MOT - THE MOVIE CHATBOT

Olga Pagnotta and Marta Soricetti

All the code can be found at the following link https://github.com/MotMovieBot/Mot.git.

Introduction

Chatbots: a first sight

Chatbots are Natural Language Processing models that enable computers to decode and even mimic the way humans communicate. They can complete tasks, achieving goals and delivering results.

Dialog systems' technology is almost everywhere these days, from the smart speakers at home to messaging applications in the workplace. Ideal chatbots should understand dialogue with human and answer in an appropriate way: such an architecture could be based on a trained neural network using real dialog and be able to converse with humans to deliver data or information. Essentially it should be trained on a corpus to answer to a specific stimulus. The latest AI chatbots are often referred to as "virtual assistants" or "virtual agents." They can use audio input, such as Apple's Siri, Google Assistant and Amazon Alexa, or interact with the user via SMS text messaging. Either way, the final user is able to ask questions about what he needs in a conversational way, and the chatbot can help refine his search through responses and follow-up questions.[1]

Today's AI chatbots use natural language understanding (NLU) to discern the user's need. Then they use advanced AI tools to determine what the user is trying to accomplish. These technologies rely on machine learning and deep learning to develop an increasingly granular knowledge base of questions and responses that are based on user interactions. This improves their ability to predict user needs accurately and respond correctly over time.
Anyway, it is important to highlight the fact that real chatbots are task oriented, meaning that they are trained in a specific domain to manage a certain activity.

AI chatbots are commonly used in social media messaging apps, standalone messaging platforms, or applications on websites. Some typical use cases include:

- Finding local restaurants and providing directions
- Defining fields within forms and financial applications
- Getting answers to healthcare questions and scheduling appointments
- Receiving general customer service help from a favourite brand
- Setting a reminder to do a task based on time or location
- Displaying real-time weather conditions and relevant clothing recommendations

[1] What is a chatbot?, available at https://www.ibm.com/topics/chatbots.

Generic architecture:

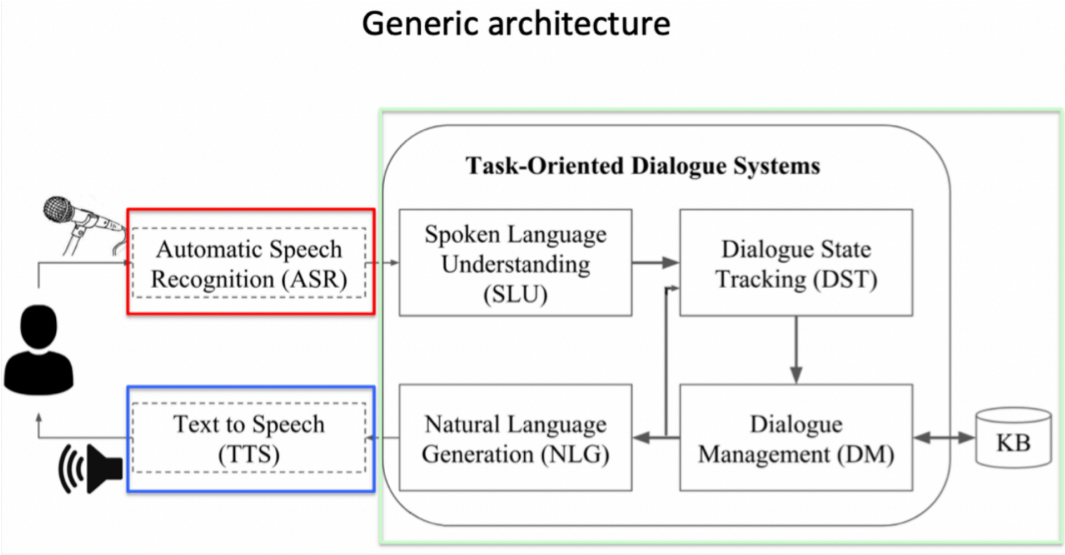

Figure 1 Generic dialog system architecture (Spoken Dialogue Systems set of slides of the NLP course)

First, on one side we find ASR (Automatic speech recognition). The system aims at finding the most likely sentence out of all sentences in the language L given some acoustic input O, which is a sequence of individual symbols or observations. This means that finding the most probable sentence W given some observation sequence O can be computed by taking the product of two probabilities for each sentence and choosing the sentence for which this product is greatest.

$$\bar{W} = \underset{W}{argmax}\, P(W \mid O) = \underset{W}{argmax}\, \frac{P(O \mid W)\, P(W)}{P(O)} = \underset{W}{argmax}\, P(O \mid W)\, P(W)$$ [2]

The main actors include a language model that computes the so-called prior probability P(W), that allows to estimate the most likely orderings of words in a given language; a pronunciation model for each word in that sequence; an acoustic model that computes the observation likelihood P(O|W), that allows to estimate the probability of an input sequence of acoustic observations given each possible words sequence W. When we receive some spoken input, our goal would be to find the most likely sequence of text that maximizes the words probability given a speech-acoustic input.[3]
Nowadays the approach is the end-to-end ASR: they take a sequence of audio inputs and return a sequence of textual outputs and all components of the architecture are trained jointly towards the same goal, while before each part of the ASR model was composed by a neural network, each of which had to be trained individually on specific tasks.
On the other hand, we find TTS (text-to-speech synthesis), for reading output to users. It is the inverse process of the ASR model.[4]

[2] Tamburini F. (2022). Neural Models for the Automatic Processing of Italian, Pàtron.
[3] *ibidem*
[4] *ibidem*

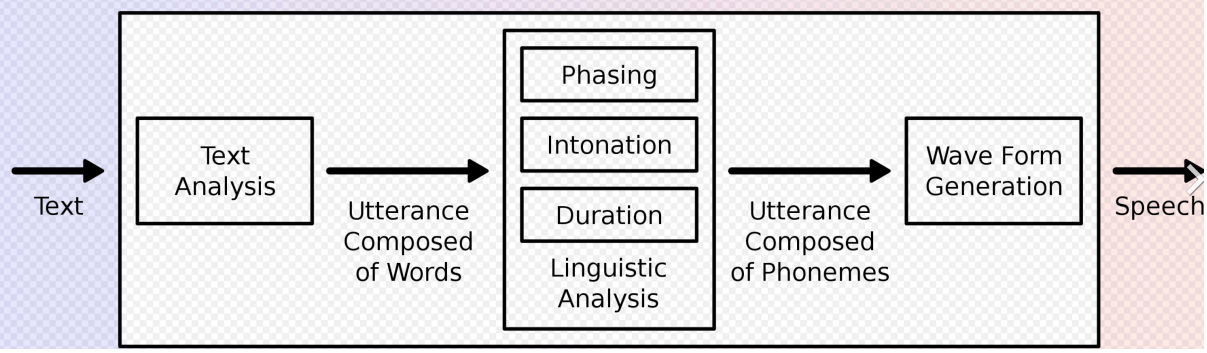

Figure 2 TTS model (Wikipedia)

The second step is intrinsically connected with NLP: Spoken Language Understanding (SLP) and Natural Language Generation (NLG) for synthesizing the written form of the answer.

Lastly, the main actors become the Dialogue State Tracking, to monitor the users' intentional states, and the Dialogue Management, which maintains some state variables, such as the dialog history, the latest unanswered question, etc. It sends instructions to other parts of the Dialog System to process the question, after having its formal representation, and to produce a useful answer to it.

