



**Politecnico
di Torino**

POLITECNICO DI TORINO

Master's Degree in Computer engineering (Data Analytics)

Academic Year: 2020/2021

Session Degree: December

Implementation and Evaluation of an Educational Chatbot based on NLP Techniques

Supervisors

Prof. TORCHIANO Marco

Dr. LEONARDI Simone

Candidate

CAIRONE Fiorentino

Matricola 277922

Summary

The thesis wants to suggest some solutions about the problems of remote communication in schools and universities, due also to the COVID-19 pandemic, thanks to the creation of a real time assistant that can answer the questions of the students and assist the teachers on their job.

I did research and created an educational bot to answer the students' questions in real time about PO course ("Programmazioni ad Oggetti" of POLITECNICO OF TURIN), using a framework called RASA Open Source and his evolution: RASA X, doing a linguistic analysis of dataset, composed by some real questions and answers related of arguments of Italian course of PO.

The Chatbot, also, offers a "Human Handoff" service, which allows the student to contact the teacher that can answer directly the questions proposed. All is managed through Slack which is one of the university communication channels most used by students and professors to communicate remotely.

After the creation of the Chatbot, an accurate analysis was carried out and I compared various NLP (Natural Language Programming) pretrained models: BERT, SpaCy and ConveRT used to train edu-bot in learning process.

Acknowledgments

Index of Contents

1.Introduction

1.1	The remote communication and the school.....	8
-----	----------------------------------------------	---

2.AI Assistants

2.1	What is Virtual Assistant.....	9
2.2	The advantages of AI Assistant.....	9
2.3	AI Assistant in Schools and Universities.....	10

3.Background

3.1	Slack.....	13
3.1.1	Slack Bot App.....	13
3.2	RASA Open Source.....	16
3.2.1	Natural Language Understanding.....	17
3.2.2	Dialogue Management.....	17
3.2.3	Integrations RASA on Slack.....	17
3.3	How to create, tested and run a RASA project.....	22
3.4	How write stories (rules).....	23
3.5	Principal configuration models.....	25
3.6	How create custom actions.....	27
3.6.1	Custom actions Class.....	28
3.7	Rasa X: an “evolution” of RASA.....	29
3.7.1	How install Rasa X.....	30
3.7.2	Rasa X Server.....	31
3.8	A brief introduction of Python.....	35
3.8.1	How write a class in python.....	35
3.9	A brief introduction of SQLite.....	36
3.9.1	SQLite in Python.....	36

4. Edu-Bot implementation

4.1	The problem and the goals.....	38
4.2	The dataset.....	38
4.3	The Technologies used.....	39
4.4	The AI Assistant’s architecture.....	40
4.4.1	SPACY Model.....	40

4.4.2	BERT Model.....	41
4.4.3	ConveRT Model	44
4.5	The final prototype	46
5. Evaluation		
5.1	The Evaluation metrics.....	49
5.2	The NLU Evaluation	50
5.2.1	SPACY Model Evaluation.....	50
5.2.2	BERT Model Evaluation	54
5.2.3	ConveRT Model Evaluation.....	58
5.2.4	Some considerations	62
5.3	The Conversation Evaluation	62
5.3.1	SPACY Model Evaluation.....	64
5.3.2	BERT Model Evaluation	67
5.3.3	ConveRT Model Evaluation.....	70
5.3.4	Some considerations	72
6. Conclusions		
6.1	Final considerations.....	74
References.....		77

1. Introduction

1.1 The remote communication and the school

Today technological remote communication using smartphones, laptops and tablets, caused also by the ongoing COVID-19 pandemic, are enabling a new transition in the educational world that uses different communication channels and different modes of interaction. In both schools and universities, students and professors could not have no way of communicating "face to face" in the same room, as they usually do, and this leads to problems for both sides:

- Students very often have several questions to ask their teachers and would like they answers as fast as possible
- Professors have to answer students questions but, due to the number of messages/emails they receive, they often fail to meet all the requests by solving students doubts even many days after or even not responding at all

This thesis, wants to try to solve these problems by introducing a bot in few known apps, as Slack, that allows you to answer students' questions/doubts quickly (within a few seconds) and also facilitate the work of professors, who have fewer direct messages (*eg.* through the institutional email) in order to:

- **answer the questions** proposed by the students by summarizing the content requested through a complete answer
- **offer a "Human Handoff" service**, which allows the student to contact the teacher if the bot has not answered the student's request in an exhaustive manner
- **allowing everything through a communication channel** used by the university by both students and professors as Slack

The thesis is composed to 6 chapters (including "Introduction" and "Conclusion"):

- in "*AI Assistants*" chapter will we presented some general consideration about the AI Assistants and their use in educational world
- in "*Background*" chapter will show the technologies used to create the educational Chatbot, *i.e.* RASA Open Source framework, programming language (Python), ...
- in "*Edu-Bot implementation*" chapter will present the educational bot implemented, including the models used and it functionalities
- in "*Evaluation*" will analyse the three different pre-trained models (SpaCy, BERT, ConveRT) used in learning process

The source code and dataset of the thesis can be found on the following page [1].

2. AI Assistants

2.1 What is Virtual Assistant

Virtual Assistant [2] or Artificial Intelligence (AI) Assistant is a software agent that can perform tasks or services for an individual based on commands or questions. Another term, usually used, is “**Chatbot**” and often refers to online chats. Online chat programs are used for entertainment, but in this period it could be used for instance in the schools to simulate a professors-students communications. So, AI Assistant can ask users questions, via text or voice, control automation devices or play music or send emails.

As of 2017, the virtual assistants capabilities and usage are expanding rapidly, with new products entering the market as *Amazon Alexa* or *Cortana* of *Microsoft*, but first experiments decade just in XX centuries.

Virtual assistant work via:

- **Text:** online chat, SMS, Text, e-mail or other’s text-based communication channels
- **Voice:** an example, Amazon Alexa
- By **taking** and/or **upload images**
- **Multiple methods** of previous, as Google Assistant

Virtual Assistant is based on **NLP (Natural Language Processing)** to match *i.e.* user text, voice input and execute commands or answer questions. It is based on Machine Learning techniques, for instance question-answer pre-trained models, or image recognition.

2.2 The advantages of AI Assistant

Today, the increase in usage of smartphones, laptops, or something else, has made people smarter than in the past. For this reason it is possible to use AI technologies that can help us in a few fields of our life like school, job, entertainment.

As said in previous paragraph, nowadays were created some AI Assistant that in few time have become popular and practically used to everyone: *i.e. Siri, Amazon Alexa, Google Assistant, Cortana* that try to work like human personal assistant using Machine Learning.

So, we can find these advantages [3]:

- AI Assistant can **offer a lot of services** as question-answering, send emails, make a to-do list, make phone calls, set a reminder or play music
- **Less human efforts:** you do no need to spend money on its salary, because, usually, is completely free

- **More productivity:** Assistants are faster than humans and can work twice the speed of a physical person and can give more and efficient output. Also, I want add that is completely available 24/7
- **Flexible work:** it is adaptable to human needs and commands. Usually is used a voice modulation of assistant to simulate a human voice and have the suggestion to talk (or text to) with a real person
- **Accuracy:** smart AI Assistants is often more accurate and in the future will be more and more, I want add, thanks improvements of technologies and new Machine Learning methods
- **Faster Approach:** if you have a good internet connections, the Assistant do everything for you at an instance
- **Optimize workflow:** AI Assistant is also used in industries and business companies to optimize the work. In my opinion it can used in the schools or university to improve the communication and help students and professors
- **Mobility of device:** you can used AI assistant in device as you prefer like smartphones or laptops

2.3 AI Assistant in Schools and Universities

AI Assistant, as I wrote, can be used in the educational world. This approach, in this period, also due to COVID-19 pandemic, could be used to help the teachers in their work, answering the principal students' questions about one or more arguments, sending emails, and helping in the administration. So, now, I want to talk about two examples of educational bot.

First I wanted to analyse the report of Sachin Waikar [4] that presents the research of Chris Piech, an assistant professor of computer science at Stanford, and other experts in reinforcement learning, human interaction, pedagogy and other areas, about the educational system.

Some part of this research of Stanford cite:

“According to U.N. estimates, about 69 million teachers will be needed to achieve 2030 sustainable educational goal. Today, more than 260 million children and youth do not attend school [...]. The online educational tools can help, the lack of resources is particularly challenging for open-ended tasks [...]. We need to make roles teachers play easier – especially understanding how to help students work through open-ended tasks.[...]”

The team propose an AI-based engine that “understand students” that could have a positive impact and these include, as Piech suggests, expanding the impact of teachers and taking the role of a one-on-one tutor. This “Super Teaching Assistant” could provide volunteer teachers automated, detailed reports on what exactly students want, and help the teachers to understand their students and deliver a better education. Also, this system would enable teachers to spend less time on grading and more effort on teaching.

In their first focus, the Assistant helped students learn scientific methods and coding. As Piech notices:

“[...] There are great online tools that enable to practice experimentation and see results. But there’s no tool to look at your process of learning experimental methods, assessing your understanding, and giving you feedback [...]”

Finally, the Stanford research want to suggest:

“[...] the system should be adapted to diverse learning needs and contexts, and can help train new teachers, multiplying its effects and lowering the barrier of creating scaled human-centred education. [...]”

Secondly, I want to report a research by Dyllan Furness [5] that analyses an AI teacher assistant created and used in a class of Georgia Institute of Technology. The Chabot answers the questions, teaches and helps the students to learn. Goel said in the interview:

“[...] We thought that if an A.I. TA would automatically answer routine questions that typically have crisp answers, then the (human) teaching staff could engage the students on the more open-ended questions [...]”

Some part of this interview of Digital Trends:

“A.I. is quickly integrating into every aspect of our lives [...], will alter both the face and function of education. [...] A student’s engagement with A.I. will only increase as he/she graduates through the school system. Educational A.I. toys will be replaced by tutors whose job it will be to identify subjects of weakness and facilitate additional training [...]”

“Systems like Wolfram Alpha can already answer complex math equation and queries in language that’s informative and accessible.[...] These digital learning partners are meant to support teachers rather than replace them.”

This system is not designed to replace humans, as Thilo Michael said:

“[...] the system is able to answer pragmatic questions about the courses and majors available, but is not able to answer questions on a broader level. I think the system could very well be used in combination with counselling to have the best of both worlds. [...]”

So, with this two articles, I want to resume some very useful advantages, adding at the previous describe in 2.2 paragraph, of use of AI Assistant in the education world:

- We can **improve the teaching**, help teachers to graduate, divide the effort, adapt the learning based on needed and context, understand the students request in better way
- Try to **take the possibility to learn at all people**, also if they do not go in the school

- **Help students in learning** and teach some helpful methods of study in the different subjects such as scientific methods or coding ...
- ... but **without deleted the “human teacher variable”** that is fundamental to give feedbacks, train the bot and help students to understand the concept

3. Background

3.1 Slack

For this project, *Slack* [6] was chosen as bot application channels. It is one of the business collaboration tools used to send instant messages to team members. It was developed by Stewart Butterfield in 2013. Today is one of the most popular instant messaging sites with millions of daily active users.

Slack can be used by all iOS, Android, Windows devices (pc, smartphone or tablet) as an application or as a web browser page.

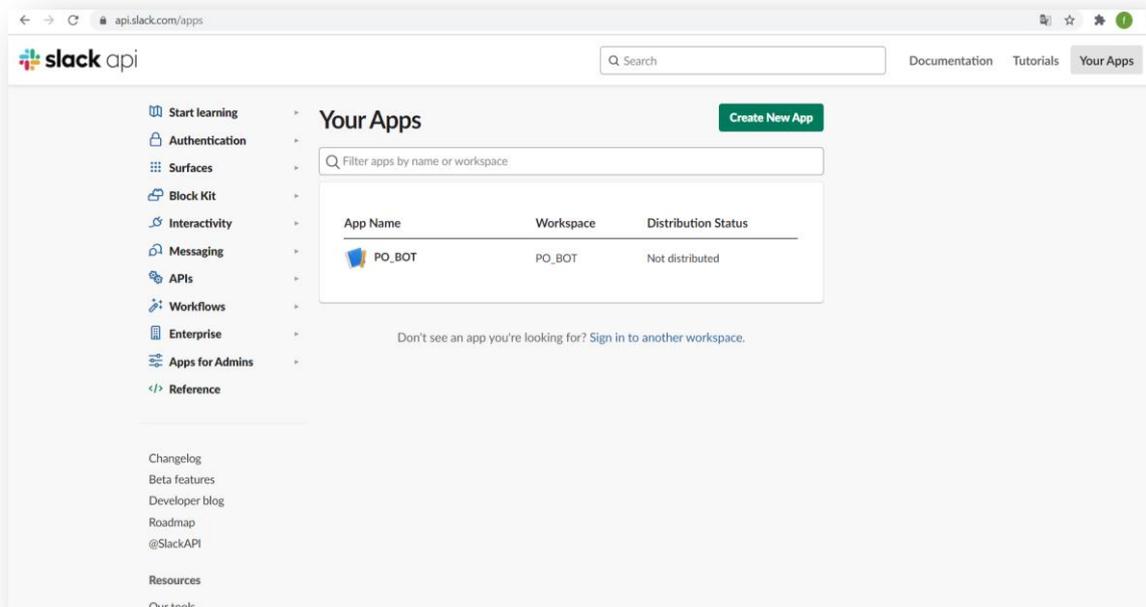
One of its functions is the ability to organize team communication through specific channels that can be accessible to the whole team or just to some members. It is also possible to communicate with the team through private individual chats or chats with two or more members. Another feature of Slack is the possibility to create some applications used in the same slack channel. The next paragraph will describe the Chabot creation on slack channel, used to answer users, in our case the students, questions.

3.1.1 Slack Bot App

Now let's see how to create the Chabot in Slack [7]:

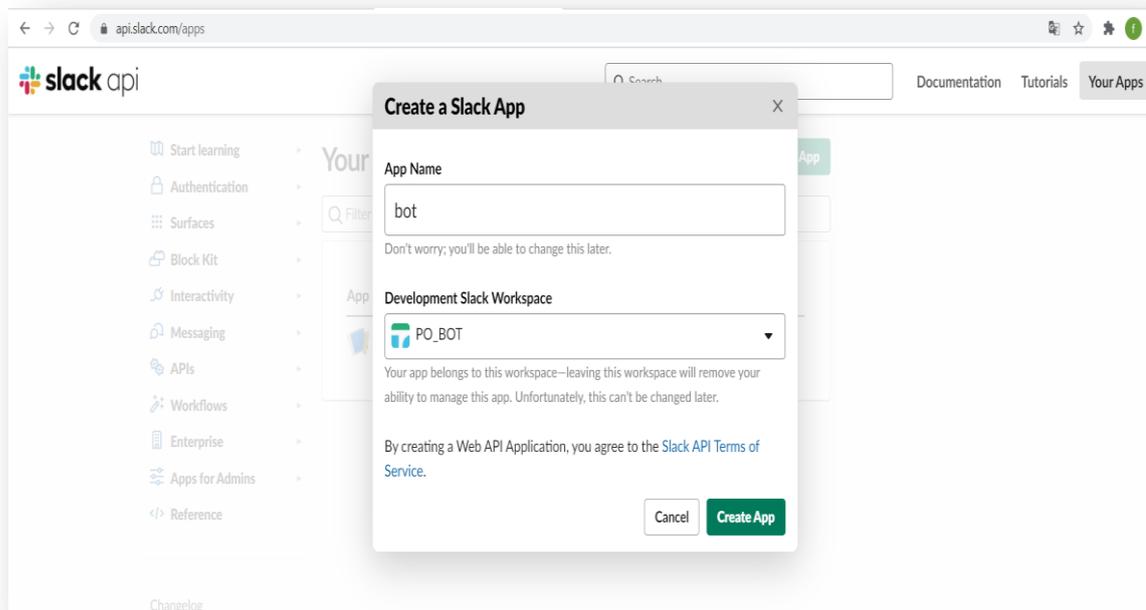
Step 1:

Go to <https://api.slack.com/apps>, click on “**Create New App**” as shown below:



Step 2:

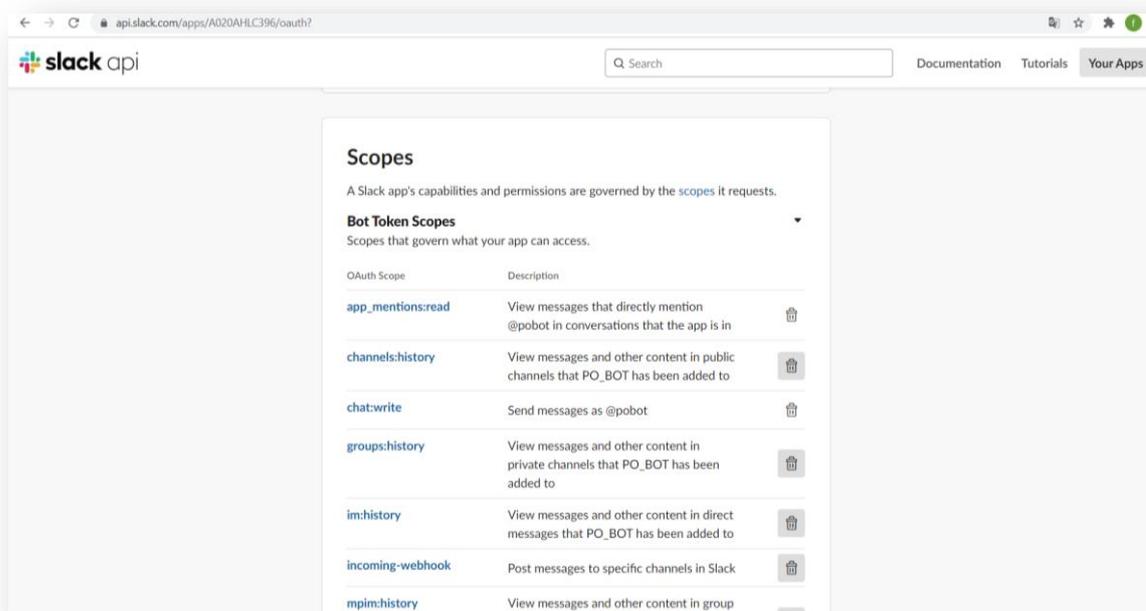
Choose an App name and the Slack Workspace where the bot will use:



Step 3:

