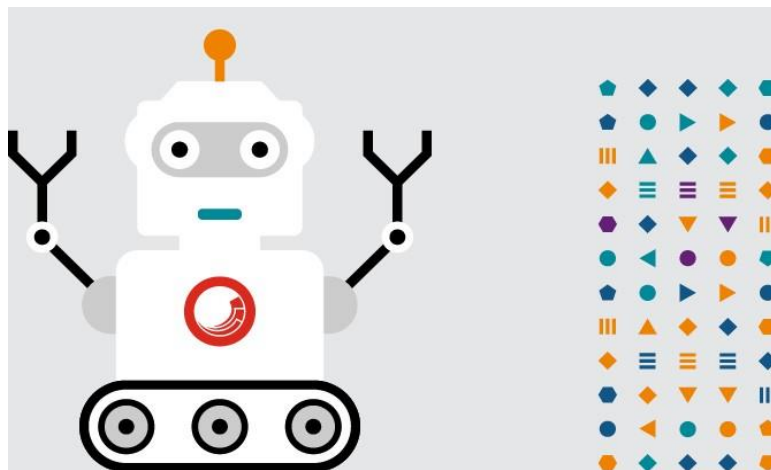


**Time to check how conversant you are!**

- Before starting the training phase, there are few hacks that needs to be done in order to train on Hindi data.

- Let us download a multi-language model from **Spacy**. Then link the short-form 'hi' (for Hindi) to this model. Make sure the NLU

environment is activated and then use the following commands shown below.

```
$ python -m spacy download xx_ent_wiki_sm
$ python -m spacy link xx_ent_wiki_sm hi
```

- We need to add a config file for training our Hindi dataset on RasaNLU. The configuration we are going to use is:
- Save the above lines as 'config_tensorflow.yml'. Move the file into *rasa_nlu → sample_configs* directory where all the other predefined config files are saved.
- Before we proceed towards training, have a look at the **token pattern** and **max_ngram**. *Token pattern* is regex defining tokens that are considered. The default regex by Rasa ignores words with single character, which might be a problem for languages apart from English. Hence, the token pattern is changed as shown above.
- *max_ngram* defines the maximum number of consecutive words that can be considered together during featurization.

Generally *max_ngram* performs well when set to 3. But there is an option to evaluate each and every model and finding out the best model.

- Now, it's time to **train** the NLU using this config file. But before that just one last step, separate out 10% of the utterances for each intent and create another test dataset for evaluation task.

```
$ python -m rasa_nlu.train \
    --config sample_configs/config_tensorflow.yml \
    --data <path to your training data file> \
```

```
    --path models \
    --fixed_model_name nlu \
    --project paytmbot --verbose
```

🞢 Once the model is successfully trained, let's **evaluate** our trained
model.
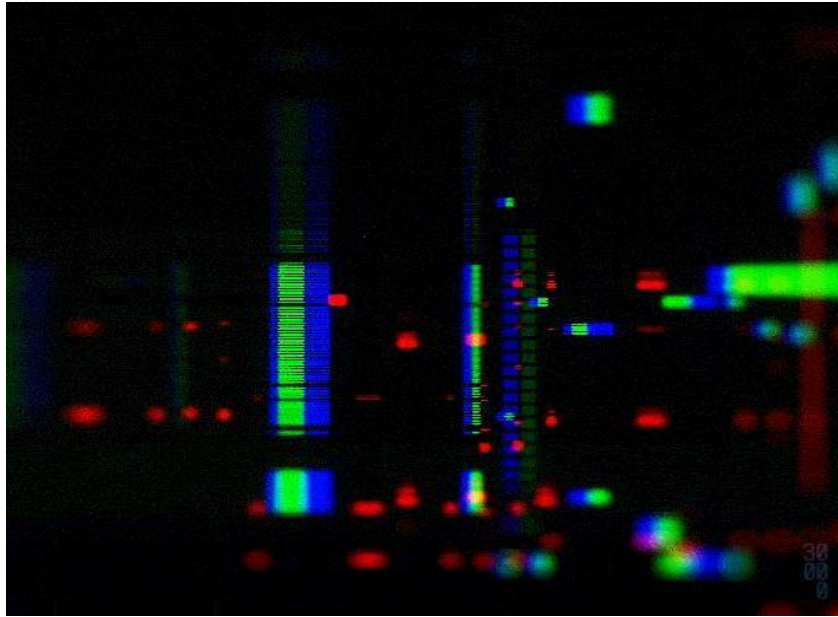
```
$ python -m rasa_nlu.evaluate \
    --data <path to test data set> \
    --model models/paytmbot/nlu
```

# You might run into TclError because of the command line. Fix it using this link.

## 5. SETTING UP  NLU  SERVER:

Great!!

✟ So, now we are almost done with our NLU. All we need is to run a
**server** locally and test some of our own utterances.

**Almost done now! Let's get the server running**.

```
$ python -m rasa_nlu.server -c sample_configs/config_tensorflow.yml --path models --port <port number>
```

✞ Open up the browser to check the NLU server.

```
localhost:<port number>/parse?q="<example_utterance>"&project=paytmbot&model=nlu
```

**You can also use curl from terminal:**

```
$ curl -XPOST localhost:<port no>/parse -d '{"q":"<example_utterance>", "project":"paytmbot", "model":"nlu"}'
```

# 6. HOW TO MAKE NLU BETTER ?

✞ Now you might have noticed that the entity *number* is not that appropriate for our use case. Because number could mean *OTP, mobile number,* or any other random number.

✞ One way to classify this is by creating an **external NLU layer** through an API which makes [duckling calls](#) to perfectly extract phone number separately. There are a lot of other types of entities that duckling can extract such as *emails, names, distance, time* and many more in multiple languages.

✞ Even if there is no external NLU layer, there is nothing to worry about. To complete the payment bot, we shall be using slots and validate the *number* entity that has been extracted to differentiate among *OTP, mobile,* etc. (*more on this in the next part*).

✞ Apart from using an API, there are various other tips that will help you to improve the performance of NLU. Needless to say, the most important component is **data**. If the training data for NLU is clean and contextual then the classification performs much better.

✞ One more thing you can do is to host the NLU server on **Apache**. It helps in reducing the lag after deployment.

✞ If you are reading this part, by now you must be having a NLU server up and running in your desired port.

## CONCLUSION:

✞ A chatbot is one of the simple ways to transport data from a computer without having to think for proper keyword to look up in a search or browse several web pages to collect information.