Task oriented Chatbots:

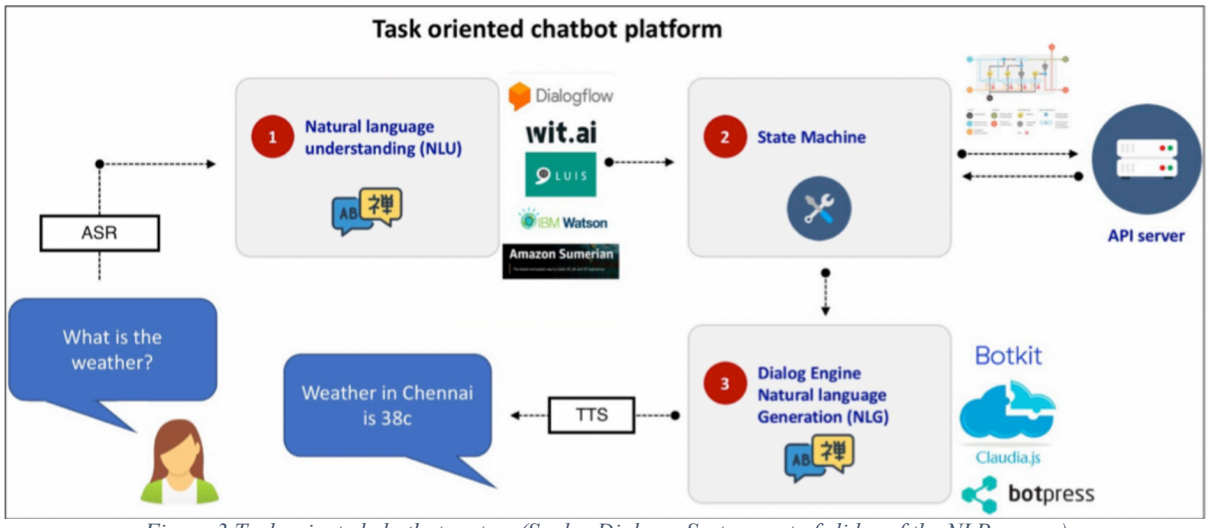

Figure 3 Task oriented chatbot system (SpokenDialogueSystems set of slides of the NLP course)

We have three main steps in a task oriented chatbot platform:

1. NLU. Natural language processing, and thus natural language understanding, can be delegated to some powerful platforms, which perform all the necessary actions to process user's inputs and convert it into structured data (such as intents or entities).
2. State Machine. It keeps the dialogue state, an important step for the creation of a continuous flow in the conversation. For example, it keeps track of the empty slots in the intent frame to be filled in a second moment through a series of question asked by the chatbot to obtain needed information.
3. NLG. Natural Language Generation for producing the sentence which will then be parsed by the TTS to generate the needed answer.

DialogFlow CX

Dialogflow is a natural language understanding platform used to design and integrate a conversational user interface into mobile apps, web applications, devices, bots, interactive voice response systems and related uses.[5]

Google Dialogflow recently introduced Dialogflow CX (Customer Experience), the version we decided to use. The older version of Dialogflow has been renamed to Dialogflow ES (Essentials). Dialogflow CX provides a new way of designing virtual agents, taking a state machine approach to agent design. This gives a clear and explicit control over a conversation, a better end-user experience, and a better development workflow. Some of the new functionalities that CX offer are:

- a visual flow interface, which allows to visualize flows and pages in a graph-like structure[6], to have a clearer view of the conversation;
- a state machine model[7], which allows developers to reuse intents, intuitively define transitions and data conditions, and handle supplemental questions;
- separate flows, allowing the developer to divide the agent into smaller conversation topics.[8]

It offers a wide variety of components to achieve the intended goal, but we will present only a few of them, the fundamental ones.

- *Agent*
  First, to create a dialog system on Dialogflow, we need to create a Dialogflow agent, a virtual agent that handles concurrent conversations with your end-users. It is a natural language understanding module that understands human language and its characteristics. Dialogflow is responsible for the first step in the task oriented chatbot framework, NLU, since it translates end-user text during a conversation to structured data.[9]
- *Flows*
  Flows are used for defining conversation topics in complex dialogs and the associated conversational paths. They provide better conversational control.[10]
- *Pages*
  A Dialogflow CX conversation (session) can be described and visualized as a state machine. The states of a CX session are represented by pages. For each flow, you define many pages, where your combined pages can handle a complete conversation on the topics the flow is designed for.[11]
- *Entity types*

---

[5] Dialogflow on Wikipedia, https://en.wikipedia.org/wiki/Dialogflow.
[6] Dialogflow editions, https://cloud.google.com/dialogflow/docs/editions.
[7] *ibidem*
[8] Dialogflow CX vs ES: A Complete Overview, available at https://chatbotsjournal.com/dialogflow-cx-vs-es-a-complete-overview-33580eca529c.
[9] Dialogflow documentation, https://cloud.google.com/dialogflow/cx/docs/basics.
[10] *ibidem*
[11] *ibidem*

Entity types are used to control how data from end-user input is extracted.

- *Parameters*
  Parameters are used to capture and reference values that have been supplied by the end-user during a session. Each parameter has a name and an entity type.
- *Intents*
  An intent categorizes an end-user's intention for one conversation turn.
- *Fulfillments*
  For an agent's conversational turn, the agent must respond to the end-user with an answer to a question, a query for information, or session termination. Your agent may also need to contact your service to generate dynamic responses or take actions for a turn. Fulfillment is used to accomplish all of this. In our chatbot they mostly take the form of customized payloads, supplied in a JSON format.[12]
- *Interactions*
  For each conversational turn, an interaction takes place. During an interaction, an end-user sends input to Dialogflow, and Dialogflow sends a response. We have used the Dialogflow API. The interaction takes places according to the following steps:
  1. The end-user types or says something, known as *end-user input*.
  2. The user interface receives the input and forwards it to the Dialogflow API in a detect intent request.
  3. The Dialogflow API receives the detect intent request. It matches the input to an intent or form parameter, sets parameters as needed, and updates session state.
  4. Dialogflow creates a detect intent response, using the static response defined in the agent. Dialogflow sends a detect intent response to the user interface.
  5. The user interface receives the detect intent response and forwards the text or audio response to the end-user.
  6. The end-user sees or hears the response.

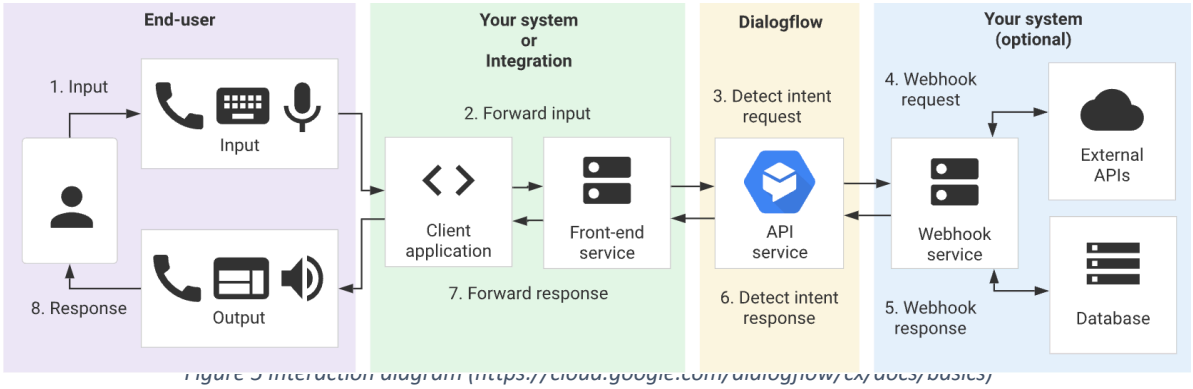

Figure 4 The interaction process in Dialogflow

[12] Dialogflow documentation, https://cloud.google.com/dialogflow/cx/docs/concept/fulfillment#payload.