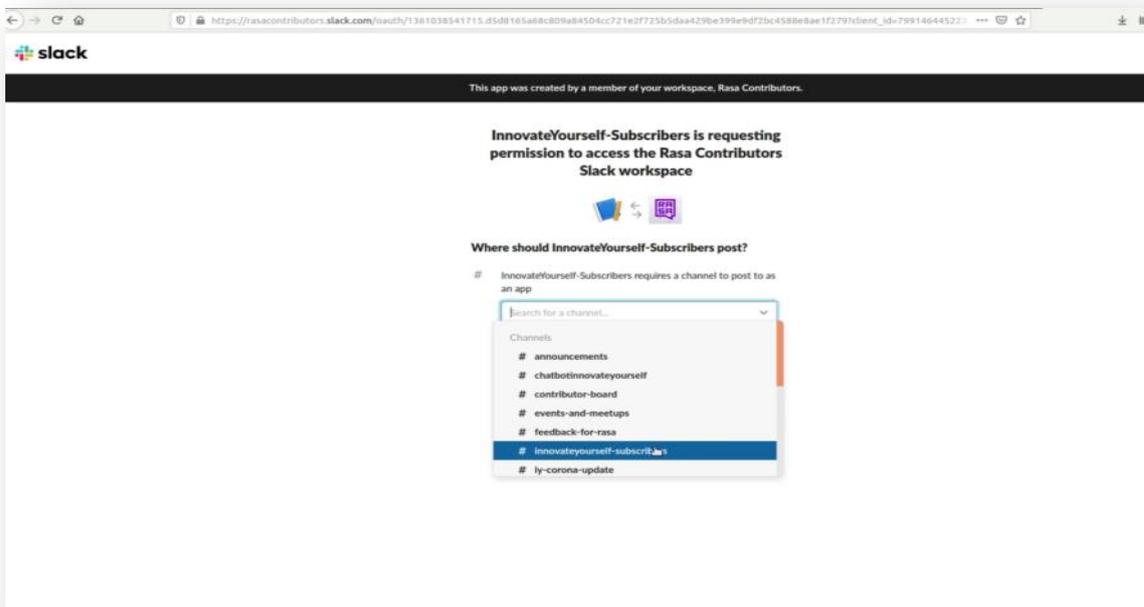
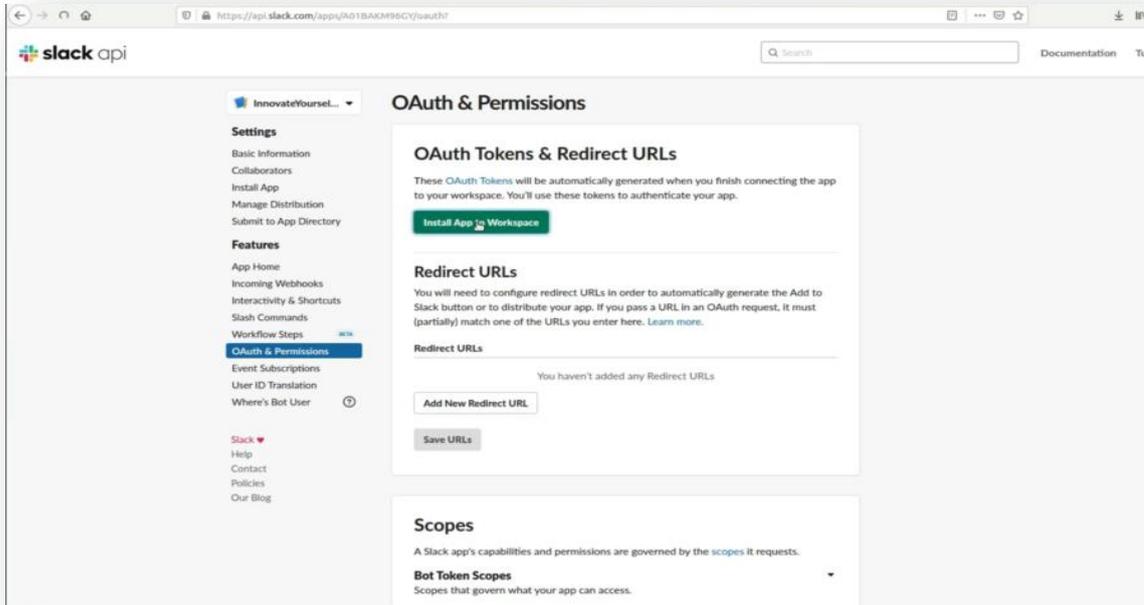
First of all in “**Oauth&Permission**” section, below “**Scopes**” choose the **Bot Token Scopes**, and **User Token Scopes** that bot will use to work. The scopes govern capabilities and permissions of bot and users and tokens support them.

Available scopes are on following page: <https://api.slack.com/scopes>



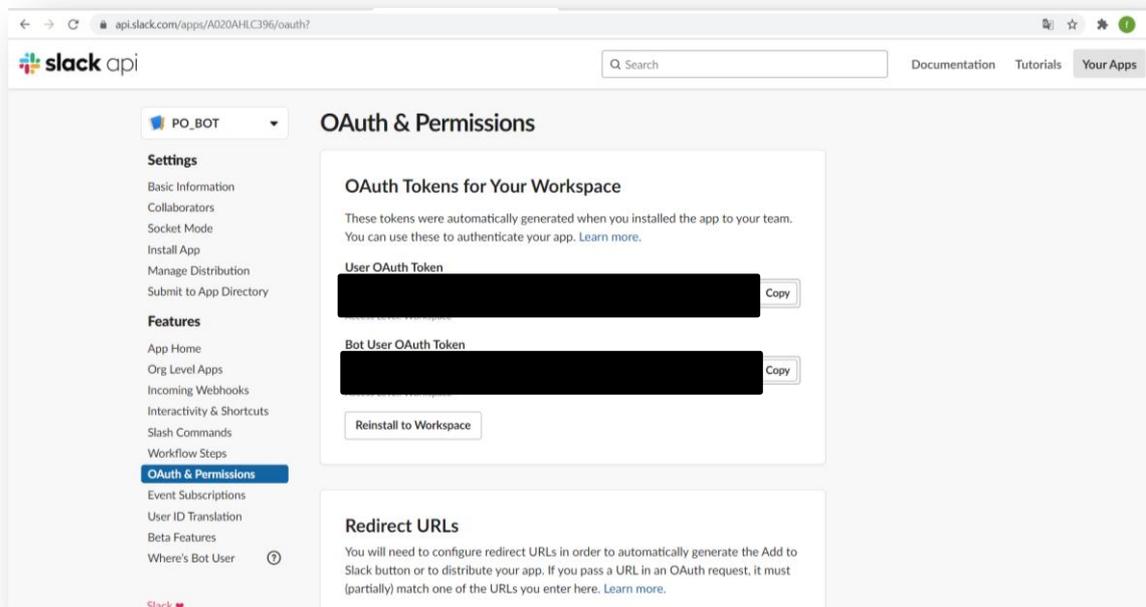
Step 4:

After the choose of scopes, click on the “**Install App on Workspace**” and choose the channel:



Step 5:

After the successful connection of the channel you will get the OAuth Token as shown here:



The **Bot Token** is important for the bot to manage the connection on the app and it must be secret and start with `xoxb-...`. These tokens will be saved, for instance as environment variables on the server, where the project is running.

3.2 RASA Open Source

Rasa Open Source [8] is a powerful framework that supplies the building to creating virtual assistants (bot). It is used to automate human-to-computer interaction anywhere from websites to social media platforms. It performs three principal functions:

- **Natural Language Understanding [9]:** it provides an open source natural language processing to turn messages from your users, in our case students, into intents and entities that assistants can understand in a better way
- **Dialogue Management [10]:** it manages the contextual conversation analysing step by step the dialogues
- **Integrations [11]:** it provides many built-in connectors to connect to common messaging and voice channels, such as Slack, Telegram, Discord

Rasa Open Source is licensed under the Apache 2.0 license, and the full code is available on this page: <https://github.com/RasaHQ/rasa>.

3.2.1 Natural Language Understanding

NLU [9] (Natural Language Understanding), a subset of NLP (Natural Language Processing), classifies the text intent based on the context and content of the message. NLU goes beyond converting text to its semantic parts and interprets the message of the user.

Rasa Open Source is based on lower-level machine learning libraries like TensorFlow and SpaCy and provides NLP software that is approachable and as customizable as you need. It gets up and running fast with easy to use default configurations, or swap out custom components and fine-tune hyper parameters to get the best possible performance for the dataset.

Advantage of Rasa in NLU are:

- Being **open source**, it is possible to see the source code, modify the components, and understand why your models behave the way they do
- Open Source NLP **offers the most flexible solution**. It is possible to plug in personal pre-trained models, build components, tune models with precision based on dataset
- It works with the principal pre-trained models like *BERT*, *SpaCy*, *ConveRT*
- **Support multiple intents** in a single message and can define hierarchical entities
- It is **possible to test**, in a simple way, the data using examples of conversations
- It supports some pre-built starter packs that can help developers in development of bot

3.2.2 Dialogue Management

To create a context-aware conversational assistant, the important thing is to define how the conversation history affects the next response.

The principal tasks to create it are [10]:

- **Slots**: are the assistant's memory. They store pieces of information that the bot needs to refer to later and can direct the conversation flow based on `slot_was_set` events. There are a lot of slot type and you can see all here: <https://rasa.com/docs/rasa/domain#slot-types>
- **Stories**: are the examples of conversation between user and bot. In the stories we write the steps of dialogues with the questions, phrases and correlates answers/actions that the assistant should say/do
- **Policy**: ML (Machine Learning) policy can help model to predict in a better way the response also in unseen conversation paths, but *"It is important to understand that using machine-learning policies does not mean letting go of control over your assistant"*

3.2.3 Integrations RASA on Slack

First of all, we should go to the `credentials.yml` file on the root of the Rasa project (see paragraph 3.3). And add the following rows [12]:

```
slack:
  slack_token: <BOT_SLACK_TOKEN> #token bot
  slack_channel: "A020AHL396" #channel ID bot
  slack_signing_secret: <SLACK_SIGNING_SECRET> # secret number
  slack_retry_reason_header: "x-slack_retry_reason"
  slack_retry_number_header: "x-slack-retry-num"
  errors_ignore_retry: None # Any error codes given by Slack included
                           in this list will be ignored.
```

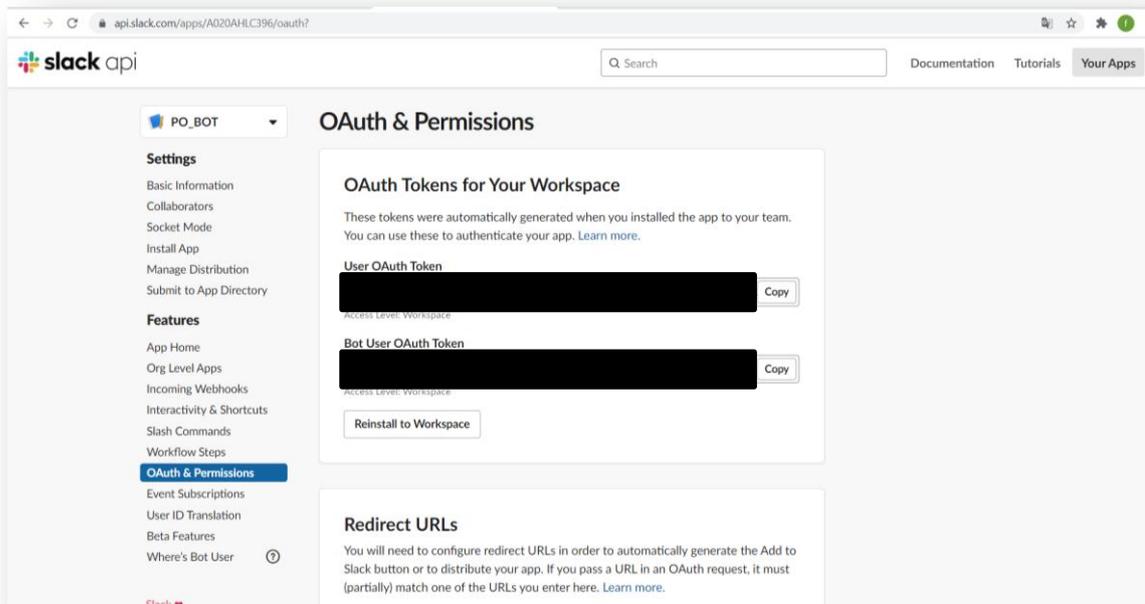
For `slack_token` and `slack_signing_secret` is recommended to create two environment variables to not expose the tokens clearly.

Step 1:

In the page of your app (example: <https://api.slack.com/apps/A020AHL396> where A020AHL396 is the channel ID) go in the “**OAuth&Permissions**” and scroll down to **Scopes**. You should add, at least, the following scopes:

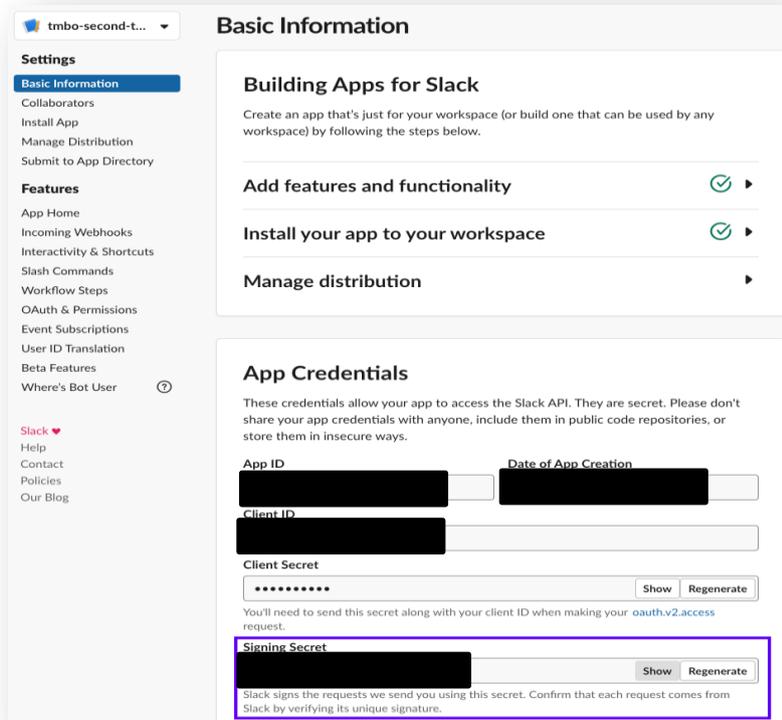
- `app_mentions:read`
- `channels:history`
- `chat:write`
- `groups:history`
- `im:history`
- `mpim:history`
- `reactions:write`

After “**reinstall the App**”, if you have changed some Scopes, in the same page copy the **Bot User Token** and replace the value in the `slack_token`.



Step 2:

In “**Basic Information**” section there is the **Signing Secret** that will be the `slack_signing_secret` value:



Step 3:

Now we must configure the bot to receive/send messages. First run the bot, for example using `rasa run` on the prompt.

Step 4:

On “**Event Subscriptions**” sections, active **Enable Events** and write the Request URL in this format:

`<public_url>/webhooks/slack/webhook`

NB: If you want to run locally the server using the localhost address, you must use ngrok [13] (or other tools) that retrieve a public URL for the server, because you won't be able to use the localhost address.

Step 5:

After that you need to **Subscribe to the bot events** on the same page. You will need to add the following events:

- message.channels
- message.groups
- message.im
- message.mpim

Event Subscriptions

Enable Events

Your app can subscribe to be notified of events in Slack (for example, when a user adds a reaction or creates a file) at a URL you choose. [Learn more.](#)

Request URL

[Change](#)

We'll send HTTP POST requests to this URL when events occur. As soon as you enter a URL, we'll send a request with a `challenge` parameter, and your endpoint must respond with the challenge value. [Learn more.](#)

Subscribe to bot events

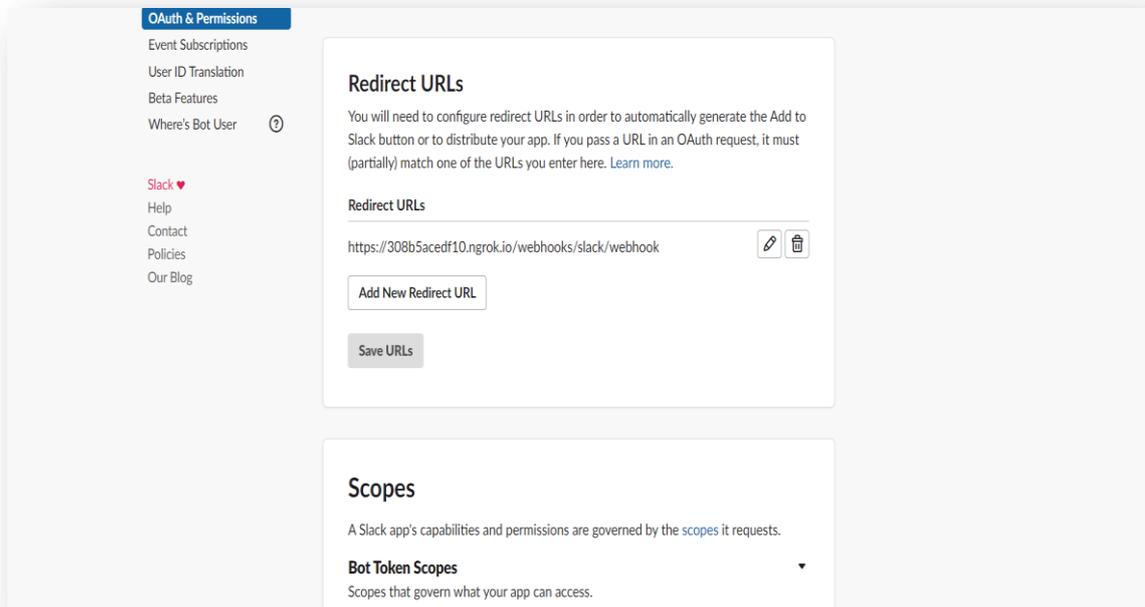
Apps can subscribe to receive events the bot user has access to (like new messages in a channel). If you add an event here, we'll add the necessary [OAuth scope](#) for you.

Event Name	Description	Required Scope	
message.channels	A message was posted to a channel	channels:history	
message.groups	A message was posted to a private channel	groups:history	
message.im	A message was posted in a direct message channel	im:history	
message.mpim	A message was posted in a multiparty direct message channel	mpim:history	

[Add Bot User Event](#)

Step 6:

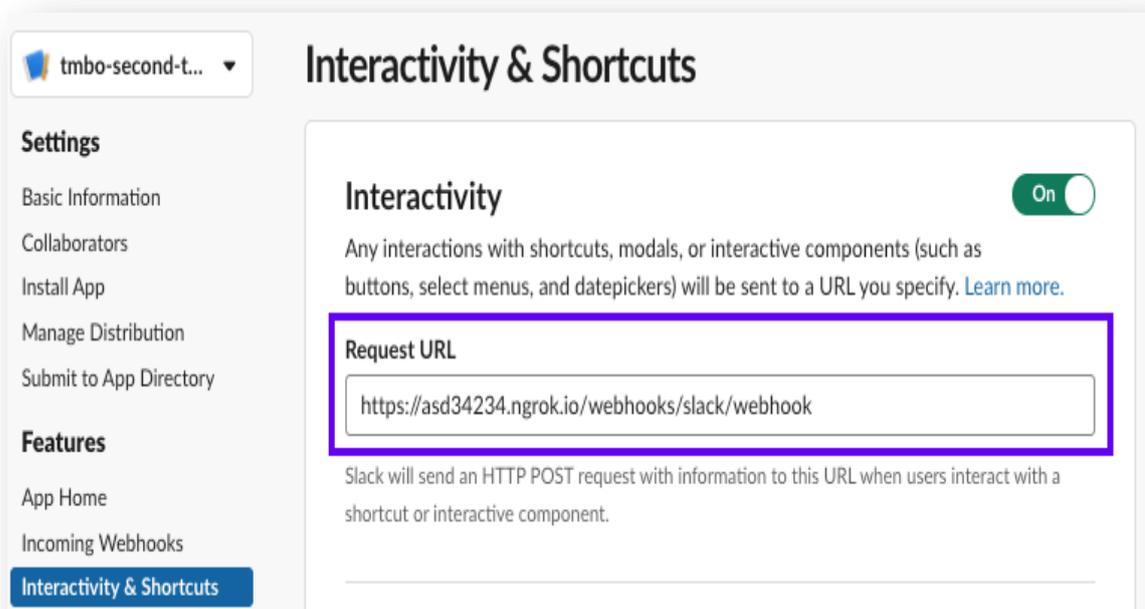
Go to the “**OAuth&Permission**” section, below **Redirect URLs**, click on “**Add New Redirect URL**” and insert the same URL on Step 4.



