Interaction

1. Overview

You can understand your voice assistant as a composition of different parts:

- `policy` : meaning the way it reacts to a certain human utterance, which is based on the activated/hacked/coded skills at its disposal, depending also on his state.
- 'limb', the way he talks: through the gpt-2 ML model, fine-tuned on the data you fed him.
- 'state': encompasses both its memory and some custom parameters (specified by the human before an interaction, randomized or default).

These processes will be detailed below. The exchange with your voice assistant consists of three parts: interaction, growth (with machine learning), and self-Quest (optional).

Interaction

The direct interaction with your voice assistant has several steps:

1. Voice assistant 'hears' what you say (your voice is converted into text, and in turn classified as intent; "what is being asked of me?")
2. If an intent is detected, it will trigger the corresponding skill, depending on the skills and triggers you will choose for your project. Default implemented skills include Wikipedia, DuckDuckGo, remembering what you said, record/play, laughing, etc. Cf. Below for more details about Mycroft skills or customizable skills.
3. If no intent is detected, VA will respond through machine learning (as a so-called 'fallback skill' we implemented). The mode of this response and other parameters can be provided as input of the interaction, default ones, or randomized.
4. If specified, your conversation is stored in the memory and used for further training of the machine learning algorithm (Mycroft's expressivity). This conversation is recorded in the file whatVAHeard.txt, inside the data folder.

Growth

Your voice assistant's verbal way of communicating (in the fallback Skill implemented) will be progressively influenced by your interactions with it and the data you choose to 'feed' it. This is done with the help of machine learning, and more precisely by fine-tuning a gpt-2 language model through the Jupyter notebook on the data you provide. To have access to more computational resources, we use Google Collaboratory to train this model. For this step, look at the Google Drive, in the limb folder, and follow the quick Machine Learning guide there.

Self Quest (optional)

This feature is a trace of our (Claire and Vytas) Unfamiliar Convenient project and can provide a simple visualisation of what the VA heard. For people who have less time available, it is advised to skip the Self Quest and focus on their VAs :)
The voice assistant grows its knowledge of 'Self', based on your interactions with it. Note that the 'Self' concept of your VA is a dynamic networked entity dependent on your interactions with it. The Self is seen as a set of concepts (nodes of a graph), with various weights and connections between them called edges. By running selfQuest.py script, words from previous conversations will be extracted and looked up on Wikipedia, and compared to the concepts already present within the 'Self' (semantic proximity analysed). If the newly obtained concepts are close enough, they will be appended with a certain importance (weight).

2. Getting Started

1. Once you have defined your project, it is time to interact with your VA. If it is your first interaction, you can simply run it and test a few skills randomly (only the ones installed by default and the machine-learning fallback skill are activated for now. Later on, you can customize this interaction (both ML model and skills) to fit your project.
2. Because we're not simply using Mycroft, but doing some development on it, we need to access something called a virtual environment. This is where all Mycroft's 'hidden bits' are contained. We will do this every time we're doing some work on Mycroft.
3. In mycroft-core folder, right-click to open terminal, then type source .venv/bin/activate. You should notice that in your terminal, next to where you put new lines, and extra .venv appeared. This indicates that you are currently operating in a virtual environment and can access otherwise inaccessible features of Mycroft. When you're done working with Mycroft, you can simply type deactivate in the terminal to exit the environment. Keep an eye on that extra line in your terminal. If something's not working or you get an error that a module is missing, double-check if you virtual environment is active and you see that extra .venv in your terminal
4. Then, to interact with your VA, type:
   ./runInteract.sh
   Note that by default, your interaction is recorded in whatVAHeard.txt, for your VA to later *evolve* from it (through machine learning). If you do not want this, open runInteract and set the value of the argument ifEvolve to False.
5. At the end of your first interaction, you can have a look at the file /workshop/data/whatVAHeard.txt, to see what the voice assistant recorded.

   Furthermore: In the script interact.py is also specified a parameter called keepThreshold. This indicates that above a certain character threshold, the answer of Mycroft is also recorded in the files whatVAHeard.txt (to train the ML algorithm). For instance, if Mycroft is talking about the weather, you may *not* want to keep track of this information. If on the other hand, Mycroft is reading out a Wikipedia article, you may want to record it. If you do

not want any speech of Mycroft to be recorded, you should set the character number parameter very high.