MOT: purpose and workflow

Have you ever been stuck on the Netflix menu for minutes trying to find the right movie for your chill time? It happened to us many times, therefore we have thought of a possible solution for this problem, and we came up with Mot, the Movie Chatbot.
Mot is a personal assistant that simply suggests users a Netflix movie to watch.
There are two main types of chatbots: rule-based or AI, which are based on machine learning techniques and NLP.
We have decided to adopt the button-based framework to better present to the user the available data, thus the movies' categories and titles, which were taken from an already existing dataset of Netflix America, available at the following link https://www.datacamp.com/workspace/templates/dataset-python-netflix-movie-data.
"Menu-based," "Button-based," or "Rule-based" chatbots are simpler forms of chatbots with pre-defined rules. They're mostly used in instant messaging apps to perform automated customer support, presenting the information as a button menu or navigational system to guide the visitor along a predetermined path. These chatbots can detect common phrases for answering simple questions, such as booking a restaurant, buying movie tickets, purchasing online services or, as in our case, choosing a Netflix movie.
Guided by a decision tree, the virtual assistant gives customers a set of pre-defined options that lead to the desired answer.
If a query falls outside the pre-defined rules, the chatbot won't be able to assist further and will have to end the conversation.[13] These chatbots do not learn through interactions. Also, they only perform and work with the scenarios you train them for.
While rule-based bots have a less flexible conversational flow, these guard rails are also an advantage. You can better guarantee the experience they will deliver, whereas chatbots that rely on machine learning are a bit less predictable. They are also generally faster to train.[14] In addition, button-based chat bots have the potential advantage of presenting all available choices to the visitor. A menu-based bot is also generally easier to update. Rather than training the bot extensively for a new question, a new button menu and response for the topic that is being added is all that needs to be created. The bot does not require additional training when the menu has been added. There is also no ambiguity.[15]
Mot too uses these advantages to increase its performance and the overall user experience: it offers the possibility to choose among all the categories contained in the dataset, and therefore in the Netflix database, allowing the user to have a clearer view of the titles available.

[13] How do chatbots work and what is the technology behind them?, available at https://mindtitan.com/resources/guides/chatbot/how-do-chatbots-work/.
[14] Rule-Based Chatbots vs. AI Chatbots: Key Differences, available at https://www.hubtype.com/blog/rule-based-chatbots-vs-ai-chatbots#:~:text=Rule%2Dbased%20chatbots%20are%20also,based%20chatbots%20map%20out%20conversations.
[15] Types of Chat Bots, available at https://www.formilla.com/blog/types-chat-bots/.

The Workflow:

MOT deals with lots of movies and film categories so as a first step, we have created a schema of the dialog structure by drawing the different flows. From the *Default Start Flow*, the virtual agent moves to the *Movies flow*, which is able to lead the user to all the other 18 flows corresponding to the Netflix dataset's movies categories.

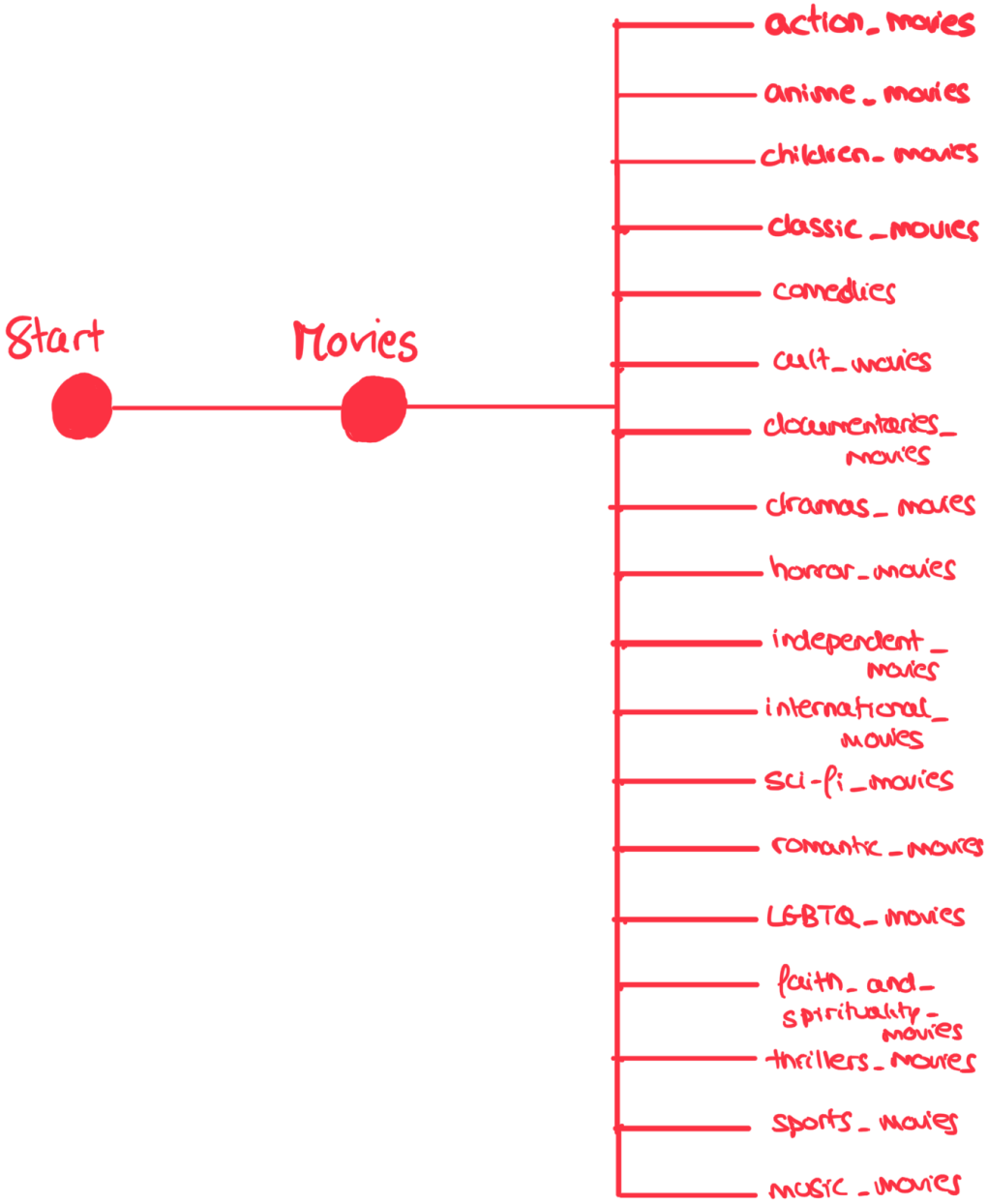

Figure 6 Schema of our Chatbot's flows

Entities

Afterwards, we have proceeded by defining all the categories and the whole list of titles of each category as custom entities basing on the following general schema:

Movies_categories
Dramas
Children & Family Movies
LGBTQ Movies
Anime Features
Documentaries
Music & Musicals
Action & Adventure
Classic Movies
Cult Movies
Thrillers
International Movies
Romantic Movies
Faith & Spirituality
Horror Movies
Sci-Fi & Fantasy
Sports Movies
Independent Movies
Comedies

Action_and_Adventure
America: The Motion Picture
Army of the Dead
Battle: Los Angeles
Battlefield Earth
Beowulf
Black Beach
Blade Runner: The Final Cut
Brick Mansions
Chappie
Charlie's Angels
Due Date
Green Lantern
Inception
Initial D
Into the Grizzly Maze
InuYasha the Movie: Affections Touching Across Time
Kingsglaive: Final Fantasy XV
Monster Hunter: Legends of the Guild
Mortal Kombat
Naruto the Movie: Ninja Clash in the Land of Snow
Pineapple Express
S.W.A.T.: Under Siege
Snowpiercer
Star Trek
Starship Troopers: Traitor of Mars
Team America: World Police
The Darkest Hour
The Interview
The Karate Kid
The Paper Tigers
The Tourist
The Whole Nine Yards
Time Trap
Underworld
Vampire Academy

Figure 7 General entities' schema

For this purpose, we have created a python script, create_entities.py, using the dfcx_scrapi,[16] the Python Dialogflow CX Scripting API, a high-level API that extends the official Google Python Client for Dialogflow CX.

```
from dfcx_scrapi.core.scrapi_base import ScrapiBase
from dfcx_scrapi.core.entity_types import EntityTypes
import pandas as pd
from google.cloud.dialogflowcx_v3beta1.types import entity_type
import re

creds_path = 'alert-vista-366713-15e1dff5c322.json'
agent_path = 'projects/alert-vista-366713/locations/europe-west3/agents/5baf2b13-8689-4784-bd58-38df4aa55395'

# DFCX Agent ID paths are in this format:
# 'projects/<project_id>/locations/<location_id>/agents/<agent_id>'

db = pd.read_csv("data_chatbotACTION.csv", sep=";")
```