Step 7:

If you want that your assistant manage interactive Components, go on “**Interactivity&Shortcut**”, enable **Interactivity** and insert the same URL on the step 4.

Remember to Save all changes in any pages to have success of changes!



3.3 How to create, tested and run a RASA project

To create a Rasa project [14], first of all we need to install **python=3.6+** libraries, for instance using **Anaconda** [15] that is an open source, simple and very powerful distribution for python and R languages.

So, in Anaconda Console we can write this commands:

```
conda create -n venv python=3.7
conda activate venv
pip3 install rasa
```

After we can create initial Rasa project and install some useful components:

```
pip3 install rasa[spacy]
pip3 install rasa[transformers]
rasa init
```

`rasa init` command is used to create all files that Rasa needs and train a simple example using simple data.

All project must have some `*.yaml/*.py` file that programmer have to modify to implement his personal virtual assistant (* can be changed with any words). One organization of project could be:

- `actions/actions_*.py`: file or files code in python to create custom actions
- `configs/config_*.yaml` [16]: file or files that contain the configuration of NLU/NLP and Core models. Using the pipeline we can personalize models used to train the data, containing *i.e.* Tokenizers, Featurizers, Classifiers, ResponseSelector and also manage the ML rules policy. It supports all pre-trained models such as SpaCy, TensorFlow, BERT, ConveRT
- `data/nlu.yaml` [17]: examples of user utterances categorized by intent (as a topic) to be trained. Also it possible defined entities, structured pieces of information inside utterances
- `data/stories.yaml` [18]: contains examples of conversations between user and AI assistant, converted in a specific format: intents (and entities is necessary) are user inputs; actions are responses of bot or something customized, depending of topic predicted
- `data/rules.yaml` [19]: describe small pieces of conversations that should always follow the same path
- `endpoints.yaml`: detail for connecting to channels like Slack
- `models/*.tar.gz`: models trained
- `domain.yaml` [20]: contain the domain of assistant: the list of intents, entities, slots, responses, forms and actions that bot should know about

If wanted training the data with particular configurations, run this command:

```
rasa train --config [config file]
```

To run the server:

```
rasa run
```

To tested the stories contains in test/test_*.yaml:

```
rasa test --config [config file] --cross-validation --runs [num] --
folds [num] --out [output directory]
```

To tested the intent recognition:

```
rasa test nlu --nlu {test_file} --out {dir_out} --model {model name}
```

3.4 How write stories (rules)

As written in the previous paragraph, the stories [18] (and also the rules equally) contain examples of conversation between AI Assistant and User. They are a type of training data used to train dialogue management models, and they can be used to generalize to unseen conversation paths.

A story should be written in stories.yaml file and they have a simple format. An example of story, take in Rasa website:

```
stories:
- story: name of story
steps:
- intent: greet # user message with no entities
- action: utter_ask_howcanhelp # action or response that the bot
  should execute
- intent: inform # user message with entity
- location: "rome"
- price: "cheap"
- action: utter_on_it
- action: utter_ask_cuisine
- intent: inform
entities:
- cuisine: "spanish"
- action: utter_ask_num_people
```

Where:

- **Intent:** represent the NLU domain, the topic, argument of question that user ask
- **Action:** the response of Assistant or custom action (see next paragraph). Every action have a key (or name) that make it unique, as intent
- **Entities:** particular structured pieces of information used inside user messages

Stories can also describe **events** [21], returned often by custom actions, and are tracked automatically by Rasa (i.e. user messages). The problem is that the assistant's model doesn't

know which events will return. Because of this in stories we should activate/deactivate form or setting slots explicitly:

- **Slots Events:** after custom action we should add the information about slot to set/reset:

```
- story: set/reset slot
steps:
# ... other story steps
- action: my_custom_action
- slot_was_set:
- my_slot1: null # set slot to None (reset)
- my_slot2: "hi" # set slot to initial text
```

- **Form Events [22]:** a form, often, is used to save information of users, through slots. After defining the form (and slots or entities) in `domain.yml`:

```
forms:
restaurant_form:
required_slots:
cuisine:
- type: from_text
```

We can define stories that activate/deactivate it. There are three kinds of events used to manage **forms**:

- A form action server (i.e. `- action: restaurant_form`) that is used to starting a form and resuming the form action if is already active
- A form activation event (i.e. `- active_loop: restaurant_form`) to activate the form
- A form deactivation event (i.e. `- active_loop: null`) to deactivate the form

An complete example of story, take on Rasa website:

```
- story: User interrupts the form and doesn't want to continue
steps:
- intent: request_restaurant
- action: restaurant_form # start form
- active_loop: restaurant_form # activate form
- intent: stop
- action: utter_ask_continue
- intent: stop
- action: action_deactivate_loop
- active_loop: null # deactivate form
```

Now talking about **checkpoints** and **OR statements** [18].

- **Checkpoints:** are used to modularize and simplify training data. They can be useful but we do not overuse them, because they could make stories hard to understand and slow down training. An example take on Rasa website:

```
story: beginning of flow
steps:
- intent: greet
- action: action_ask_user_question
- checkpoint: check_asked_question

- story: handle user affirm
steps:
- checkpoint: check_asked_question
- intent: affirm
- action: action_handle_affirmation
```

- **OR statements:** are used to write shorter stories or to treat multiple intents the same way. Also an overuse of them will slow down training. An example take on Rasa website:

```
- story:
steps:
# ... previous steps
- action: utter_ask_confirm
- or:
- intent: affirm
- intent: thankyou
- action: action_handle_affirmation
```

3.5 Principal configuration models

The model [20], is the “brain” of an assistant. Thanks to it, the bot predicts the topic of a question, and answers, using components that work sequentially creating a pipeline that processes user input.

The principal components [23] are:

- **Language Models:** load pre-trained models that are needed in case you want to use pre-trained word vectors in the pipeline. The principals are:
 - **SpacyNLP:** initialized spaCy structures. You have to specify the language model to use and if it will be case sensitive or not.
 - **HFTransformersNLP:** use HuggingFace’s Transformers based pre-trained language model, as BERT. You have to specify the `model_name` and the `model_weights`.
- **Tokenizers:** split text into tokens. The principals are:
 - **SpacyTokenizer**
 - **ConveRTTokenizer:** if we use the ConveRT model
 - **LanguageModelTokenizer**

- **Featurizers:** they can be *sparse*, that return feature vectors with a lot of missing values, *i.e.* zeros, and usually take up a lot of memory, or *dense*, that store only value not missing and the position of features in the vector. The featurizers return a sequence feature (matrix contains a feature vector for every token in sequence) or a sentence feature (matrix that contains the feature vector of all utterances).
The principals are:
 - **SpacyFeaturizer**
 - **ConveRTFeaturizer**
 - **LanguageModelFeaturizer**
 - **CountVectorsFeaturizer:** it is used for intent classification and response selections. create bag-of-words represent the user messages, intent and responses. All tokens consisting only of digits will be assigned to the same feature.
- **Intent classifiers:** assign one of intents defined in the domain file to user messages (predict the topic of question in our case):
 - **DIETClassifier** (Dual Intent Entity Trasformer Classifier) is used for intent classification and entity extraction. The architecture is based on a transformer which is shared for both tasks. A CRF (Conditional Random Filed) layer tagging the top 10 (usually) entity labels prediction using user input. It uses dot-product loss to maximize the similarity with the target label and minimize similarity with negative samples. It possible to change some hyperparameters as *epochs*, *model_confidence* (*softmax or linear_norm*), *transformer_size*
- **Entity Extractors:** extract the entities, such a person's name or a location, from user input. The most used are:
 - **SpacyEntityExtractor**
 - **DietClassifier**
 - **EntitySinonymMapper**
- **Selectors:** predict an assistant answer form a set of candidate responses included in the domain file. The principal used is ResponseSelector, and we can modify some hyperparameters as *DIETClassifier* to customize the model.

Also we can create custom components and use them in the model to perform specific tasks if pre-trained models are not sufficient to have high accuracy.

In the configuration models, there are also the policies that assistants use to decide which action to take at each step in a conversation. At every turn, each policy defined in configuration will predict a next action with a certain value of confidence, and the policy that predicts with the highest confidence decides the bot's next action, managed if present also the policy priority.

The most used Policies [24] are:

- **TED Policy:** Transformer Embedded Dialogue Policy is a multi-task architecture for next action prediction and entity recognition. It possible configure the hyperparameters and manage the “*nlu_fallback_actions*” (if there is no actions that have confidence major of a certain threshold the bot not predict the action but answer wit “I don't understand” or something else defined in default utterance in domain file)

- **Memoization Policy:** remember the stories from training data. It checks if the current conversation matches the stories in your stories file
- **Rule Policy:** handles conversation parts that follow a fixed behaviour. It manages the rules contained in the rules file. Also here we can manage the “*nlu_fallback_actions*”

As for the model, we can customize our policy and use it in the configuration file.

3.6 How create custom actions

When a user posts a message, the model used to predict the intent and response, will choose the best action [25][26] , with the highest probabilities of success, that the assistant should perform next. It could be a simple text response (the `utter_*` created in `responses` section in `domain.yml`) or a more complex action that can be customized creating classes written in Python language saved in `actions` directory.

A custom action can run any code we want, including API calls (for instance we can use Slack API, for instance, to send messages in a channel or in a private chat with another user), manage databases, add particular events (*i.e.* calendar), do maths calculations.

All custom actions should be defined in the `actions` section of `domain.yml` file.

When dialogue engine predict that assistant should be execute a custom action, it will call the action server, with this information:

```
{
  "next_action": "string",
  "sender_id": "string",
  "tracker": {
    "conversation_id": "default",
    "slots": {},
    "latest_message": {},
    "latest_event_time": 1537645578.314389,
    "followup_action": "string",
    "paused": false,
    "events": [],
    "latest_input_channel": "rest",
    "active_loop": {},
    "latest action": {},
  },
  "domain": {
    "config": {},
    "session_config": {},
    "intents": [],
    "entities": [],
    "slots": {},
    "responses": {},
    "actions": [],
    "forms": {},
    "e2e_actions": []
  },
  "version": "version"
}
```

The tracker [27] represents a Rasa conversation tracker. Thanks it, lets you access the assistant's memory and get information about past events, current state through it attributes:

- `sender_id`: unique user ID talking to the assistant
- `slots`: the slots list that can be filled as defined in the domains
- `latest_message`: a dictionary containing the latest message attributes containing: intent, entities and text
- `events`: all previous events list
- `active_loop`: the currently active loop name. It used to manage the form
- `latest_action_name`: the last action name that the bot executed

The domain [20] contains information about the “universe” in which assistants operate. In contains:

- `intents`: intents list (the arguments or topics) used in NLU data
- `entities`: the entities that can be extracted in NLU pipeline
- `slots`: the assistants memory. It is used as a key-value variables with type (bool, text, float, ...);
- `responses`: the responses that assistant can use
- `forms`: is a special type of action that it is used to collect information (i.e. modify a slot) from user
- `actions`: the possible actions list, including the customized

And server respond with a events and responses list:

```
{
  "events": [{}],
  "responses": [{}]
```

To run the server of Rasa actions [28], first of all we can add in `endpoints.yml` file the following lines:

```
action_endpoint:
  url: "{address}/webhook"
```

where `{address}` is the address of the action server.

After in a console, we should write this commands:

```
rasa run actions
```

3.6.1 Custom actions Class

Every custom action classes [29] have the following format:

```

from rasa_sdk import Action, Tracker
class MyCustomAction(Action):

def name(self) -> Text:
return "action_name"
async def run(
self, dispatcher, tracker:Tracker, domain:Dict[Text, Any],
) -> List[Dict[Text, Any]]:
...
return []

```

It compose of two principal methods (but we can add others static methods if want):

- `name()`: return the custom action name
- `run(dispatcher, tracker, domain)`: it is called when action is predicted and it contains the “custom” action that the assistant should do. The three parameters are:
 - **tracker** [27]: It contains attributes and methods useful for manage the state of current user
 - **domain** [20]: contains information about the domain
 - **dispatcher** [30]: it used to generate the responses to send back to the user. An instance of `CollectingDispatcher` contains only the method `utter_message` that is used to send the responses

3.7 Rasa X: an “evolution” of RASA

Another component used in this thesis is Rasa X [31], an improvement of the framework Rasa Open Source.

Rasa X is a very powerful tool for Conversation-Driven Development (CDD). It processes, saves automatically all conversations about Bot and User and uses them to improve AI assistant and model used.

With Rasa X it is possible to analyse the conversation and model to try to learn how users write and how he should be respond, improving over time.



Figure 1: functionalities of RASA X, from [31]

The principal Rasa X advantages are:

- it is a layer on top of Rasa Open Source and try to **improve the model** used using a fine-tuning approaches to build a very custom and **better assistant**
- In particular [32]:
- **Review conversations:** sort and filters, tag important messages, get insight into user behaviour
 - **Annotate data:** label real user messages, create new flows from real conversations, fix incorrect predictions
 - **Share & Test:** shareable links to test the assistant, analyse performance, create robust test cases
- it is **free**, closed source tool available to all developers
 - it can be **deployed anywhere:** train your data securely and proprietary
 - It also offers very **user-friendly web pages** (it appears when the Rasa X server is in running)

3.7.1 How install Rasa X

Rasa X can be installed in four mode [33]:

- Local mode
- Server Quick-Install Mode
- Helm Chart Mode
- Docker Compose Mode

In this thesis it will describe the first mode.

Local Mode Installation [34]

If Rasa Open Source is just installed, the installation of Rasa X is very simple.

- 1) First of all in a console we should write this command:

```
pip3 install rasa-x --extra-index-url https://pypi.rasa.com/simple
```

- 2) After in `credentials.yml` file we should add this line:

```
rasa:  
  url: "{address}/api"
```

where `{address}` is the address of the rasa x server.

- 3) To run the Rasa X server, for instance we can write:

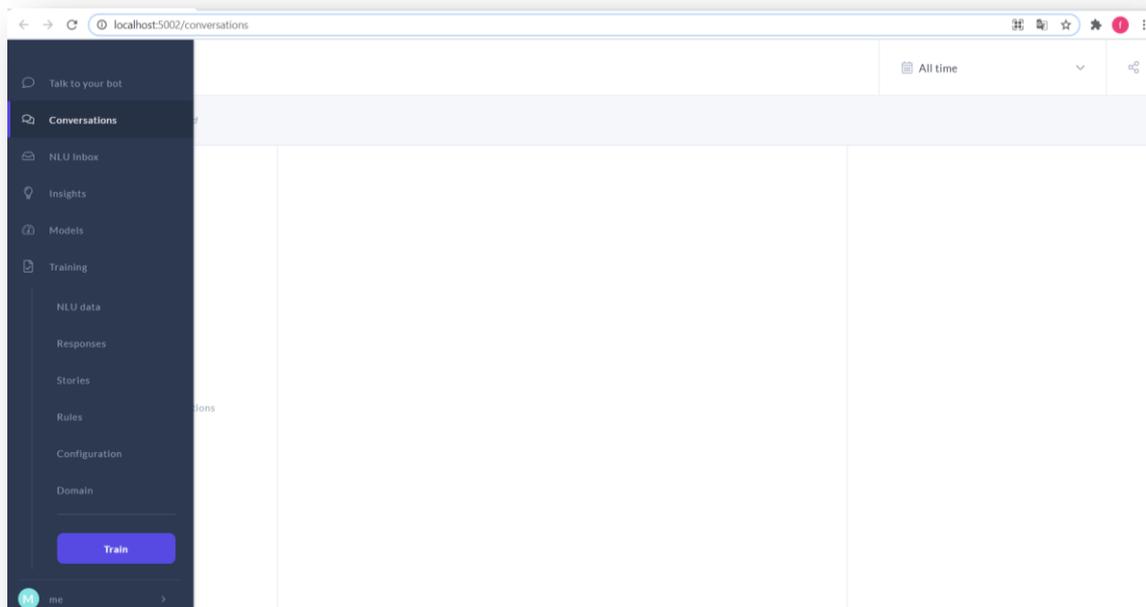
```
rasa x --connector slack --config configs/config_BERT.yml
```

To connect Rasa X server with Slack, for instance, using `config_BERT` (model that uses BERT pipelines) file.

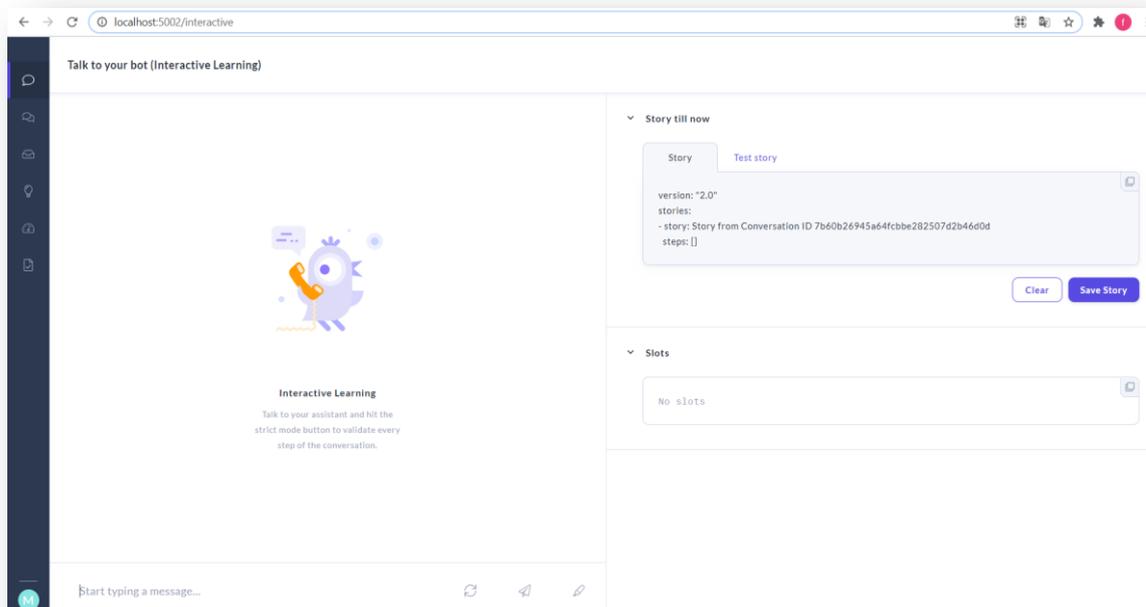
3.7.2 Rasa X Server

When Rasa X [35] is running, in the browser appears a web page that is useful to the developer to manage the AI assistant and do the operation mentioned previously.