Figure 8 Import of the needed libraries and reading of the csv

We have read the csv with pandas to manipulate it with the API library.
Then we started to create the entities by extracting all the action and adventure titles from the database to insert them into a list (list_action) and we have then created a list of dictionaries (entity_action) each containing the value, e.g. the title, and its synonyms, necessary for the creation of the entities but corresponding for us to the title itself.

```
#extract action and adventure
list_action=[]
i = 0
for row in db.iterrows():
    if "Movie" in db["type"][i]:
        if "Action & Adventure" in db["listed_in"][i]:
            list_action.append(db["title"][i])
    i += 1
entity_action = []
for each in list_action:
    each_entity_value = {}
    each_entity_value['value'] = str(each)
    each_entity_value['synonyms'] = [each]
    entity_action.append(each_entity_value)
```

Figure 9 The code for creating Action and Adventure entities

We have finally uploaded the entities into Dialogflow with the following command, where creds_path stands for a JSON containing the credentials of our Dialogflow service account.

```
# Instantiate your class object and pass in your credentials
i = EntityTypes(creds_path, agent_id=agent_path)

# Create entities
i.create_entity_type(kind = entity_type.EntityType.Kind.KIND_MAP, display_name="Action_and_Adventure", entities=entity_action)
```

Figure 10 Code to upload entities into Dialogflow

[16] dfcx-scrapi 1.5.1 documentation, available at https://pypi.org/project/dfcx-scrapi/.

Flows: structure

Default start flow

The root-flow for the creation of any chatbot is the *default start flow*. In Mot it consists in the following steps, which go from the start, welcoming the user by asking his/her name, to the movies' flow:

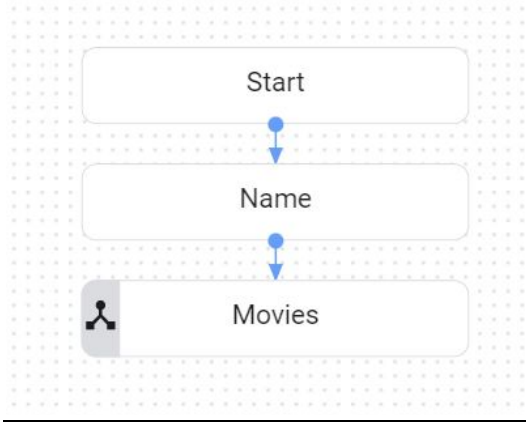

Figure 11 Default start flow

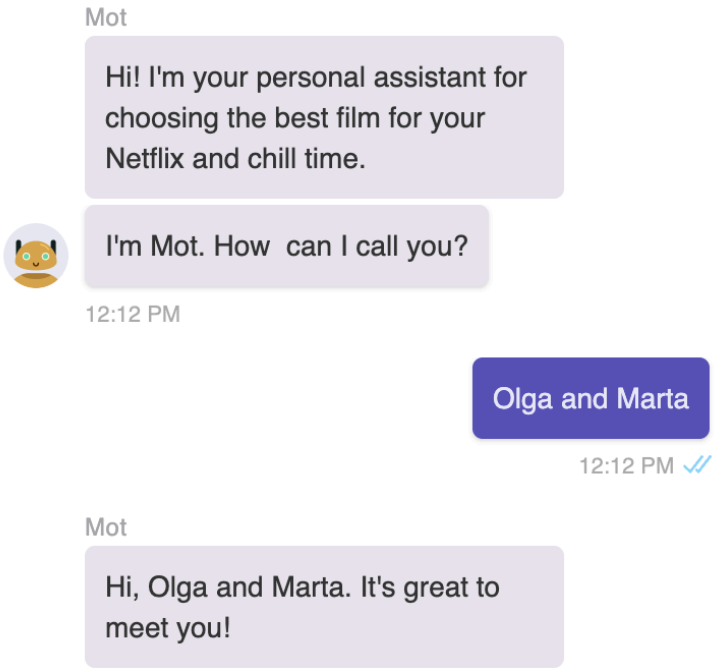

Figure 12 Welcome message from Mot

Movies flow

The second step is the choice of the movies categories by the user, following the below flow:

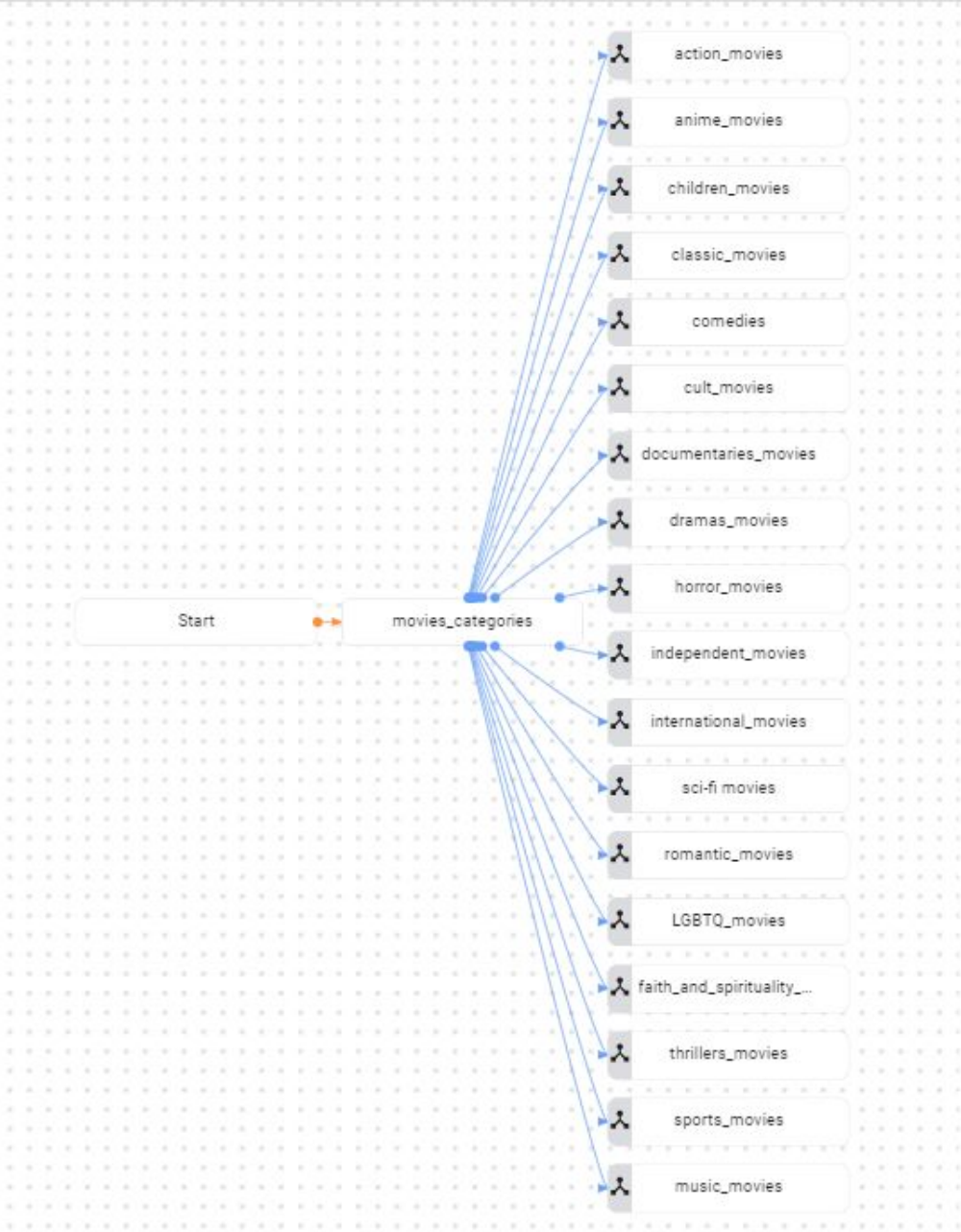

Figure 13 Movies flow

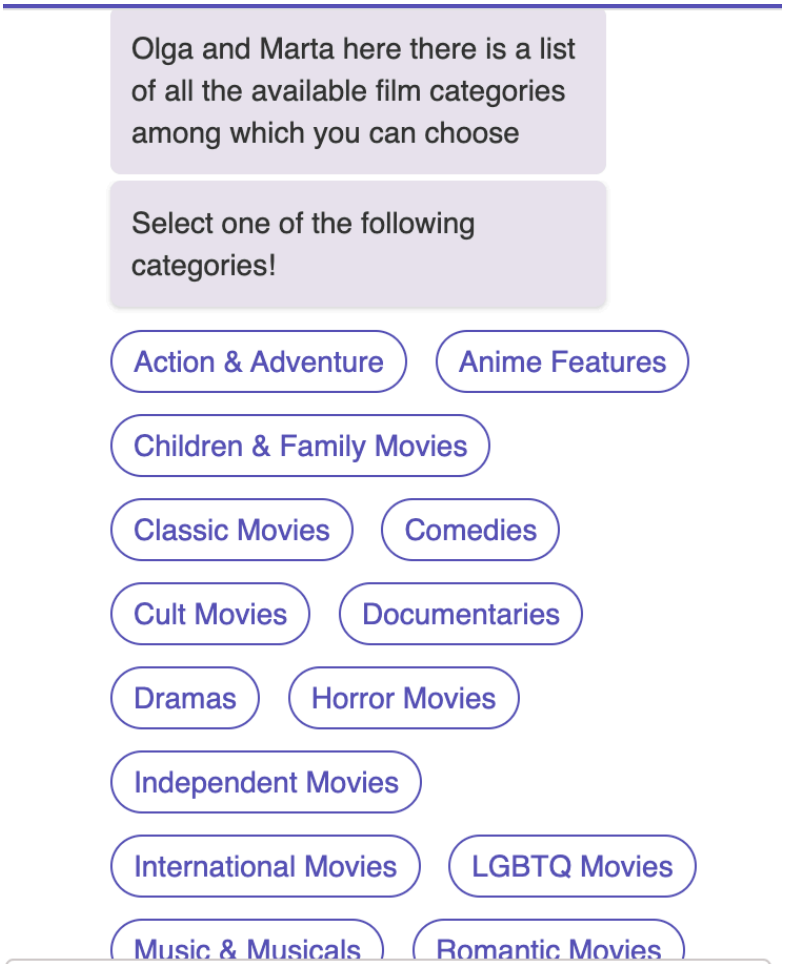

Figure 14 Choice of the movies categories

In the movies_categories page we have used a custom payload as entry fulfillment.
For interacting with our chatbot we have used Kommunicate,[17] a live-chat and chatbots powered customer support software. Kommunicate allows to add live chat on websites as widgets. It provides a codeless integration with Dialogflow, just by enabling the Dialogflow API of the project and creating a service account key. It also offers a wide range of commands for the customization of the widget that will be displayed into your project website. It also provides the way to implement rich messages using Dialogflow custom payload. We have decided to create different buttons with a set of suggested replies, corresponding to the movies categories, as shown in the following image:

[17] Kommunicate documentation, available at https://docs.kommunicate.io/docs/.

```
{
    "metadata": {
      "templateId": "6",
      "payload": [
        {
          "title": "Action & Adventure",
          "message": "I'd like to watch an Action & Adventure film"
        },
        {
          "message": "I'd like to watch an Anime Features film",
          "title": "Anime Features"
        },
        {
          "title": "Children & Family Movies",
          "message": "I'd like to watch one of the available Children & Family Movies"
        },
        {
          "title": "Classic Movies",
          "message": "I'd like to watch one of the available Classic Movies"
        },
        {
          "title": "Comedies",
          "message": "I'd like to watch one of the available Comedies"
        },
        {
          "title": "Cult Movies",
          "message": "I'd like to watch one of the available Cult Movies"
        },
        {
          "message": "I'd like to watch one of the available Documentaries",
          "title": "Documentaries"
        },
        {
          "message": "I'd like to watch one of the available Dramas",
          "title": "Dramas"
        },
        {
          "message": "I'd like to watch one of the available Horror Movies",
          "title": "Horror Movies"
        },
        {
          "title": "Independent Movies",
          "message": "I'd like to watch one of the available Independent Movies"
        },
        {
          "title": "International Movies",
          "message": "I'd like to watch one of the available International Movies"
        },
        {
          "message": "I'd like to watch one of the available LGBTQ Movies",
          "title": "LGBTQ Movies"
        },
        {
          "message": "I'd like to watch a film of the category Music & Musicals",
          "title": "Music & Musicals"
        },
        {
          "title": "Romantic Movies",
          "message": "I'd like to watch one of the available Romantic Movies"
        },
        {
          "message": "I'd like to watch a Sci-fi & Fantasy film",
          "title": "Sci-Fi & Fantasy"
        },
        {
            "message": "I'd like to watch one of the available Sports Movies",
            "title": "Sports Movies"
        },
        {
            "title": "Thrillers",
            "message": "I'd like to watch one of the available Thrillers"
        }
        ],
        "contentType": "300"
    },
    "message": "Select one of the following categories!",
    "platform": "kommunicate"
    }
```


Figure 15 Custom payload for the movies categories page

An example of a category-flow: action_movies

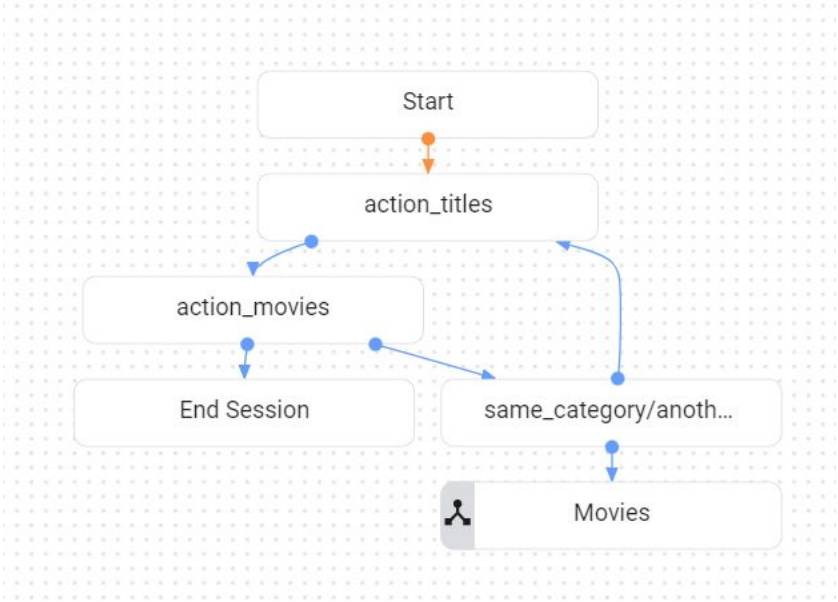

Figure 16 Action movies flow

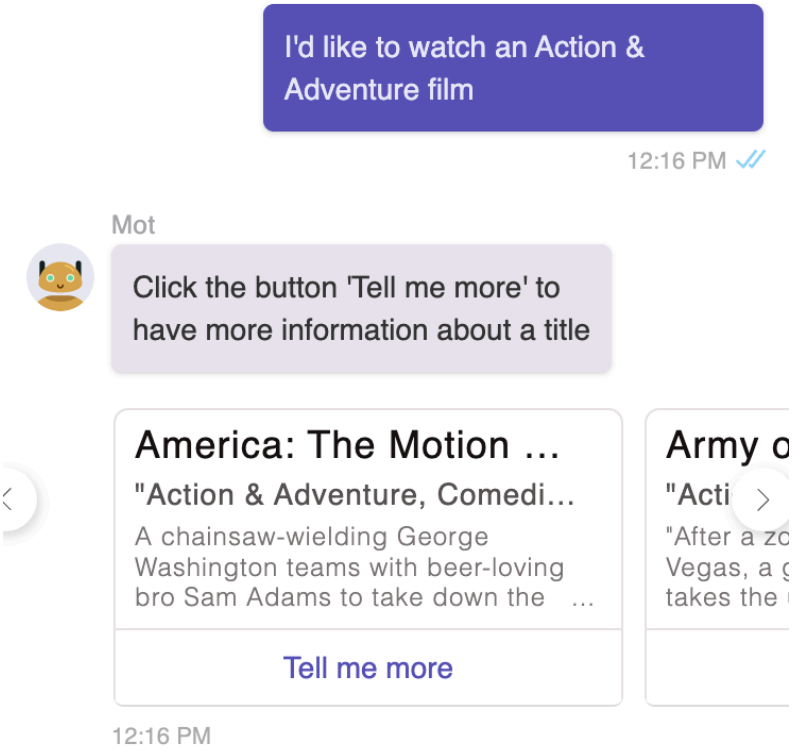

Figure 17 Action and Adventure movies' titles

The flow starts by presenting to the user all the available titles for the chosen category. For each title an abstract of the plot is displayed together with the other categories to which the title belongs. In addition, it offers the possibility to see a more detailed plot description, cast, director, duration, release date and related titles, obtained by the execution of a function that we will explain later in this paper.