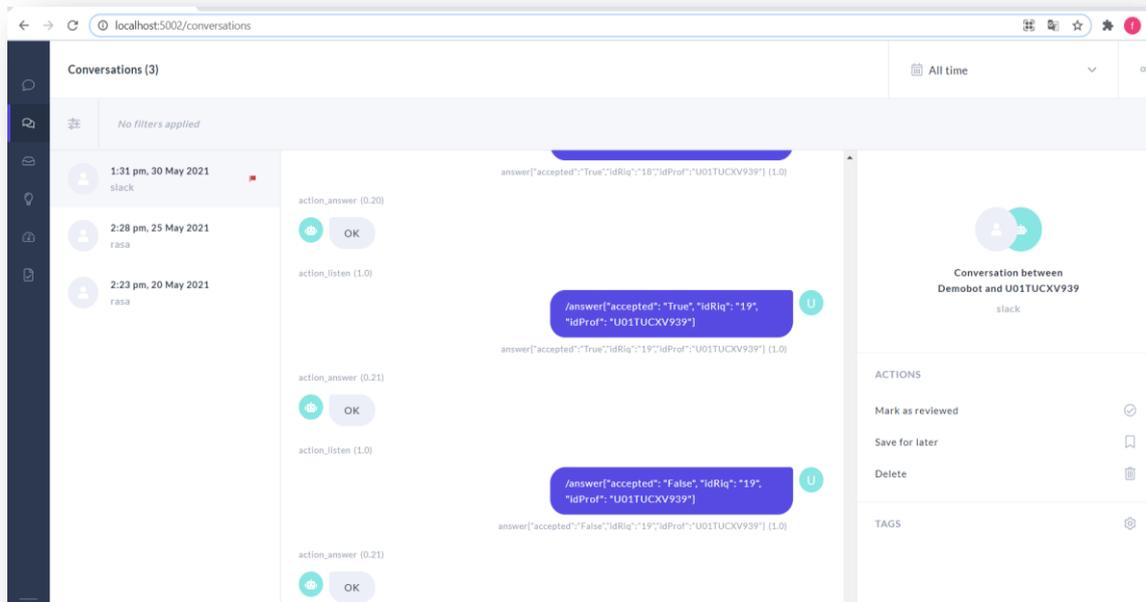
The home page include a menu that contains:



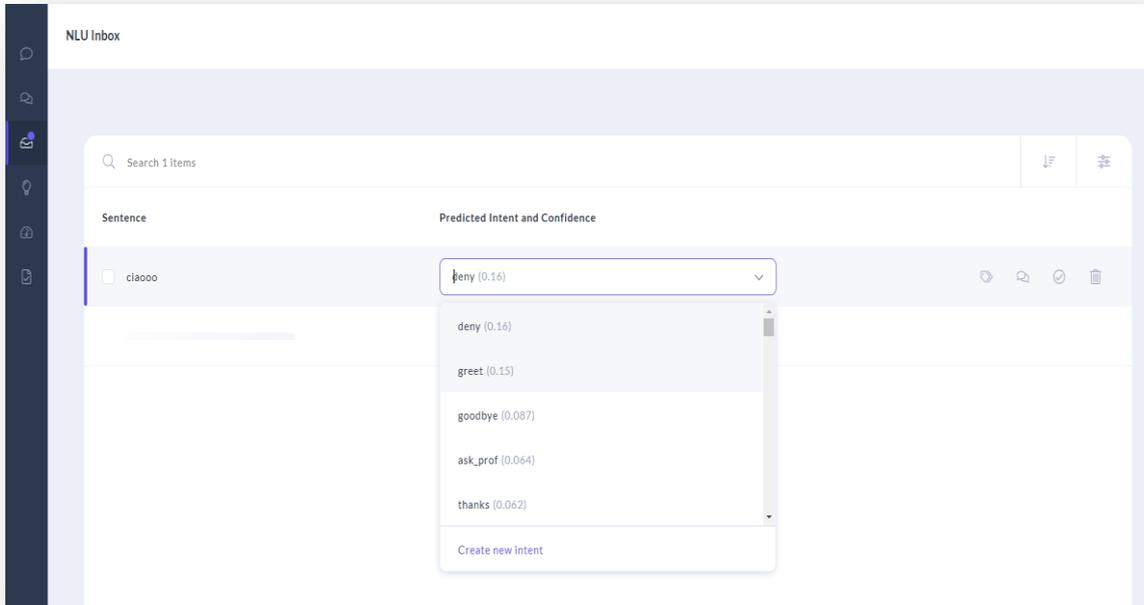
- **Talk to your bot:** in this part you can talk with your assistant, test the conversation and analyse the story (or create) and slots.



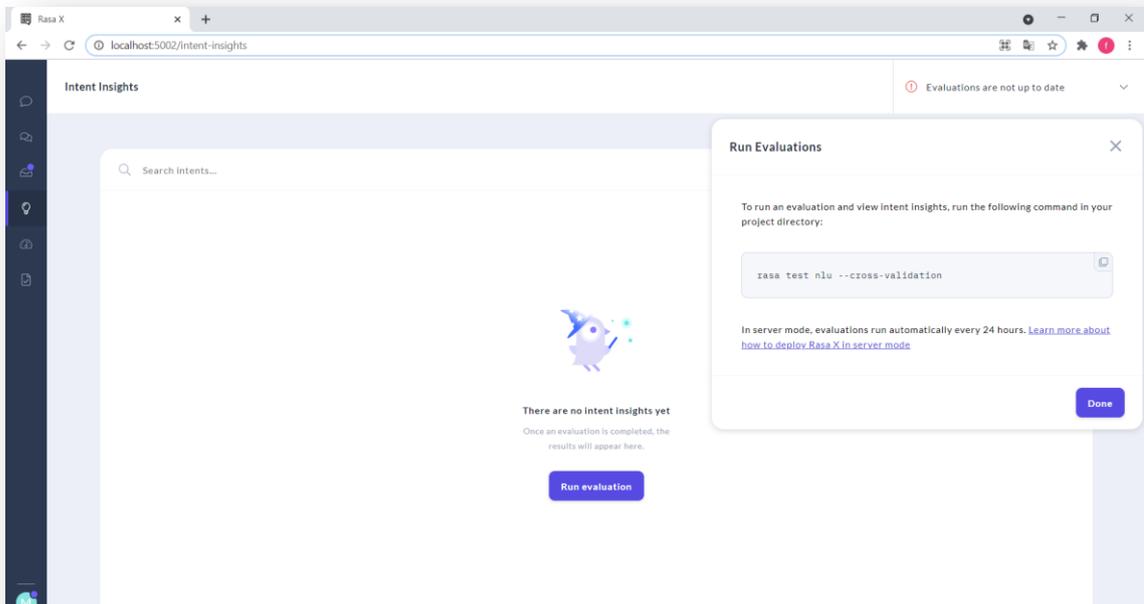
- **Conversations:** It contains the list of all conversations, with some metrics evaluation, between Assistant and user and we can manage it (for instance mark if an intent is predicted correctly or not, save or delete it, ...).



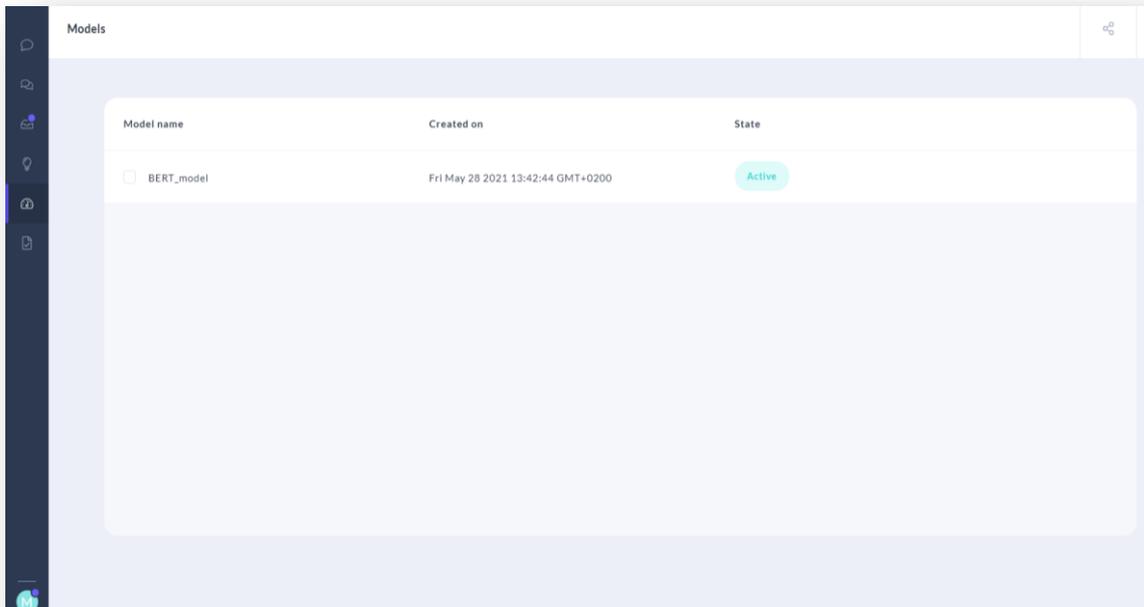
- **NLU inbox:** it contains all users utterances that NLU dataset do not contain, for instance. It predicts the intent and if prediction is incorrect could be changed/created or message can be deleted.



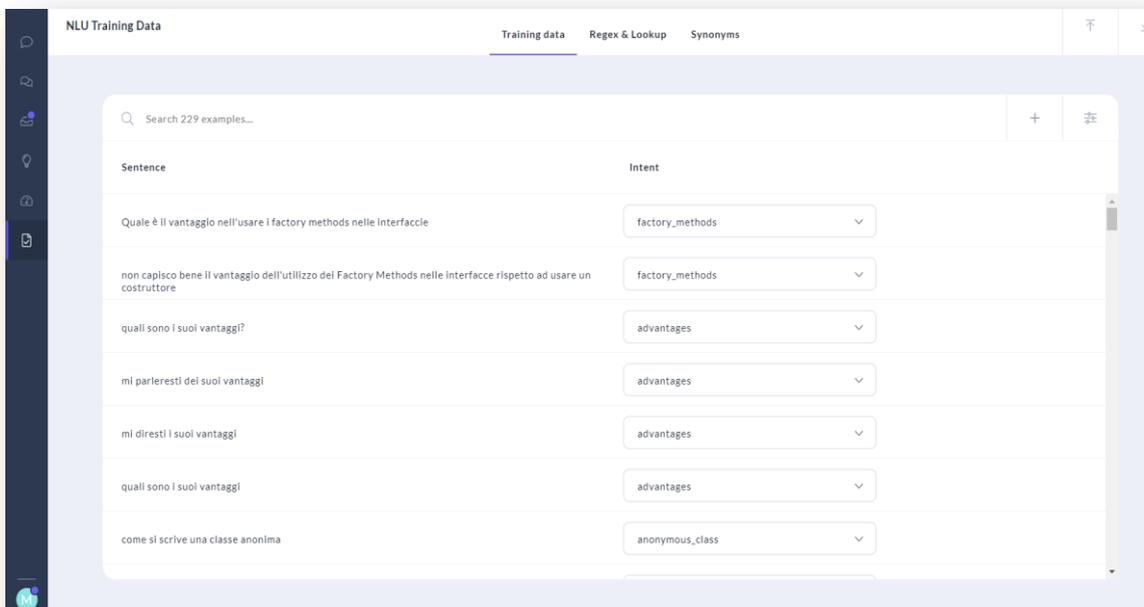
- **Insight:** it is useful to test the dataset of test stories contained in the `test` directory.

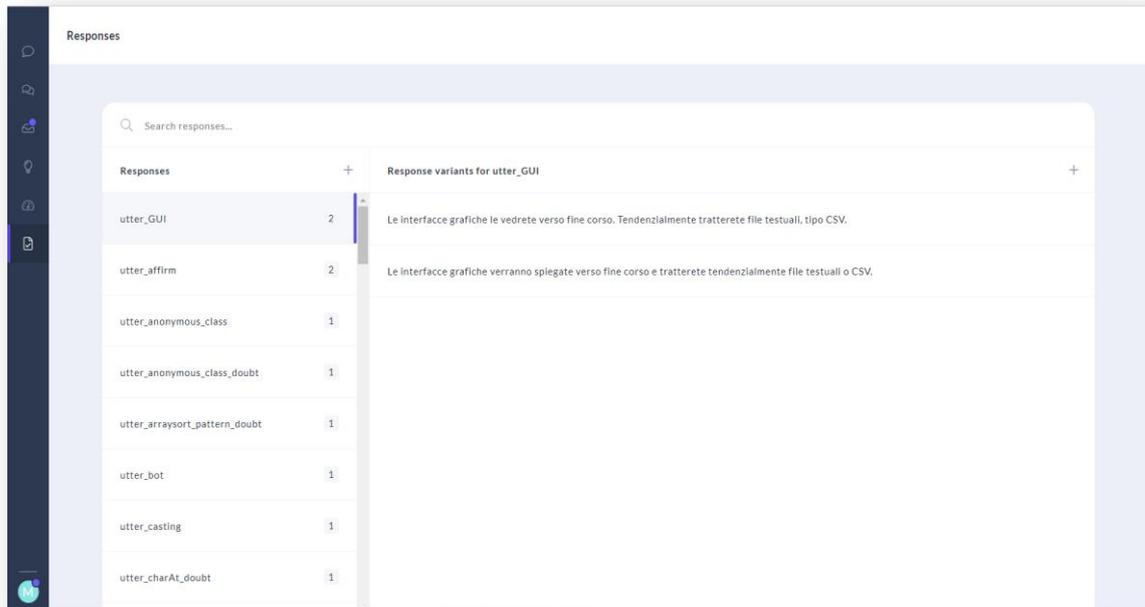


- **Models:** It manages the list of models available and we can activate one of them.



- **Training:** We can manage the NLU data, stories, rules and responses, configuration and domain files.





3.8 A brief introduction of Python

Python [36][37] is an interpreted high-level general-purpose programming language. It could be used to develop distributed applications, scripting and system testing. It emphasizes code readability with its notable use of significant indentations. Its Object Oriented approach aims to help programmers write clear, logical code. The first that began working on Python was Guido van Rossum in the 1980s.

This language is dynamically-typed and garbage-collected, supports multiple programming paradigms, and a comprehensive standard library that provides tools suited to many tasks as Data Analytics, Databases, Machine Learning, Scientific computing, System administration as Threads manage.

3.8.1 How write a class in python

To define a class in python [37] you can use a specific instruction called `class`. It is possible to inherit multiple times, definition of attributes using initialization, and operators.

The reflective parameter is called `self` by default, and it represents the pointer of the class object. Usually is the first parameter of every method defined in the class.

The `__init__` method represents the constructor and usually is used to define the attributes of class and call super class if derived from a superclass.

Exist also other special methods associated to operators and built-in functions, *i.e.* `__add__`, `__str__`

An example of class and his use is:

```
class Vehicle(object):
    def __init__(self, name, number_of_ports):
        self.name = name
        self.number_of_ports = number_of_ports

    def name_of_vehicle(self):
        full = f'{self.name}'
        return full

vehicle = Vehilce('Fiat', 5)
print(persona.name_of_vehicle())
```

3.9 A brief introduction of SQLite

SQLite [38][39] is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, serverless, transactional, zero-configuration SQL database engine. SQLite is Open Source and one of the most used in the world. To manage the DB it is possible to use a graphical tools class DB Browser for SQLite to manage DB (creation, modification and run the principal SQL operation like SELECT).

So the principal features are [40]:

- The transaction are atomic, consistent, isolated and durable (**ACID**)
- **Zero-configurations**: no administration needed
- **Full-featured SQL**
- A complete DB is stored in a **single cross-platform** disk file
- Support **terabyte-sized** DB and gigabyte-sized string and blobs
- **Small code**
- **Fast**
- **Cross-platform**
- **Open Source**

3.9.1 SQLite in Python

The sqlite3 library [41] is used to manage SQLite DB in python, and is written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification.

After the import the first thing to do is create a connection with the DB. Then we must create a **cursor** that is used to move in the DB. The cursor has some methods to call the SQL of DDL, DML, *etc.* save the modification and close the connections with DB. The `execute` method returns a table that can be used to process and read value if you want directly as a list.

A complete example, take on <https://docs.python.org> website:

```
import sqlite3
con = sqlite.connect()
cur = con.cursor()

cur.execute("create table lang (name, first_appeared)")
cur.execute("insert into lang values (?, ?)", ("C", 1972))

lang_list = [
    ("Fortran", 1957),
    ("Python", 1991),
    ("Go", 2009),
]

cur.executemany("insert into lang values (?, ?)", lang_list)
cur.execute("select * from lang where first_appeared=:year",
            {"year": 1972})
print(cur.fetchall())
con.close()
```

4. Edu-Bot implementation

4.1 The problem and the goals

The initial problem, was to create a Chatbot that would help the students of some courses held at the Politecnico of Turin in learning their curricular contents and also in purely organizational management, for example knowing where are the slides of the courses, consequently also helping teachers in the management of requests from students who, given the increasing number of students enrolled at the university every year, it is increasingly difficult to manage them all in a reasonable time.

A resume of the main goals, presented on Introduction section, that the AI Assistant had to solve were:

- **answer the questions** proposed by the students by summarizing the content requested by the student through a complete answer
- **offer a “Human Handoff” service**, which allows the student to contact the teacher if the bot has not answered the student's request in an exhaustive manner
- **allowing everything through a communication channel** used by the university and students and it was decided to use Slack, which offers, among other things, the possibility of creating APPs that can also be a Chatbot

After implementing the Chatbot, another goal of research has been **to evaluate everything** using different prediction models used in Machine Learning in the NLP: BERT, Spacy and ConveRT.

4.2 The dataset

During the work, we focused on one course in particular: "**Programmazione ad Oggetti (09CBIxx)**" [42]. So, in the end it was decided to create a virtual bot that would help the students of the PO course.

The topics of the course are vast and run along the theoretical and practical line, through the study of the Java language, of object-oriented programming. Initially it was decided to create a virtual assistant that would be able to answer all the topics of the course both theoretical and practical, then in the course of work we opted to a selection of a particular topic, the *inheritance* (in addition to some questions of organizational type) in theory and in practice (Java programming), already vast as a topic and given that the technology used makes it very easy, in the future, to add new topics and therefore to expand the dataset now implemented.

The dataset is a mixture of questions and answers (with the related "topics") taken from the Slack channels of the PO course of the year 2020, as Train Set, and 2021, as Test Set,

included some created specifically to try to expand the **argument of inheritance** as much as possible.