The entry fulfillment of the action_titles page has been created through the following script, json_titles.py:

```
import pandas as pd
import json

def create_json_with_titles(csv_file, category1, category2, json_file):
    db = pd.read_csv(csv_file, sep=";")
    #extract action and adventure
    dic_action = dict()
    movies = []
    i = 0
    for row in db.iterrows():
        if category1 in db["type"][i]:
            if category2 in db["listed_in"][i]:
                dic_i = dict()
                buttons = dict()
                action = dict()
                message=dict()
                buttons_list=[]
                dic_i["title"] = db["title"][i]
                dic_i["subtitle"] = db["listed_in"][i]
                dic_i["description"] = db["description"][i]
                buttons["name"] = "Tell me more"
                message["message"] = "Tell me more about"+ " "+db["title"][i]
                action["payload"] = message
                action["type"] = "quickReply"
                buttons["action"] = action
                buttons_list.append(buttons)
                dic_i["buttons"] = buttons_list
                #dic_i["message"] = "Tell me more about"+" "+ db["title"][i]
                movies.append(dic_i)
        i += 1
    dic_action["payload"] = movies
    with open(json_file,'r') as jsonfile:
        json_content = json.load(jsonfile)
    json_content["metadata"]["payload"]= movies
    with open(json_file,'w') as jsonfile:
        json.dump(json_content, jsonfile, indent=4)
create_json_with_titles("data_chatbotACTION.csv", "Movie", "Music & Musicals", "music_movies_titles.json")
```

Figure 18 Entry fulfillment of the action_titles page

We have created a function, create_json_with_titles, which creates a JSON based on the Kommunicate JSON structure for creating card carousels.[18] Part of the information inserted in the JSON are taken directly from the CSV, while others are created by the function on the fly, as the name the button “tell me more”.

[18] Kommunicate documentation, available at https://docs.kommunicate.io/docs/message-types#card-carousel.

```
{
    "message": "Click the button 'Tell me more' to have more information about a title",
    "platform": "kommunicate",
    "metadata": {
        "contentType": "300",
        "templateId": "10",
        "payload": [
            {
                "title": "America: The Motion Picture",
                "subtitle": "\"Action & Adventure, Comedies\"",
                "description": "A chainsaw-wielding George Washington teams with beer-loving bro Sam Adams to take down the Brits in a tongue-in-cheek riff on the American Revolution.",
                "buttons": [
                    {
                        "name": "Tell me more",
                        "action": {
                            "payload": {
                                "message": "Tell me more about America: The Motion Picture"
                            },
                            "type": "quickReply"
                        }
                    }
                ]
            },
            {
                "title": "Army of the Dead",
                "subtitle": "\"Action & Adventure, Horror Movies\"",
                "description": "\"After a zombie outbreak in Las Vegas, a group of mercenaries takes the ultimate gamble by venturing into the quarantine zone for the greatest heist ever.\"",
                "buttons": [
                    {
                        "name": "Tell me more",
                        "action": {
                            "payload": {
                                "message": "Tell me more about Army of the Dead"
                            },
                            "type": "quickReply"
                        }
                    }
                ]
            },
```


Figure 19 A part of the JSON file obtained for the Action and Adventure movies and then used as custom payload for the entru fulfillment of the page action_titles

As anticipated above, after the selection of one title, a button “tell me more” is available for giving more information about the chosen movie. This is how the message looks like:

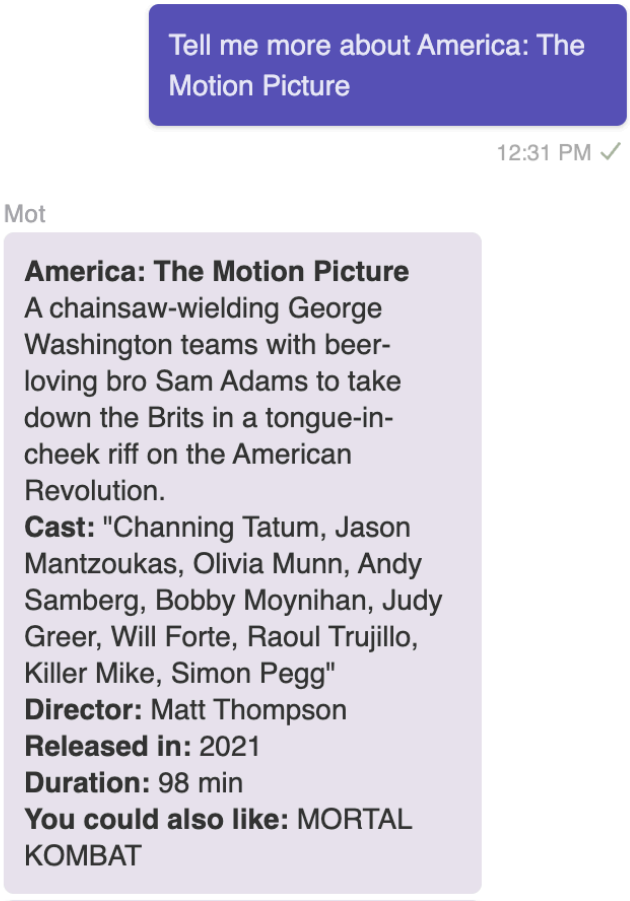

Figure 20 The full description of a title

For the creation of the movie full description, we have used a script, create_scheda_film.py, using the same approach as above and creating an html code on the fly for displaying the csv information in a clearer way.

```python
import pandas as pd
import json
from similarity import response

def create_scheda_film(csv_file, category1, category2, json_file):
    db = pd.read_csv(csv_file, sep=";")

    #extract action and adventure
    dic_titles = dict()
    i = 0
    for row in db.iterrows():
        if category1 in db["type"][i]:
            if category2 in db["listed_in"][i]:
                dic_i = dict()
                dic_i["messageType"] = "html"
                dic_i["platform"] = "kommunicate"
                suggested_titles = response(db["title"][i])
                dic_i["message"] = f"""<b>{db["title"][i]}</b></br><p>{db["description"][i]}</p><p><b>Cast: </b>{db["cast"][i]}</p><p><b>Director: </b>
                 {db["director"][i]}</p><p><b>Released in: </b>{db["release_year"][i]}</p><p><b>Duration: </b>{db["duration"][i]}</p><p><b>You could also like: </b>
                 {suggested_titles}</p>"""
                dic_titles[db["title"][i]]= dic_i
        i += 1


    json_content=dic_titles

    with open(json_file,'w') as jsonfile:
        json.dump(json_content, jsonfile, indent=4)

create_scheda_film("data_chatbotACTION.csv", "Movie", "Action & Adventure", "scheda_film_action_movies.json")
```