Then I found 53 different subtopics (with 300 examples of different questions), and manage in standard RASA format (NLU file, stories file, rules file, domain file):

<i>Inheritance in general</i>	<i>Comparable</i>
<i>Override</i>	<i>Comparator</i>
<i>Polymorphism</i>	<i>Iterator-Iterable</i>
<i>Dynamic binding</i>	<i>Observer-Observable</i>
<i>Casting</i>	<i>Lambda functions</i>
<i>Downcasting</i>	<i>Upcasting</i>
<i>Class/attribute visibility</i>	<i>Methods references</i>
<i>Constructors in inheritance</i>	<i>Instance of keyword</i>
<i>The super construct</i>	<i>“this“ construct</i>
<i>The Object class</i>	<i>“final” construct</i>
<i>Abstract classes</i>	<i>Pattern</i>
<i>Interfaces</i>	<i>Static methods</i>
<i>Defaults methods</i>	<i>Defaults methods</i>

And in addition to the questions of last year's students it included also, *the organizational questions (e.g. where are the slides, ...), and the regards.*

All data has been meticulously analysed and manually anonymised for privacy issue, another much debated topic in AI Assistant and modelled as the technology of RASA wanted.

4.3 The Technologies used

As seen in the previous chapters, the **RASA framework** was used to manage the dataset, the prediction models, the data testing and therefore the evaluation of the different models using the classical Machine Learning metrics.

In addition to offering a graphical interface (**RASA X**) that simplifies the management work even more. It also allows you to customize the actions to be done by the assistant and therefore allows the creation of the “Human Handoff” required by creating classes written in Python language.

Slack was chosen as a communication channel, as required, due to its popularity of use, simplicity on creation and use.

SQLite was also used to manage the requests of students who asked to be able to speak with the teachers "in first person" without the aid of the bot.

4.4 The AI Assistant's architecture

The choice of the architecture of the dataset training model is fundamental: we could define it as the “brain”, as well as the thinking mind of the virtual assistant. In this thesis I decided to compare three pre-trained models, widely used in Machine Learning NLP problems: *SpaCy*, *Google-BERT* and *ConveRT*.

These three configurations can be used by the RASA framework, which as you may have understood by now, is the "core technology" used in my implementation. Furthermore, Rasa tries to simplify the work as much as possible, allowing the definition of a configuration file which, through a series of pipelines that are executed consecutively and the definition of the Policies, is able to predict the intents, entities and answers of bot, all in autonomous way, without having to write code, also speeding up the implementation work.

4.4.1 SPACY Model

SpaCy [43] is the first pre-trained model that I wanted analysed. It is an open-source software library used in Natural Language Processing (NLP) problems, written in Python languages and Cython. The library is published under MIT license and its principal developers are Matthew Honnibal and Ines Montani.

Spacy is focused on providing software for production usage and support deep learning workflows that allow connecting statistical models by TensorFlow, PyTorch, or others popular Machine Learning libraries. Also its features **Convolutional Neural Network (CNN)** models for part-of-speech tagging, dependency parsing, text categorization and NER (Named Entity Recognition). Its tasks are available in different languages, as **English**, **Italian**, Spanish, and Chinese.

So, Its main features are:

- **Non-destructive tokenization**
- “**Alpha tokenization**” support 65 languages
- **Built-in support** for trainable pipeline components such as *NER*, *Part-of-speech tagging*
- Support **Statistical models** for 17 languages, including English, Italian, ...
- **Support for custom models** in *PyTorch*, *TensorFlow* and other frameworks
- State-of-the-art **speed** and **accuracy**
- **Easy model** packaging, deployment and workflow management
- **Built-in visualizer** for *syntax* and *named entities*

In my RASA configuration I used the “**it-core-news-md**” model of Spacy, to support *Italian* language. I configured the **hyperparameters** for every part of the pipeline, for instance the *epochs* (300) of DietClassifiers and other components, the *fallback threshold* in the Policies, the model confidence of , for instance, ResponseSelector.

```

language: it

pipeline:
  - name: SpacyNLP
    model: "it_core_news_md"
    case_sensitive: False
  - name: SpacyTokenizer
    intent_tokenization_flag: False
    intent_split_symbol: "*"
    token_pattern: None
  - name: SpacyFeaturizer
    pooling: "mean"
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: char_wb
    min_ngram: 1
    max_ngram: 10
  - name: DIETClassifier
    epochs: 300
    constrain_similarities: True
    model_confidence: 'linear_norm'
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 300
    constrain_similarities: True
    model_confidence: 'linear_norm'

policies:
  - name: RulePolicy
    core_fallback_threshold: 0.1
    core_fallback_action_name: "action_default_fallback"
    enable_fallback_prediction: True
  - name: MemoizationPolicy
  - name: TEDPolicy
    core_fallback_threshold: 0.1
    core_fallback_action_name: "action_default_fallback"
    enable_fallback_prediction: True
    max_history: 5
    epochs: 300
    constrain_similarities: True
    model_confidence: 'linear_norm'

```

4.4.2 BERT Model

BERT (Bidirectional Encoder Representation from Transformers) [44], developed by **Google**, is a recent open source method of pre-training language representations, a general purpose “*language understanding*” model on a large text corpus (BooksCorpus and Wikipedia) and also is used in NLP tasks (fine-tuning). It is **fast** and **relies on massive computation** and **generates multiple, contextual, bidirectional** word representations. In addition proposed a new training objective: the **MLM** (Masked Language Model), that randomly masks some of the tokens from the input and predicts the original vocabulary id of the masked word based only on its context.

The basic BERT building block is the Transformer, opposed RNN (Recurrent Neural Network): central is the notation of *self-attention*, contextual *co-occurrence* statistics.

Transformer is **simpler** and more **parallelizable, faster** than RNN, also because it use only **matrix multiplications** and *simple few layers feed forward neural network* with no recurrence and no weight sharing.

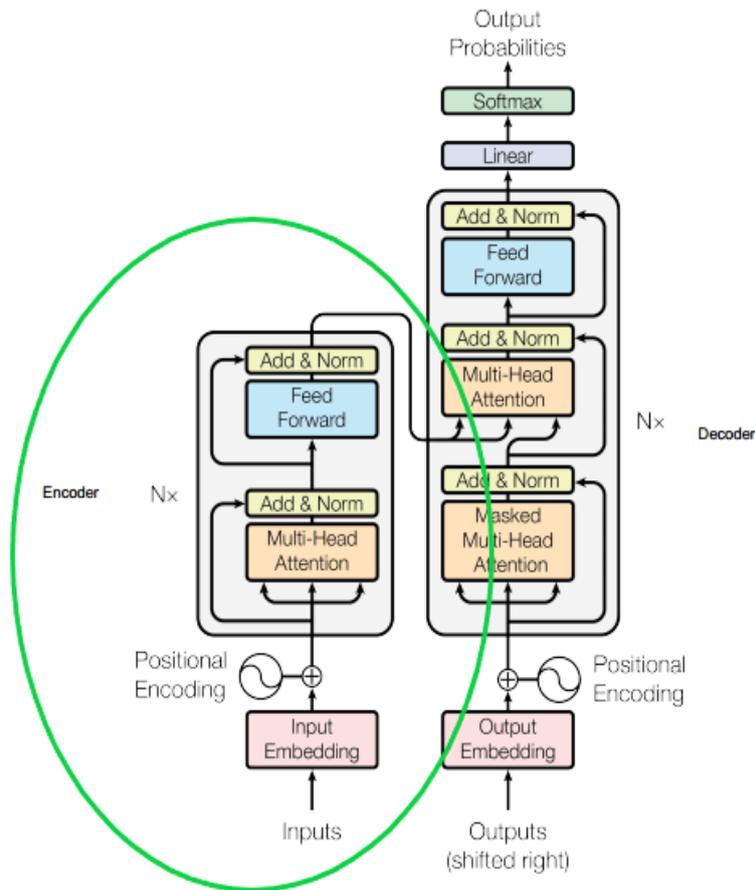


Figure 1: The Transformer - model architecture.

Figure from [44]

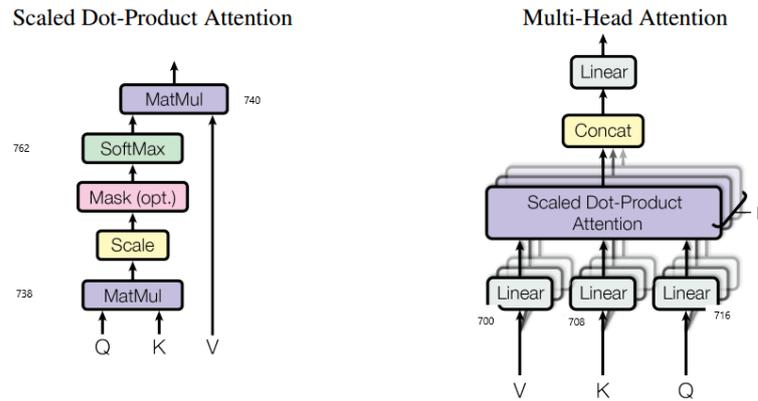


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Figure from [44]

Important is the pre-training **tokenization** of text that is divided in three sequentially operations:

- **Token Embeddings:** start to tokenize the text, clear and normalize it, change every tokens in value and transform it in vector of 768 (by default) embeddings
- **Segment Embeddings:** identify every singular phrases
- **Position Embeddings:** add positional to input embedding

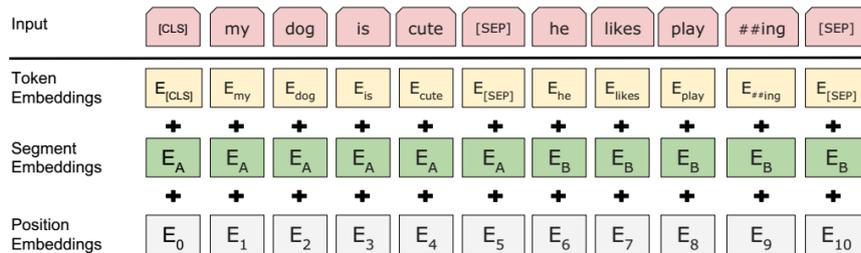


Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Figure from [44]

BERT supports a lot of different models for a lot of languages, such as English or **Italian** and there are particular variants of BERT, like *ALBERT*, *RoBERTa* that can help to train better the tasks to study.

In my RASA configuration I used “**dbmdz/bert-base-italian-xxl-uncased**” [45] model of BERT, to support *Italian* language with large vocabulary. Also I configured the **hyperparameters** for every components of pipeline, the same of SPACY configurations and ConveRT.

```

language: it

pipeline:
  - name: LanguageModelTokenizer
  - name: LanguageModelFeaturizer
    model_weights: "dbmdz/bert-base-italian-xxl-uncased"
    model_name: "bert"
    cache_dir: "./cache_bert"
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: char_wb
    min_ngram: 1
    max_ngram: 10
  - name: DIETClassifier
    epochs: 300
    constrain_similarities: True
    model_confidence: 'linear_norm'
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 300
    model_confidence: 'linear_norm'
    constrain_similarities: True

policies:
  - name: RulePolicy
    core_fallback_threshold: 0.1
    core_fallback_action_name: "action_default_fallback"
    enable_fallback_prediction: True
  - name: MemoizationPolicy
  - name: TEDPolicy
    max_history: 5
    core_fallback_threshold: 0.1
    core_fallback_action_name: "action_default_fallback"
    enable_fallback_prediction: True
    epochs: 300
    constrain_similarities: True
    model_confidence: 'linear_norm'

```

4.4.3 ConveRT Model

ConveRT (Conversational Representation from Transformers) [46] is a pre-training framework for conversational tasks and satisfying these requirements: effective, affordable and quick to train.

It is very **scalable** and **portable**: is only 59 MB in size and is significantly **smaller** than other state-of-the-art dual encoders (444MB), as BERT. Also is **more compact** than other sentence encoders, and consequently **faster** than its. This reduction in size and training are achieved through combining 8-bit embedding quantization and quantization-aware training, subword-level parameterization, and pruned self-attention. In addition, it **provides a multi-context** variant that is also very compact (73 MB).

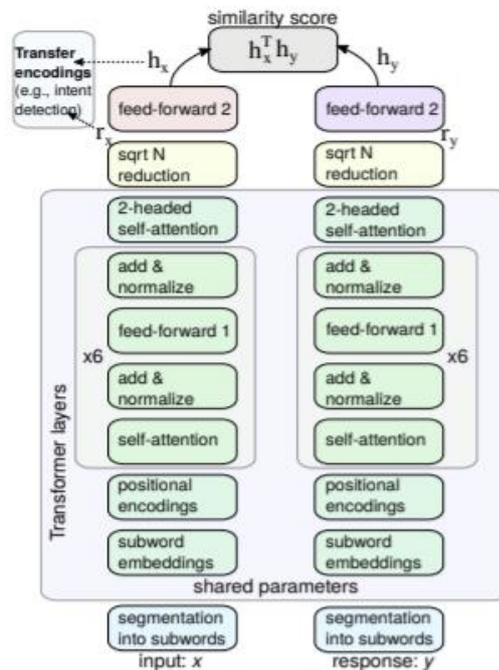


Figure 1: *Single-context ConveRT* dual-encoder model

Figure from [46]

As SpaCy and BERT models I configured the **hyperparameters** for every component of the pipeline. In this case ConveRT recognises the Italian languages, with the help of the RASA framework.

```
language: it

pipeline:
  - name: ConveRTTokenizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: CountVectorsFeaturizer
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 10
  - name: DIETClassifier
    epochs: 300
    constrain_similarities: True
    model_confidence: 'linear_norm'
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 300
    constrain_similarities: True
    model_confidence: 'linear_norm'

policies:
  - name: RulePolicy
    core_fallback_threshold: 0.1
    core_fallback_action_name: "action_default_fallback"
    enable_fallback_prediction: True
```

```

- name: MemoizationPolicy
- name: TEDPolicy
  core_fallback_threshold: 0.1
  core_fallback_action_name: "action_default_fallback"
  enable_fallback_prediction: True
  max_history: 5
  epochs: 300
  constrain_similarities: True
  model_confidence: 'linear_norm'

```

4.5 The final prototype

The final prototype of the project is able to meet all the goals initially set, that are:

- **allowing everything through a communication channel:** Slack is used as a communication channel. The bot, created as Slack App, is an additional “user” called **PO_BOT**, inserted in the channel with name **#po_bot** where students and professors can communicate with each other. When it is mentioned it answers at student request.
- **answer the students questions, resuming the content,** for the topics of *PO inheritance* and some organisation request:



Figure 1: Example of organisational question

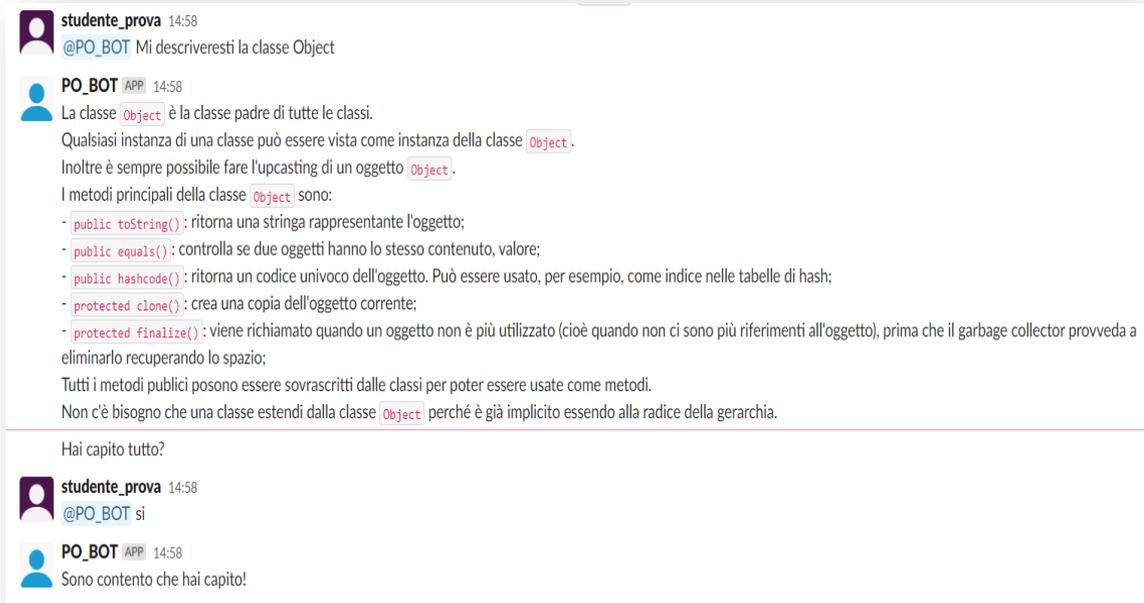


Figure 2: Example of inheritance question (about Object class)

- **offer a “Human Handoff” service:** when a student wants to contact a professor, the student asks a virtual assistant to send a message with a question to the professor. The professor receives in a private channel the request and the link of question and he/she can accept it or send it to another teacher, using the buttons included in the messages. An example seen in the figures below.

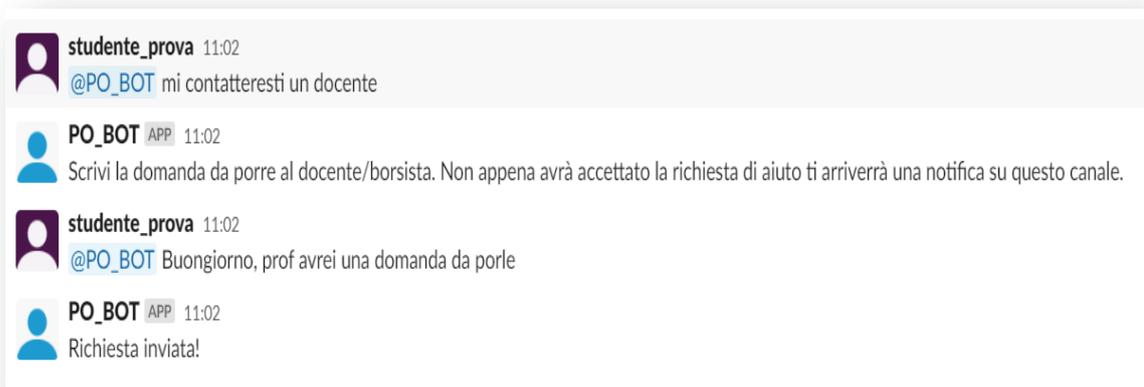


Figure 3: Example of Human Handoff request



Figure 4: Message arrived at professor in private chat

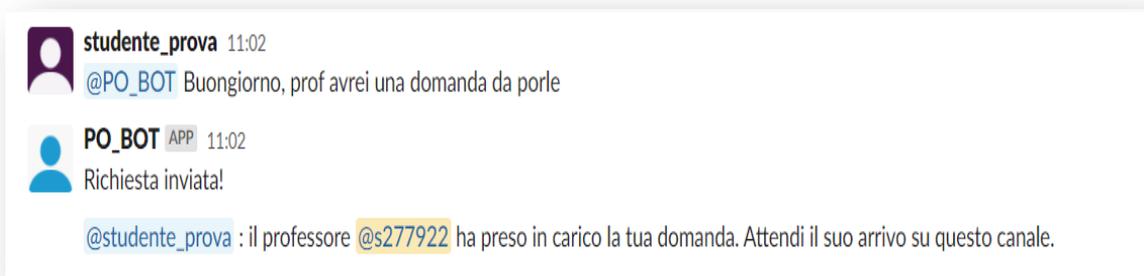


Figure 5: Notification of acceptance in main channel

5. Evaluation

5.1 The Evaluation metrics

An important aspect of the goals of this thesis is the evaluation of model [47] is used to train the dataset.