Figure 21 Script for the creation of the movie full description


```
{
    "America: The Motion Picture": {
        "messageType": "html",
        "platform": "kommunicate",
        "message": "<b>America: The Motion Picture</b></br><p>A chainsaw-wielding George Washington teams with beer-loving bro Sam Adams to take down the Brits in a tongue-
in-cheek riff on the American Revolution.</p><p><b>Cast: </b>\"Channing Tatum, Jason Mantzoukas, Olivia Munn, Andy Samberg, Bobby Moynihan, Judy Greer, Will Forte, Raoul
Trujillo, Killer Mike, Simon Pegg\"</p><p><b>Director: </b>Matt Thompson</p><p><b>Released in: </b>2021</p><p><b>Duration: </b>98 min</p><p><b>You could also like: </b>MORTAL
KOMBAT</p>"
    },
    "Army of the Dead": {
        "messageType": "html",
        "platform": "kommunicate",
        "message": "<b>Army of the Dead</b></br><p>\"After a zombie outbreak in Las Vegas, a group of mercenaries takes the ultimate gamble by venturing into the quarantine
zone for the greatest heist ever.\"</p><p><b>Cast: </b>\"Dave Bautista, Ella Purnell, Omari Hardwick, Garret Dillahunt, Ana de la Reguera, Theo Rossi, Matthias
Schweigh\u00f6fer, Nora Arnezeder, Hiroyuki Sanada, Tig Notaro, Ra\u00fal Castillo, Huma Qureshi, Samantha Win, Richard Cetrone, Michael Cassidy\"</p><p><b>Director: </b>Zack
Snyder</p><p><b>Released in: </b>2021</p><p><b>Duration: </b>148 min</p><p><b>You could also like: </b>HOSTEL: PART III</p>"
    },
    "Battle: Los Angeles": {
        "messageType": "html",
        "platform": "kommunicate",
        "message": "<b>Battle: Los Angeles</b></br><p>\"Led by their skillful staff sergeant, a platoon of gutsy Marines fights to protect all humankind from astonishingly
powerful aliens.\"</p><p><b>Cast: </b>\"Aaron Eckhart, Michelle Rodriguez, Ramon Rodriguez, Bridget Moynahan, Ne-Yo, Michael Pe\u00f1a, Lucas Till, Cory Hardrict, Adetokumboh
M'Cormack, Jim Parrack\"</p><p><b>Director: </b>Jonathan Liebesman</p><p><b>Released in: </b>2011</p><p><b>Duration: </b>116 min</p><p><b>You could also like: </b>LETTERS TO
JULIET</p>"
    },
    "Battlefield Earth": {
        "messageType": "html",
        "platform": "kommunicate",
        "message": "<b>Battlefield Earth</b></br><p>\"In the year 3000, an alien race known as the Psychlos devastate planet Earth and force the surviving human population
into slavery.\"</p><p><b>Cast: </b>\"John Travolta, Barry Pepper, Forest Whitaker, Kim Coates, Sabine Karsenti, Richard Tyson, Kelly Preston, Michael MacRae, Shaun Austin-
Olsen, Tim Post, Michael Byrne\"</p><p><b>Director: </b>Roger Christian</p><p><b>Released in: </b>2000</p><p><b>Duration: </b>118 min</p><p><b>You could also like: </b>THIS
IS THE LIFE</p>"
    },
```


Figure 22 a part of the obtained JSON file with the list of all the full descriptions for each Action and Adventure movie

The similarity function:

For the suggestion of similar titles, we have used a script, similarity.py, which is based on the movie description.

First, we have removed punctuation and transformed every description to lowercase. We have then created a new dataframe with the updated data (title and corresponding “cleaned” description). By importing the ScikitLearn library we have used the transformer called TfidfVectorizer in the module called feature_extraction.text for vectorizing with TF–IDF scores. TF-IDF (TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY) Vectorization consists in the computation of two values:

- Term Frequency (TF): The number of times a word appears in a document divided by the total number of words in the document. Every document has its own term frequency.

- Inverse Data Frequency (IDF): The log of the number of documents divided by the number of documents that contain the word. Inverse data frequency determines the weight of rare words across all documents in the corpus.

We have then computed the cosine similarity between the vectors obtained: the movie plots have been transformed as vectors in a geometric space. Therefore, the angle between two vectors represents the closeness of those two vectors. Cosine similarity calculates similarity by measuring the cosine of the angle between two vectors.[19]

```
import nltk, nltk.tokenize, nltk.corpus, nltk.stem
from nltk.tokenize import word_tokenize, RegexpTokenizer
from nltk.corpus import stopwords, wordnet
import pandas as pd
#lowercase text
def to_lower(text):
    return ' '.join([w.lower() for w in word_tokenize(text)])
#remove stopwords
def remove_stopwords(text):
    no_stop = text.split()
    stopwordz = stopwords.words('english')
    for word in no_stop:
        for stopword in stopwordz:
            if word == stopword:
                no_stop.remove(word)
    return ' '.join(no_stop)
#remove punctuation
def strip_punctuation(text):
    tokenizer = RegexpTokenizer(r'\w+')
    return ' '.join(tokenizer.tokenize(text))
#preprocess text
def preprocess(text):
    lower_text = to_lower(text)
    text_no_stopwords = remove_stopwords(lower_text)
    text_no_punct = strip_punctuation(text_no_stopwords)
    return text_no_punct
#application of the function preprocess(text) to our corpus
verses_no_punct=[]
db = pd.read_csv("data_chatbotACTION.csv", sep=";")
for x in db['description']:
    x_preprocessed= preprocess(x)
    verses_no_punct.append(x_preprocessed)
new_data= {'title': db["title"], 'description': verses_no_punct}
new_df=pd.DataFrame(new_data)
```

Figure 23 Function "preprocess" to remove punctuation and transform the text to lowercase

[19] Movie recommender, available at https://www.geeksforgeeks.org/movie-recommender-based-on-plot-summary-using-tf-idf-vectorization-and-cosine-similarity/.

```python
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
TfidfVec = TfidfVectorizer(ngram_range=(1,2))
#Convert the text to a matrix of TF-IDF features
tfidf = TfidfVec.fit_transform(new_df['description'])
feature_names = TfidfVec.get_feature_names()
#Creating a cosine similarity matrix from TF-IDF vectors
cosine_sim = cosine_similarity(tfidf, tfidf)
def response(title):
    recommended_content_titles=[]
    content_titles = pd.Series(new_df['title'])
    content_index = content_titles[content_titles == title].index[0]
    similarity_scores_list = pd.Series(cosine_sim[content_index]).sort_values(ascending=False)
    similar_content_indices = list(similarity_scores_list.iloc[1:2].index)
    for content in similar_content_indices:
        recommended_content_titles.append(str(new_df['title'][content]).upper())
    return(', '.join(recommended_content_titles))
```

Figure 24 Function "response" to find similar movies according to their description

After having showed the full description, Mot asks the user if he/she wants to choose another title or if he/she wants to end the session. If the user decides to end the session, an ending sentence is showed; otherwise, Mot asks the user to stay in the same category or to change It to receive other titles' suggestions.