In this case, again using the RASA framework, I focused on evaluating the **Intent recognition** to see if the model is able to correctly recognize the topic to which it refers to given a question. There is also an analogue for entities, but in this project research they have not been used and therefore not tested accordingly. After testing the Intent recognition, I wanted to analyse the confidence of models, using a series of **test stories** that simulate different completed dialogues and allow you to provide entire conversations and **test the actions**.

For this evaluation I used some classical metrics, used in Machine Learning, also in multi-label classification:

- **Precision [48]**: called *positive predictive value*, is the fraction of relevant instances (true positives) among the retrieved instances (the true and false positives). It can be defined also as the average probability of relevant retrieval
- **Recall [48]**: called *sensitivity*, is the fraction of relevant instances that were retrieved (the true positives divide true positives with false negatives). It can be defined also as the average probability of complete retrieval averaged over multiple retrieval queries
- **F1-score [49]**: it is calculated from the precision and recall. The formula used is:

$$\text{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

In multi-class classification the final score is obtained by *micro-averaging* (biased by class frequency) or *macro-averaging* (taking all classes as equally important)

- **Confusion matrix [50]**: is a specific table that allows visualization of the performance of a model. Each row represent the instances in an actual class while each column represent the instances in a predicted class, or vice versa
- **Accuracy [51]**: is defined as the fraction of number of correct predictions among the total number of predictions

These metrics have a range of value between *zero* and *one*, and values closer to one, are the better.

5.2 The NLU Evaluation

To test only the NLU model, so only the Intent recognition, Rasa offers the possibility to split the NLU file that contains the intents with examples, in train and test set. To do this we can run this command [47]:

```
rasa data split nlu
```

And the original file will be divided into the train dataset (80% of dataset of NLU: 240 distinct examples) and test set (20% of dataset of NLU: 60 distinct examples) and saved in a directory called `train_test_split`.

To test run this command:

```
rasa test nlu --nlu train_test_split/test data.yml
```

The results of evaluation are saved in different files:

- `intent_report.json`: contains a report contains recall, f1-score, precision for every topics
- `intent_confusion_matrix.png`: contains the confusion matrix
- `intent_histogram.png`: contains confidence histogram for intent classification model and allow to visualize the confidence for all predictions, with correct (blue bars) and incorrect (red bars) predictions
- `errors.json`: contains the incorrect predicted intents

5.2.1 SPACY Model Evaluation

The result of Spacy model for *Intent Evaluation* are reported below:

Value of metrics for every intent

Intent*	#train	#test	precision	recall	F1-score
<i>final</i>	3	1	1.00	1.00	1.00
<i>goodday</i>	4	1	1.00	1.00	1.00
<i>goodbye</i>	4	1	1.00	1.00	1.00
<i>deny</i>	4	1	0.50	1.00	0.67
<i>thanks</i>	4	1	1.00	1.00	1.00
<i>bot</i>	4	1	1.00	1.00	1.00
<i>GUI</i>	4	1	1.00	1.00	1.00
<i>upcasting</i>	4	1	1.00	1.00	1.00
<i>visibility</i>	4	1	1.00	1.00	1.00
<i>hineritance_constructor</i>	4	1	0.50	1.00	0.67
<i>super</i>	4	1	0.00	0.00	0.00
<i>class_object</i>	4	1	1.00	1.00	1.00

comparator	4	1	0.00	0.00	0.00
<i>observer_observable</i>	4	1	1.00	1.00	1.00
greet	4	1	0.00	0.00	0.00
<i>method_reference</i>	4	1	1.00	1.00	1.00
<i>objects_methods_doubt</i>	4	1	1.00	1.00	1.00
<i>static_final_doubt</i>	4	1	1.00	1.00	1.00
<i>visibility_variables_main</i>	4	1	1.00	1.00	1.00
<i>anonym_class_doubt</i>	4	1	1.00	1.00	1.00
<i>equals_doubt</i>	4	1	1.00	1.00	1.00
<i>arraysort_pattern_doubt</i>	4	1	1.00	1.00	1.00
<i>charAt_doubt</i>	4	1	1.00	1.00	1.00
<i>class_general_doubt</i>	4	1	1.00	1.00	1.00
<i>referent_methods_doubt</i>	4	1	1.00	1.00	1.00
<i>comparing_doubt</i>	4	1	1.00	1.00	1.00
<i>difference_dynaminc_bind_upcast</i>	4	1	1.00	1.00	1.00
<i>hineritance_tree</i>	4	1	1.00	1.00	1.00
<i>terminology</i>	4	1	1.00	1.00	1.00
<i>instanceof</i>	4	1	1.00	1.00	1.00
<i>this</i>	4	1	1.00	1.00	1.00
<i>pattern</i>	4	1	1.00	1.00	1.00
<i>functional_interface</i>	4	1	1.00	1.00	1.00
default_methods	4	1	0.00	0.00	0.00
<i>advantages</i>	4	1	1.00	1.00	1.00
<i>factory_methods</i>	4	1	1.00	1.00	1.00
affirm	5	1	0.00	0.00	0.00
<i>eclipse</i>	4	2	1.00	1.00	1.00
<i>override</i>	5	1	1.00	1.00	1.00
dynamic_binding	5	1	0.34	1.00	0.50
<i>casting</i>	5	1	1.00	1.00	1.00
<i>downcasting</i>	5	1	1.00	1.00	1.00
<i>class_abstract</i>	4	2	1.00	1.00	1.00
comparable	5	1	0.50	1.00	0.67
<i>iterator_iterable</i>	5	1	1.00	1.00	1.00
<i>lambda</i>	5	1	1.00	1.00	1.00
<i>static_methods</i>	5	1	1.00	1.00	1.00
<i>slides</i>	5	2	1.00	1.00	1.00
<i>polymorphism</i>	6	1	1.00	1.00	1.00
<i>anonymous_class</i>	5	2	1.00	1.00	1.00

<i>hineritance</i>	8	1	1.00	1.00	1.00
<i>interface</i>	7	2	1.00	1.00	1.00
<i>ask_prof</i>	13	3	1.00	1.00	1.00
	240	60	0.88	0.92	0.89

Accuracy 0.92

**In bold type the intents partially or totally not recognised correctly.*

Examples of intents correct predicted or not recognized

Example	Intent predicted	Intent correct	Correct?
hey	<i>dynamic_binding</i>	<i>greet</i>	NO
descrivimi il comparator	<i>comparable</i>	<i>comparator</i>	NO
cosa serve l'operator instanceof	<i>instanceof</i>	<i>instanceof</i>	YES

Intent Confusion Matrix and Intent Prediction Confidence Distribution (IPCD)

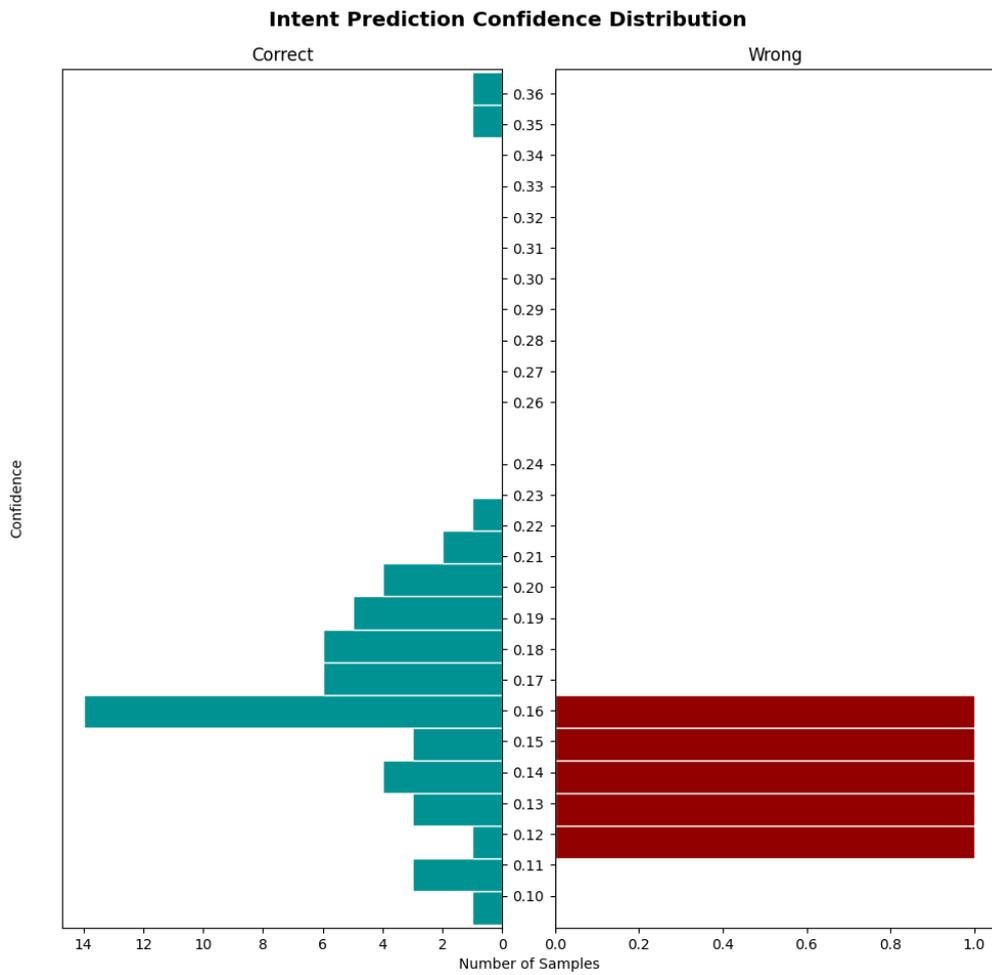


Figure 1: IPCD of Spacy model

5.2.2 BERT Model Evaluation

The result of BERT model for *Intent Evaluation* are reported below:

Value of metrics for every intent

Intent *	#train	#test	precision	recall	F1-score
<i>final</i>	3	1	1.00	1.00	1.00
<i>goodday</i>	4	1	1.00	1.00	1.00
<i>goodbye</i>	4	1	0.00	0.00	0.00
<i>deny</i>	4	1	0.33	1.00	0.50
<i>thanks</i>	4	1	1.00	1.00	1.00
<i>bot</i>	4	1	1.00	1.00	1.00
<i>GUI</i>	4	1	1.00	1.00	1.00
<i>upcasting</i>	4	1	1.00	1.00	1.00
<i>visibility</i>	4	1	1.00	1.00	1.00
<i>hineritance_constructor</i>	4	1	0.50	1.00	0.67
<i>super</i>	4	1	0.00	0.00	0.00
<i>class_object</i>	4	1	1.00	1.00	1.00
<i>comparator</i>	4	1	1.00	1.00	1.00
<i>observer_observable</i>	4	1	1.00	1.00	1.00
<i>greet</i>	4	1	0.00	0.00	0.00
<i>method_reference</i>	4	1	1.00	1.00	1.00
<i>objects_methods_doubt</i>	4	1	1.00	1.00	1.00
<i>static_final_doubt</i>	4	1	1.00	1.00	1.00
<i>visibility_variables_main</i>	4	1	1.00	1.00	1.00
<i>anonym_class_doubt</i>	4	1	1.00	1.00	1.00
<i>equals_doubt</i>	4	1	1.00	1.00	1.00
<i>arraysort_pattern_doubt</i>	4	1	1.00	1.00	1.00
<i>charAt_doubt</i>	4	1	1.00	1.00	1.00
<i>class_general_doubt</i>	4	1	1.00	1.00	1.00
<i>referent_methods_doubt</i>	4	1	1.00	1.00	1.00
<i>comparing_doubt</i>	4	1	1.00	1.00	1.00
<i>difference_dynaminc_bind_upcast</i>	4	1	1.00	1.00	1.00
<i>hineritance_tree</i>	4	1	1.00	1.00	1.00
<i>terminology</i>	4	1	1.00	1.00	1.00
<i>instanceof</i>	4	1	1.00	1.00	1.00
<i>this</i>	4	1	1.00	1.00	100
<i>pattern</i>	4	1	1.00	1.00	1.00

<i>functional_interface</i>	4	1	1.00	1.00	1.00
<i>default_methods</i>	4	1	1.00	1.00	1.00
<i>advantages</i>	4	1	0.50	1.00	0.67
<i>factory_methods</i>	4	1	1.00	1.00	1.00
<i>affirm</i>	5	1	0.00	0.00	0.00
<i>eclipse</i>	4	2	1.00	1.00	1.00
<i>override</i>	5	1	0.00	0.00	0.00
<i>dynamic_binding</i>	5	1	1.00	1.00	1.00
<i>casting</i>	5	1	1.00	1.00	1.00
<i>downcasting</i>	5	1	1.00	1.00	1.00
<i>class_abstract</i>	4	2	1.00	1.00	1.00
<i>comparable</i>	5	1	1.00	1.00	1.00
<i>iterator_iterable</i>	5	1	1.00	1.00	1.00
<i>lambda</i>	5	1	1.00	1.00	1.00
<i>static_methods</i>	5	1	1.00	1.00	1.00
<i>slides</i>	5	2	1.00	1.00	1.00
<i>polymorphism</i>	6	1	1.00	1.00	1.00
<i>anonymous_class</i>	5	2	1.00	1.00	1.00
<i>hineritance</i>	8	1	1.00	1.00	1.00
<i>interface</i>	7	2	1.00	1.00	1.00
<i>ask_prof</i>	13	3	1.00	1.00	1.00
	240	60	0.89	0.92	0.90

Accuracy 0.92

**In bold type the intents partially or totally not recognised correctly.*

Examples of intents correct predicted or not recognized

Example	Intent predicted	Intent correct	Correct?
hey	<i>affirm</i>	<i>greet</i>	NO
a dopo	<i>deny</i>	<i>goodbye</i>	NO
descrivimi il comparator	<i>comparator</i>	<i>comparator</i>	YES

Intent Confusion Matrix and Intent Prediction Confidence Distribution (IPCD)

Intent Prediction Confidence Distribution

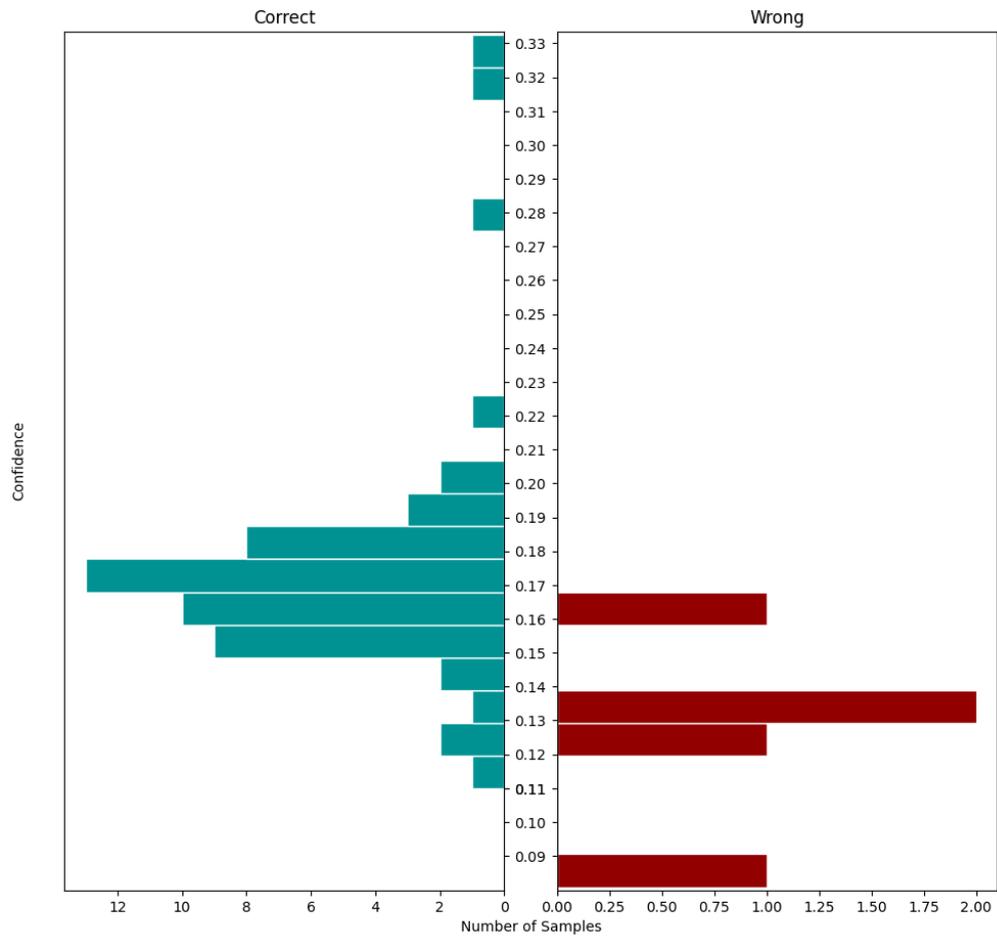


Figure 1: IPCD for BERT model

5.2.3 ConveRT Model Evaluation

The result of ConveRT model for *Intent Evaluation* are reported below:

Value of metrics for every intent

Intent *	#train	#test	precision	Recall	F1-score
<i>final</i>	3	1	1.00	1.00	1.00
<i>goodday</i>	4	1	1.00	1.00	1.00
<i>goodbye</i>	4	1	0.00	0.00	0.00
<i>deny</i>	4	1	0.33	1.00	0.50
<i>thanks</i>	4	1	1.00	1.00	1.00
<i>bot</i>	4	1	1.00	1.00	1.00
<i>GUI</i>	4	1	1.00	1.00	1.00
<i>upcasting</i>	4	1	1.00	1.00	1.00
<i>visibility</i>	4	1	1.00	1.00	1.00
<i>hineritance_constructor</i>	4	1	0.50	1.00	0.67
<i>super</i>	4	1	0.00	0.00	0.00
<i>class_object</i>	4	1	1.00	1.00	1.00
<i>comparator</i>	4	1	1.00	1.00	1.00
<i>observer_observable</i>	4	1	1.00	1.00	1.00
<i>greet</i>	4	1	0.00	0.00	0.00
<i>method_reference</i>	4	1	1.00	1.00	1.00
<i>objects_methods_doubt</i>	4	1	1.00	1.00	1.00
<i>static_final_doubt</i>	4	1	1.00	1.00	1.00
<i>visibility_variables_main</i>	4	1	1.00	1.00	1.00
<i>anonym_class_doubt</i>	4	1	1.00	1.00	1.00
<i>equals_doubt</i>	4	1	1.00	1.00	1.00
<i>arraysort_pattern_doubt</i>	4	1	1.00	1.00	1.00
<i>charAt_doubt</i>	4	1	1.00	1.00	1.00
<i>class_general_doubt</i>	4	1	1.00	1.00	1.00
<i>referent_methods_doubt</i>	4	1	1.00	1.00	1.00
<i>comparing_doubt</i>	4	1	1.00	1.00	1.00
<i>difference_dynaminc_bind_upcast</i>	4	1	1.00	1.00	1.00
<i>hineritance_tree</i>	4	1	1.00	1.00	1.00
<i>terminology</i>	4	1	1.00	1.00	1.00
<i>instanceof</i>	4	1	1.00	1.00	1.00
<i>this</i>	4	1	1.00	1.00	100
<i>pattern</i>	4	1	1.00	1.00	1.00

<i>functional_interface</i>	4	1	1.00	1.00	1.00
<i>default_methods</i>	4	1	1.00	1.00	1.00
<i>advantages</i>	4	1	0.50	1.00	0.67
<i>factory_methods</i>	4	1	1.00	1.00	1.00
<i>affirm</i>	5	1	0.00	0.00	0.00
<i>eclipse</i>	4	2	1.00	1.00	1.00
<i>override</i>	5	1	0.00	0.00	0.00
<i>dynamic_binding</i>	5	1	1.00	1.00	1.00
<i>casting</i>	5	1	1.00	1.00	1.00
<i>downcasting</i>	5	1	1.00	1.00	1.00
<i>class_abstract</i>	4	2	1.00	1.00	1.00
<i>comparable</i>	5	1	1.00	1.00	1.00
<i>iterator_iterable</i>	5	1	1.00	1.00	1.00
<i>lambda</i>	5	1	1.00	1.00	1.00
<i>static_methods</i>	5	1	1.00	1.00	1.00
<i>slides</i>	5	2	1.00	1.00	1.00
<i>polymorphism</i>	6	1	1.00	1.00	1.00
<i>anonymous_class</i>	5	2	1.00	1.00	1.00
<i>hineritance</i>	8	1	1.00	1.00	1.00
<i>interface</i>	7	2	1.00	1.00	1.00
<i>ask_prof</i>	13	3	1.00	1.00	1.00
	240	60	0.91	0.93	0.92
Accuracy 0.93					

*In bold type the intents partially or totally not recognised correctly

Examples of intents correct predicted or not recognized

Example	Intent predicted	Intent correct	Correct?
hey	<i>affirm</i>	<i>greet</i>	NO
a cosa serve il super nei costruttori	<i>hineritance_constructor</i>	<i>super</i>	NO
contatta un docente	<i>ask_prof</i>	<i>ask_prof</i>	YES

Intent Confusion Matrix and Intent Prediction Confidence Distribution (IPCD)

Intent Prediction Confidence Distribution

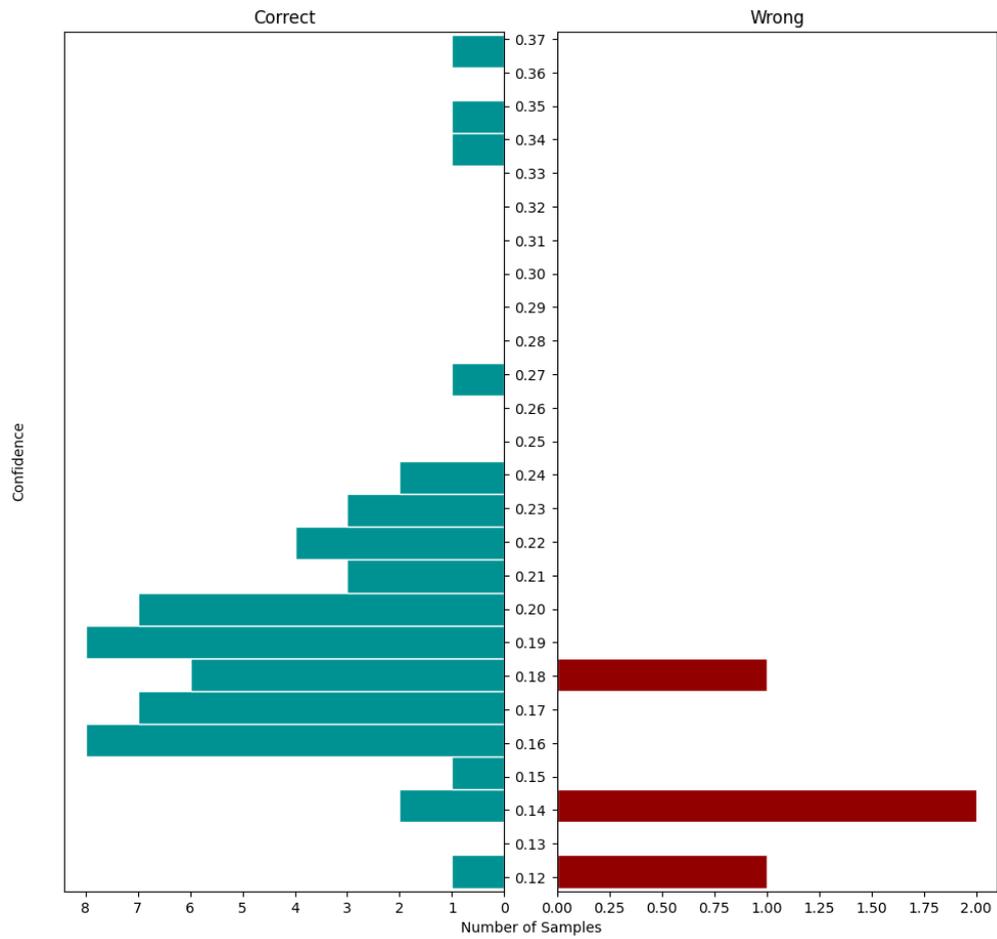


Figure 1: IPCD for ConveRT model

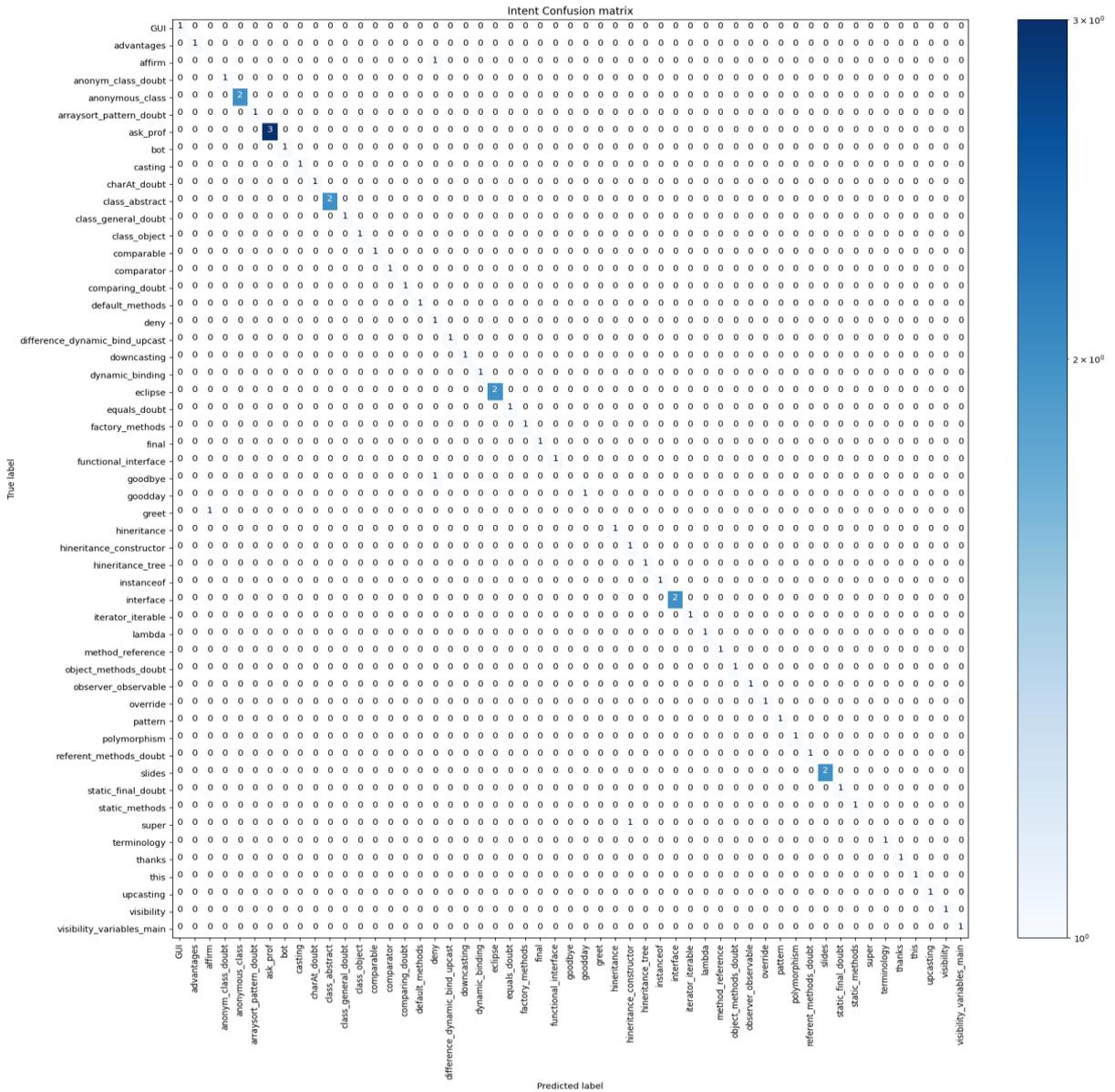


Figure 2: Intent Confusion Matrix for ConverRT model

5.2.4 Some considerations

As we can see, the three models are very accurate and exceed 90% accuracy. As for the precision metrics, recall and f1-score are also very similar, this means that the three models can be used almost completely indifferently.

The small differences between the three (the intents are predicted correctly in some models where in others they are not) could be due to the fact that their configurations are quite different, especially as regards the SpaCy which uses the NER (Name Entity Recognition) and the Italian as language (like other models), but in this case, which could be a disadvantage because in some of the examples used in the test set containing English-speaking words typically used in Computer Science, even in Italy and it might get confused to predict the intent correctly.

As for BERT and ConveRT, they are also both excellent models of Intent Recognition, and they share the use of Transformers layers and the technique of self-attention. Even if the BERT exploits its wide vocabulary that allows it to have a broader "perspective", the ConveRT, basing on tasks such as the Response Selector, Intent Recognition, in addition to being leaner and faster, is the best for all the evaluation metrics, even if briefly.

As for the confidence in the histograms, it is noted that it is not very high in all three cases. However, it must be said that the number of examples in the test set is not very large and the sentences with typically English-speaking technical terms do not help the model to have a high confidence value of the intents, especially in SpaCy case. But it can be seen that once again the ConveRT is slightly better, even if the other two still have a good reputation and could still be used as valid substitutes.

In below table you can see a resume of results:

Model	#train	#test	precision	recall	F1 score	accuracy
SpaCy	240	60	0.88	0.92	0.89	0.92
Bert	240	60	0.89	0.92	0.90	0.92
ConveRT	240	60	0.91	0.93	0.92	0.93

5.3 The Conversation Evaluation

To test the entire model, so the test stories, RASA offer also this command [47]:

```
rasa test --config [config file] --cross-validation --runs [num] --  
folds [num] --out [output directory]
```

With this configuration rasa uses **cross-validation** [52] that is a technique to estimate the metrics of predictions accurately. Train set and test set change for the value of folds written in command. The goal of the cross-validation is to test the model's ability to predict new data that was not used in estimating it, in order to see if there are present over-fitting or selection bias problems and generalize a dataset more than possible.

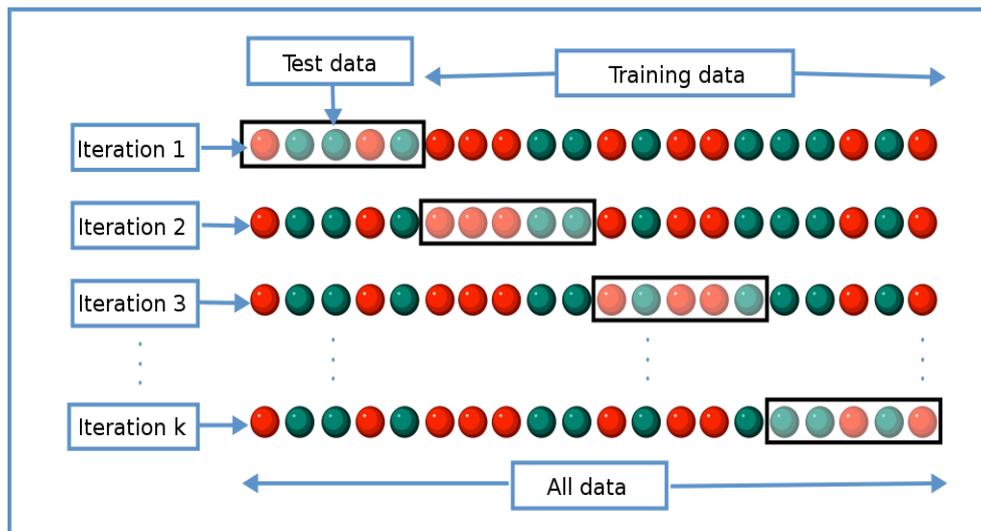


Figure 1: K-cross validation division of dataset in train and test set, from [52]

The test stories, in total 174 (considering all combinations of possible dialogues using checkpoints), saved in `tests/test_stories.yml` are written as normal stories in Rasa, the only difference is the user parameters, that simulate a user's input:

```

stories:
- story: stories1
steps:
- user: |
hello
intent: greet
- action: utter greet

```

The results of evaluation are saved in different files, including the intent evaluation files (see 5.2):

- `failed_test_stories.yml`: contains the stories predict incorrectly
- `story_confusion_matrix.png`: contains the confusion matrix with the actions predicted
- `story_report.json`: contains a report contains recall, f1-score, precision for every action
- Also offer:
 - **END_TO_END** level to analyse the accuracy of stories/conversations (the percentage of stories predicted correctly)
 - **ACTION** level to analyse the accuracy of actions, single turns in conversation (the percentage of actions predicted correctly)
 - **CV (folds)** level to analyse the train and test accuracy of intent predictions

5.3.1 SPACY Model Evaluation

The result of SpaCy model for *Action Accuracy (Stories Evaluation)* are:

Value of metrics

Evaluation Results on END-TO-END level (stories predicted)

Correct:	167 / 174
Accuracy:	0.960

Evaluation Results on ACTION level (action predicted of every part of stories)

Correct:	915 / 921
F1-Score:	0.993
Precision:	0.994
Accuracy:	0.994

Cross Validation evaluation for every Intent in stories (number of folds = 6):

Train Accuracy:	1.000
Train F1-score:	1.000
Train Precision:	1.000
Test Accuracy:	0.925
Test F1-score:	0.901
Test Precision:	0.886

Action Confusion Matrix and Intent Prediction Confidence Distribution (IPCD)

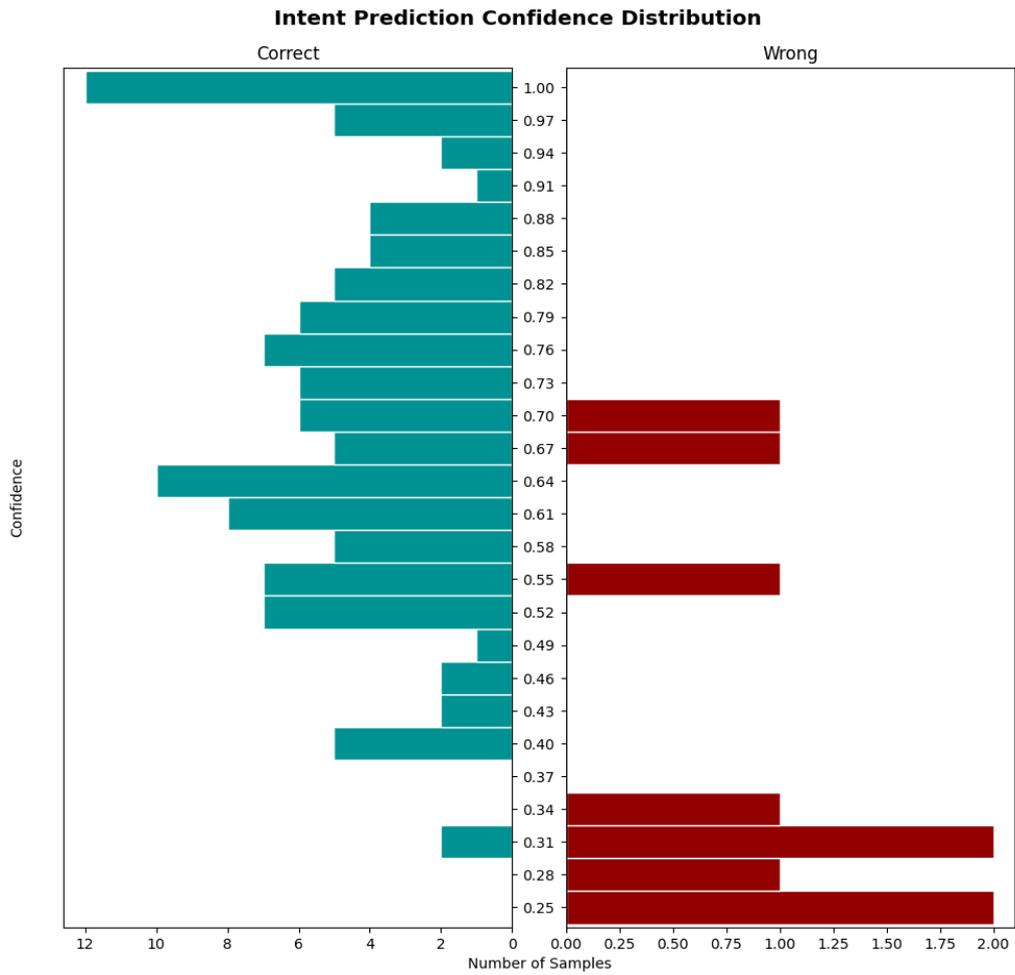


Figure 1: IPCD of Spacy model according test stories using cross validation

5.3.2 BERT Model Evaluation

The result of BERT model for *Action Accuracy (Stories Evaluation)* are:

Value of metrics

Evaluation Results on END-TO-END level (stories predicted)

Correct:	170 / 174
Accuracy:	0.977

Evaluation Results on ACTION level (action predicted of every part of stories)

F1-Score:	0.997
Precision:	0.998
Accuracy:	0.997

Cross Validation evaluation for every Intent in stories (number of folds = 6):

Train Accuracy:	1.000
Train F1-score:	1.000
Train Precision:	1.000
Test Accuracy:	0.992
Test F1-score:	0.989
Test Precision:	0.988

Action Confusion Matrix and Intent Prediction Confidence Distribution (IPCD)