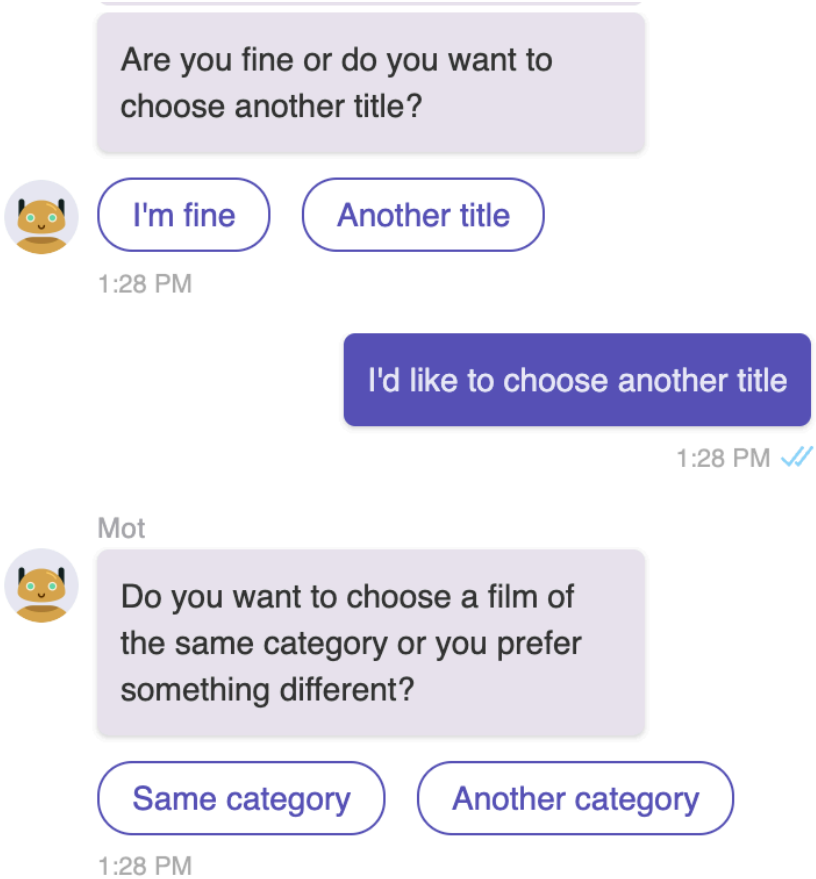

Figure 25 Choose another title message

If the user chooses the "Same category" button, Mot shows again the same list as before. If instead the "Another category" button is clicked, the process starts over from the Movies flow showing again all the available categories.

Routes

The structure of the payload for each movie title's full description explained above has been inserted as fulfillment of each route of each page, named in the form "*categoryX_titles*". For the explanation we consider again the action_titles page.

For creating the various categories' routes, we have used another python script, create_routes.py, using again the same API as above, dfcx_scrapi , and pandas to access the csv file containing the Netflix dataset. We have created a function, called "create_routes", that is shown here below:

```
from dfcx_scrapi.core.pages import Pages
from dfcx_scrapi.tools.maker_util import MakerUtil
import pandas as pd
```

Figure 26 Import of the needed libraries and dfcx_scrapi.core classes

```
def create_routes(creds_path, csv_file, intent_id, target_page_id, category1, category2, entity_name, page_id_to_update):

    db = pd.read_csv(csv_file, sep=";")

    c = Pages(creds_path = creds_path)

    j = MakerUtil()

    i = 0
    for row in db.iterrows():
        if category1 in db["type"][i]:
            if category2 in db["listed_in"][i]:
                title = db["title"][i]
                condition = f'''$session.params.{entity_name} = "{title}"'''
                route = j.make_transition_route(intent=intent_id,condition=condition,target_page=target_page_id)
                dsf = c.get_page(page_id_to_update)
                dsf.transition_routes.append(route)
                c.update_page(page_id=page_id_to_update, obj=dsf, transition_routes=dsf.transition_routes)
        i += 1

creds_path = 'alert-vista-366713-15e1dff5c322.json'
csv_file = "data_chatbotACTION.csv"
intent_id_action = "projects/alert-vista-366713/locations/europe-west3/agents/5baf2b13-8689-4784-bd58-38df4aa55395/intent
target_page_id_action = "projects/alert-vista-366713/locations/europe-west3/agents/5baf2b13-8689-4784-bd58-38df4aa55395/f
category1_action = "Movie"
category2_action = "Action & Adventure"
entity_name_action = "Action_and_Adventure"
page_id_to_update_action = "projects/alert-vista-366713/locations/europe-west3/agents/5baf2b13-8689-4784-bd58-38df4aa5539
```

Figure 27 Function "create_routes" and its application for the page action_titles

In this function we have used the dfcx_scrapi.tools class "MakerUtil", that has been thought for creating CX objects, and in particular the method "make_transition_route", that allows the developer to create a single Route object suitable for being used interchangeably with Pages or Route Groups as needed.[20]

[20] dfcx-scrapi repository, https://github.com/GoogleCloudPlatform/dfcx-scrapi/blob/main/src/dfcx_scrapi/tools/maker_util.py.

Talking about all the routes created for the page action_titles, we have used as parameters of this method:

- the id of the intent created in order to recognize movies' titles belonging to the category Action and Adventure, thus all the reference values of the entity type "Action_and_Adventure";
- the condition to evaluate on the root, that takes into account the title of the movie chosen by the user;
- the id of the page action_movies, that is the target page to transition to.

Then the method "get_page" of the dfcx_scrapi.core class "Pages" has been used to get the CX Page object based on the provided Page ID, that in our case is stored in the variable "page_id_to_update_action", and the method "transition_routes" to add to it the newly created route. Finally we have updated the Page object in our agent.

Intents

Routes in Dialogflow CX have two requirements, in addition to the condition one, there is also the intent requirement,[21] that is, an intent which must be matched to end-user input for the current conversational turn.

We have used mainly two kinds of intents, one to catch the Movies_categories entities' values in the user input and the other to match the titles of each categories, such as catch_action_movies_titles.

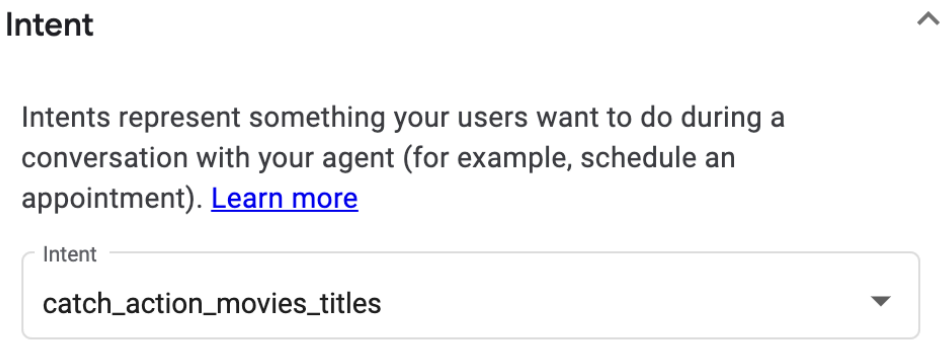

Figure 28 Intent for movies titles

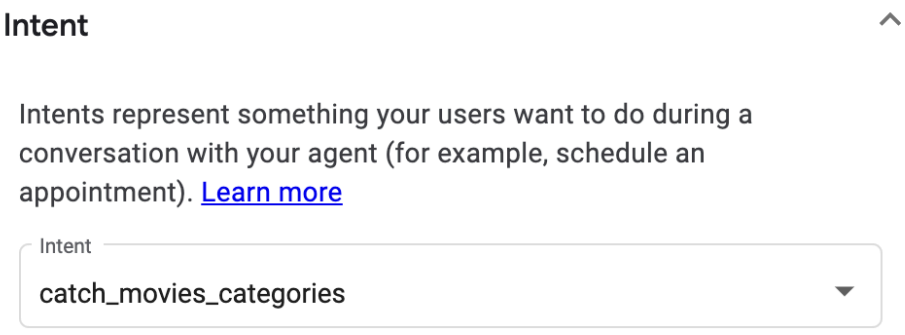

Figure 29 Intent for movies categories

[21] Dialogflow documentation, https://cloud.google.com/dialogflow/cx/docs/concept/intent.

Faced problems

For building Mot, we have been forced to reduce the dimensions of the original Netflix dataset. At the beginning, our intention was to build a chatbot to suggest not just Netflix movies, but also Netflix Tv Series included in the original dataset. But we had to face the Dialogflow CX limitations: the maximum number of flows per agent is 20[22], so including also all the tv series categories would have meant to create many more flows, considering that, taking into consideration just the movies, we had already built 19 flows.
We had also to reduce the number of movies' titles per categories because the maximum number of per-minute "all other requests" is 60, so we haven't been able to execute the function "create_routes" with more than 60 titles. In fact, the function "create_routes", as explained above, creates different routes with different conditions based on the movie's title chosen by the user for the same page. It was also funny that for executing the function and consequently building the routes for different "CategoryX_titles" pages we had to wait that the current minute passed, because the Dialogflow per-minute quotas are refreshed every 60 seconds on the minute.[23]

[22] Dialogflow documentation, https://cloud.google.com/dialogflow/quotas#count_limits
[23] Dialogflow documentation, https://cloud.google.com/dialogflow/quotas