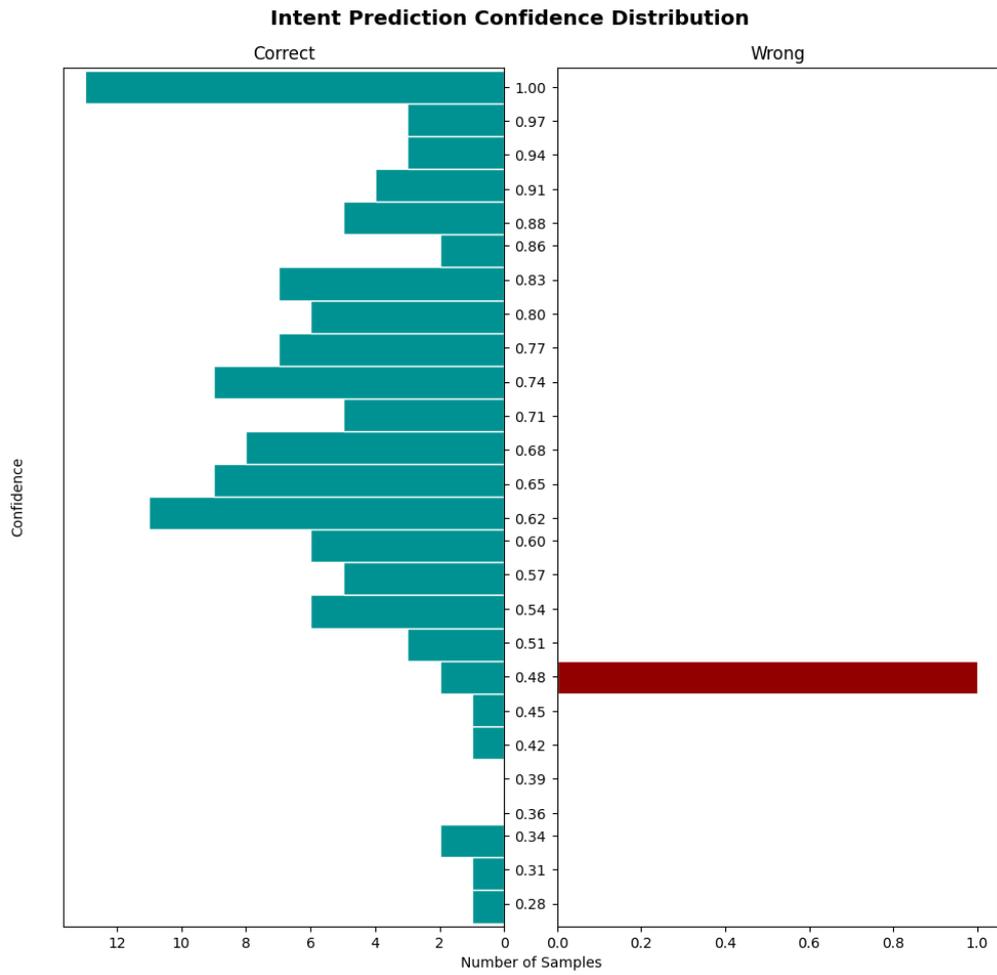


Figure 1: IPCD of BERT model according test stories using cross validation

5.3.3 ConveRT Model Evaluation

The result of ConveRT model for *Action Accuracy (Stories Evaluation)* are this:

Value of metrics

Evaluation Results on END-TO-END level (stories predicted)

Correct:	174 / 174
Accuracy:	1.000

Evaluation Results on ACTION level (action predicted of every part of stories)

Correct:	921 / 921
F1-Score:	1.000
Precision:	1.000
Accuracy:	1.000

Cross Validation evaluation for every Intent in stories (number of folds = 6)

Train Accuracy:	1.000
Train F1-score:	1.000
Train Precision:	1.000
Test Accuracy:	0.983
Test F1-score:	0.981
Test Precision:	0.982

Action Confusion Matrix and Intent Prediction Confidence Distribution (IPCD)

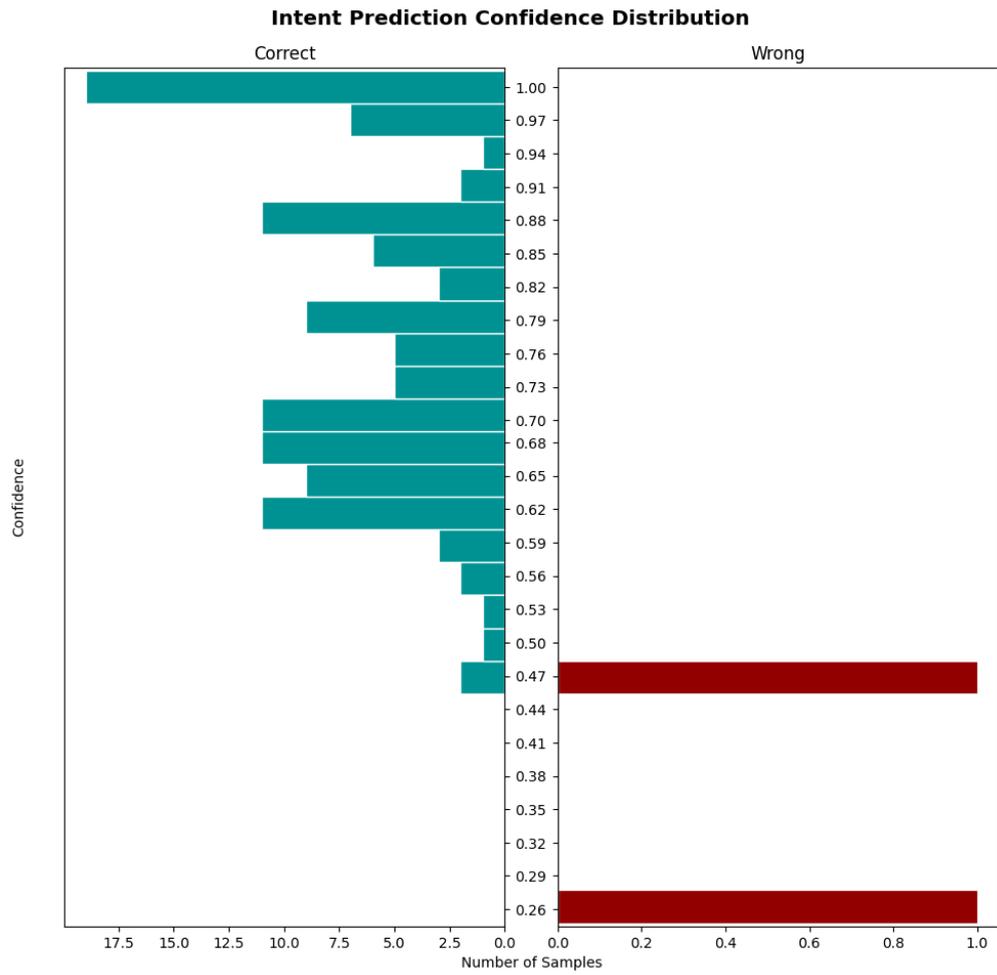


Figure 1: IPCD of ConvertRT model according test stories using cross validation

practically perfect. However, it must be said that the number of tests is not that high and therefore it is assumed that a significant increase in the data or arguments in the dataset could lead to a presumably lower accuracy value, albeit slightly.

This also applies to the ACTION level where the number of actions are analysed individually, without considering the stories as a whole, which therefore has a more detailed perspective of the analysis.

As regards the evaluation of the Intents, already analysed previously with the division of the NLU file into a fraction 80% train – 20% test, we can notice that using Cross-Validation technique (with number of folds = 6) on the whole set and with a higher number of data to be tested, the confidence value is significantly improved in all three cases.

However, we can say that in general the three models all seem very valid also at the level of stories testing and therefore, as the first level of Intent Recognition analysis had already announced, they can all be used in an almost completely indifferent way.

In below table you can see a resume of results for END-TO-END and ACTION level:

Model	#Conversation	#Action	% Correct conversation	Precision	F1 score	Accuracy
SpaCy	174	921	0.960	0.994	0.993	0.994
BERT	174	921	0.977	0.998	0.997	0.997
ConveRT	174	921	1.000	1.000	1.000	1.000

In below table you can see a resume of results for INTENT with CV(n=6) level:

Model	Test precision	Test F1 score	Test accuracy
SpaCy	0.886	0.901	0.925
BERT	0.998	0.989	0.992
ConveRT	0.982	0.988	0.983

6 Conclusions

6.1 Final considerations

After analysing the various models, we can then draw the necessary conclusions. As we have seen, using virtual assistants in education can offer numerous advantages:

- allow students to help in the learning process, also extending the possibility to help those who cannot afford to physically go to the school / university
- offer teachers support to students' requests in order to better manage their work and therefore offer a richer and broader offer
- then provide a set of answers to the students' frequent questions while guaranteeing the possibility of being able to discuss also with the teachers in person, always through the online communication channels in order to have more detailed feedback

So the advantages are available to both students and teachers, but always with a view to using the virtual assistant as an additional support to teaching and never as a substitute.

We have also seen that there are various frameworks that allow you to create these edu-bots in a fairly simple way. RASA was chosen, being one of the most used, very complete and open source and which allows the connection with the most popular communication channels such as Slack, and with its intuitive and clear configuration, it has allowed to create this educational assistant prototype for the "Programmazione ad Oggetti" course for the topic of Inheritance. Certainly, as mentioned previously, it will be possible to increase the training and testing data in the future as well as the topics to which the bot can offer an answer. It also allows the "Human Handoff", the possibility for a student to contact a teacher and then be able to speak in "first person" without the help of an Artificial Intelligence "brain". It also guarantees learning that can evolve over time thanks to RASA X, a broader version of RASA, which, thanks to Interactive Learning, offers the possibility of adding data taken from real conversations between virtual assistant and student or correcting wrong predictions.

Finally, three prediction models were analysed:

- **SpaCy**: exploiting the NER and allowing a wide range of other tasks such as Part-of-speech tagging, as well as allowing the support the Italian language, it is a valid prediction model both in terms of Entity Recognition and Stories Accuracy
- **BERT**: allows a wider range of vocabularies, same including also the Italian language, various versions for different tasks, including Question Answering, and the management of even large data using the encoders and decoders formed by various layers of Trasformers, a library widely used in Machine Learning especially for the tasks concerning the management of text, it is also excellent for this as a model of the Educational Assistant

- **ConveRT**: given its compactness, speed in learning and which was created to optimize *i.e.* Response Selector or Intent Recognition tasks used in our case, it is the best, even if slightly, in all the analysed metrics

This research thesis, also given the high results obtained from the various tests, and the Chatbot implemented seems an excellent starting point to one day be used in real environments such as schools or universities. Certainly further improvements and research are possible such as:

- adding data to the dataset regarding other object-oriented programming topics, teaching in PO course at the Polytechnic of Turin, and testing them with a larger amount of data to have more accurate evaluation metrics
- add data also for other courses in order to have a more generic Chatbot
- implement other predictive models, for example more geared towards modelling courses delivered in English rather than Italian
- update the database used for student requests in order to manage request of different courses (based second point of improvement)

Therefore it can be concluded that using a virtual bot in schools and universities can be very interesting and educational both for students, who "having fun" can ask questions and get immediate answers, and for teachers who allow a more detailed study of the requests of the students and therefore also create more specific paths based on the needs of their students. All this can be created using powerful frameworks like RASA, communication channels like Slack, and pre-trained models like SpaCy, BERT or ConveRT, and also a pinch of passion that must never be lacking, especially in the educational field.

References

- [1] Cairone Fiorentino. *Educational PO Bot*.
URL: https://github.com/caironefiorentino97/TESI_POLI_POBOT (cit. on page 8)
- [2] Wikipedia. *Virtual Assistant (en)*.
URL: https://en.wikipedia.org/wiki/Virtual_assistant (cit. on page 9)
- [3] Mobinius Editor. *How is AI advantageous as a Smart Assistant?*. July 6, 2020.
URL: <https://www.mobinius.com/blogs/advantages-of-ai-as-smart-assistant> (cit. on page 9)
- [4] Sachin Waikar (Stanford Engineering). *How an AI-based “Super Teaching Assistant” could revolutionize learning*. August 10, 2020.
URL: <https://engineering.stanford.edu/magazine/article/how-ai-based-super-teaching-assistant-could-revolutionize-learning> (cit. on page 10)
- [5] Dyllan Furness (digitaltrends). *A.I. teaching assistants could help fill the gaps created by virtual classrooms*. November 10, 2020.
URL: <https://www.digitaltrends.com/computing/how-ai-is-changing-education/> (cit. on page 11)
- [6] Wikipedia. *Slack*.
URL: <https://it.wikipedia.org/wiki/Slack> (cit. on page 13)
- [7] Innovate Yourself. *SLACK CHANNEL | EASY STEPS 2 CONNECT YOUR RASA CHATBOT TO SLACK*. November 3, 2020.
URL: <https://innovationyourself.com/connect-your-rasa-chatbot-to-slack-channel/> (cit. on page 13)
- [8] Rasa Technologies Inc. *RASA Open Source*.
URL: <https://rasa.com/open-source/> (cit. on page 16)
- [9] Rasa Technologies Inc. *RASA NLU*.
URL: <https://rasa.com/solutions/open-source-nlu-nlp/> (cit. on pages 16, 17)
- [10] Rasa Technologies Inc. *RASA contextual conversation*.
URL: <https://rasa.com/docs/rasa/contextual-conversations/> (cit. on pages 16, 17)
- [11] Rasa Technologies Inc. *RASA Interagations*.
URL: <https://rasa.com/docs/rasa/messaging-and-voice-channels/> (cit. on page 16)
- [12] Rasa Technologies Inc. *RASA Interagations on Slack*.
URL: <https://rasa.com/docs/rasa/connectors/slack> (cit. on page 17)

- [13] Rasa Technologies Inc. *RASA Connecting to Messaging and Voice Channels*.
URL: <https://rasa.com/docs/rasa/messaging-and-voice-channels/#testing-channels-on-your-local-machine> (cit. on page 19)
- [14] Mohammad Shahil. *How to create Chatbot using RASA*. July 22, 2020.
URL: <https://medium.com/voice-tech-podcast/how-to-create-chatbot-using-rasa-82954e141ae7> (cit. on page 22)
- [15] Andrea Mannini. *Anaconda*.
URL: <http://www.andreaminini.com/datascience/anaconda/> (cit. on page 22)
- [16] Rasa Technologies Inc. *RASA Components*.
URL: <https://rasa.com/docs/rasa/components> (cit. on page 22)
- [17] Rasa Technologies Inc. *RASA NLU*.
URL: <https://rasa.com/docs/rasa/nlu-training-data/> (cit. on page 22)
- [18] Rasa Technologies Inc. *RASA Stories*.
URL: <https://rasa.com/docs/rasa/stories/> (cit. on pages 22, 23, 25)
- [19] Rasa Technologies Inc. *RASA Rules*.
URL: <https://rasa.com/docs/rasa/rules/> (cit. on page 22)
- [20] Rasa Technologies Inc. *RASA Domains*.
URL: <https://rasa.com/docs/rasa/domain/> (cit. on pages 22, 25, 28, 29)
- [21] Rasa Technologies Inc. *Rasa Events*.
URL: <https://rasa.com/docs/action-server/events> (cit. on page 23)
- [22] Rasa Technologies Inc. *Rasa Forms*.
URL: <https://rasa.com/docs/rasa/forms/> (cit. on page 24)
- [23] Rasa Technologies Inc. *Rasa Model Configurations*.
URL: <https://rasa.com/docs/rasa/model-configuration/> (cit. on page 25)
- [24] Rasa Technologies Inc. *Rasa Policies*.
URL: <https://rasa.com/docs/rasa/policies> (cit. on page 26)
- [25] Rasa Technologies Inc. *Rasa Actions*.
URL: <https://rasa.com/docs/rasa/actions/> (cit. on page 27)
- [26] Rasa Technologies Inc. *Rasa Custom Actions*.
URL: <https://rasa.com/docs/rasa/custom-actions> (cit. on page 27)
- [27] Rasa Technologies Inc. *Rasa Tracker*.
URL: <https://rasa.com/docs/action-server/sdk-tracker> (cit. on pages 28, 29)
- [28] Rasa Technologies Inc. *Rasa run actions*.
URL: <https://rasa.com/docs/action-server/running-action-server/> (cit. on page 28)
- [29] Rasa Technologies Inc. *Rasa Default Actions*.
URL: <https://rasa.com/docs/rasa/default-actions> (cit. on page 28)

- [30] Rasa Technologies Inc. *Rasa Dispatcher*.
URL: <https://rasa.com/docs/action-server/sdk-dispatcher/> (cit. on page 29)
- [31] Rasa Technologies Inc. *Introduction to Rasa X*.
URL: <https://rasa.com/docs/rasa-x/> (cit. on pages 29, 30)
- [32] Rasa Technologies Inc. *Rasa X*.
URL: <https://rasa.com/rasa-x/> (cit. on page 30)
- [33] Rasa Technologies Inc. *Rasa X Installation Guide*.
URL: <https://rasa.com/docs/rasa-x/installation-and-setup/installation-guide> (cit. on page 30)
- [34] Rasa Technologies Inc. *Rasa X Local Mode Installation*.
URL: <https://rasa.com/docs/rasa-x/installation-and-setup/install/local-mode> (cit. on page 30)
- [35] Rasa Technologies Inc. *RASA X Share your Assistant*.
URL: <https://rasa.com/docs/rasa-x/user-guide/share-assistant> (cit. on page 31)
- [36] Wikipedia. *Python (en)*.
URL: [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)) (cit. on page 35)
- [37] Wikipedia. *Python (it)*.
URL: <https://it.wikipedia.org/wiki/Python> (cit. on page 35)
- [38] Sqlite.org. *SQLite Home Page*.
URL: <https://www.sqlite.org/index.html> (cit. on page 36)
- [39] Sqlite.org. *About SQLite*.
URL: <https://www.sqlite.org/about.html> (cit. on page 36)
- [40] Sqlite.org. *SQLite Features*.
URL: <https://www.sqlite.org/features.html> (cit. on page 36)
- [41] Docs python. *sqlite3*.
URL: <https://docs.python.org/3/library/sqlite3.html> (cit. on page 36)
- [42] Softeng Polito. *Corso di Programmazione ad Oggetti*.
URL: <https://softeng.polito.it/courses/09CBI/> (cit. on page 38)
- [43] Wikipedia. *SpaCy (en)*.
URL: <https://en.wikipedia.org/wiki/SpaCy> (cit. on page 40)
- [44] Ranko Mosaic (Medium.com). *Google BERT – Pre Training and Fine Tuning for NLP Tasks*. November 5, 2018.
URL: <https://ranko-mosaic.medium.com/googles-bert-nlp-5b2bb1236d78> (cit. on page 41, 42, 43)
- [45] Hugging Face. *dbmz/bert-base-italian-xxl-uncased*.
URL: <https://huggingface.co/dbmz/bert-base-italian-xxl-uncased> (cit. on page 43)

- [46] Matthew Henderson, Iñigo Casanueva, Nikola Mršić, Pei-Hao Su, Tsung-Hsien Wen and Ivan Vulić. *ConveRT: Efficient and Accurate Conversational Representations from Transformers*. April 29, 2020.
URL: <https://arxiv.org/pdf/1911.03688v2.pdf> (cit. on pages 44, 45)
- [47] RASA Technologies Inc. *Rasa Testing*.
URL: <https://rasa.com/docs/rasa/testing-your-assistant/> (cit. on pages 49, 50, 62)
- [48] Wikipedia. *Precision and Recall (en)*.
URL: https://en.wikipedia.org/wiki/Precision_and_recall (cit. on page 49)
- [49] Wikipedia. *F-Score (en)*.
URL: <https://en.wikipedia.org/wiki/F-score> (cit. on page 49)
- [50] Wikipedia. *Confusion matrix (en)*.
URL: https://en.wikipedia.org/wiki/Confusion_matrix (cit. on page 49)
- [51] developers.google.com (Machine Learning Crash Course). Classification: Accuracy.
URL: <https://developers.google.com/machine-learning/crash-course/classification/accuracy> (cit. on page 49)
- [52] Wikipedia. *Cross validation (en)*.
URL: [https://en.wikipedia.org/wiki/Cross-validation_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics)) (cit. on pages 62, 63)