3. (Optional) Self-Quest

Hatching 'Self'

1. To 'hatch' a new seedling of a voice assistant, you first have to provide it with a baseline 'idea'. This should come in the form of text. Choose whatever type of writing you want based on the project you have in mind.
2. Replace the contents in /mycroft-core/workshop/data/hatchVA.txt with your text
3. Now right-click anywhere in the Linux window and click Open Terminal
4. Navigate to /mycroft-core/ folder. Now launch the runSelfQuest.sh by typing:
   ./workshop/scripts/runSelfQuest.sh --firstTime=True
   What this will do is activate a virtual environment which allows to access various hidden functions of Mycroft, as well as keeping the project contain; it will automatically launch Mycroft if it is not yet running; Finally, it will launch the required scripts to set up the required bits for Mycroft to start mapping his Self. Beware, this Quest may be long, as he has to compute all the semantic similarities.
5. All (optional) parameters that you can enable are:
   --firstTime(=True/False): Indicates if it is the first time you start a selfQuest.
   --walkNetwork(=True/False): whether at the end creation of the self graph, VA should 'walk on the network' speaking aloud a few concepts.
   --audibleSelfQuest(=True/False): Indicate if the SelfQuest should be audible. If yes, you can hear part of what Mycroft is doing (look for words on wikipedia etc.).
   --visualizeGraph(=True/False): Indicate if the self Graph should be visualised at the end.
   --ifMLDrift(=True/False): Indicates if in between the quests of concepts your VA shall do a machine-learning drift (Only make sense if the quest is audible).
   --lengthML(=number): Character length of the machine-learning drifts (if any)
   --nSimMax(=number): When the VA looks for a concept on Wikipedia, nSim indicates the number of words it will try to match with this new word, to see how similar they are.
   --nSearch(=number): the maximum number of words the VA should look for.
   --lengthWalk(=number): how many concepts are spoken out during the 'walk on network'
   --finetuned_ML_model(=True/False): if you want to use your fine tuned ML model or the default gpt-2. If you did a fine tuning, say True, else false.
   --path_finetuned_ML_model: path to where your fine tuned ML model is, if you trained it.

Self-Mapping

1. Later on, once you've interacted a bit with your VA, you can decide to launch a Self Quest via ./workshop/scripts/runSelfQuest.sh. Before launching this script, open it, modify the arguments according to your preferences (see step 5. In 'Hatching Self') above where they are detailed), and save it. Do not forget to set the firstTime value to False!

2. Once you are ready, go in your terminal, navigate to /mycroft-core and type ./workshop/scripts/runSelfQuest.sh Beware, this Quest may be long if the number of looked-up words is high.

4. Customisation

The above framework is just a starter. You can now feel free to tweak, tune, and add some skills or any other patches of code. Below, we detail a few options for customizations of different difficulty levels, along with a few links and references. Before going on further, some terminology:

utterance: a phrase spoken by the human, after human says the wake word (Hey.. “Mycroft”) and hears a corresponding beep.
intent: Mycroft matches utterances that you speak with a skill by determining an intent from the utterance. For instance, for ‘Hey Mycroft, what's the temperature like?’ the intent will be identified as *temperature* and matched with the *Weather Skill*. One skill can have several intents.
dialog: A dialog is a phrase that is spoken by Mycroft. Different Skills will have different dialogs, depending on what the Skill does.
FallBack skill: a Skill that is designated to be a backup dancer when Mycroft cannot understand what you say (a.k.a get the intent of an utterance)
handle_nameIntent: is the name of the function Mycroft will use in each skill to handle each intent.

1. Recording Time

The maximum timeout for a recorded message (RECORDING_TIMEOUT) is 10s by default. In case your project involves you *dictating* longer than 10, this can be changed in (in ResponsiveRecognizer Class):
/mycroft-core/mycroft/client/speech/mic.py
Also you can alter other parameters like RECORDING_TIMEOUT_WITH_SILENCE, etc.
Alternatively, you can add it (overwrite) in the config files:
/mycroft-core/mycroft/configuration/mycroft.conf
NB: Adding parameters to .conf allows tweaking them via the client interface.

2. ML-Fallback Skill

In the default workshop framework, the default fallback skill invokes a machine learning model to react to what you say if no other skill is triggered. The gpt-2 language model (see the guide in Limb folder) can be fine tuned on textual data. There are also a few other options in this skill we prepared that you can customize according to your project.

1. For this, navigate up to the folder home/opt/mycroft/skills/fallback-MLdrift/.
   It is not visible but you can go in your terminal and type
   cd
   xdg-open /opt/mycroft/skills/fallback-MLdrift
2. Open the file __init__.py.
3. You can alter different parameters of the ML drift defined in the preamble of the code: lengthDrift, number drifts, and so on. Some are classic parameters of a ML language model, others are omitted for simplification, but you are welcome to look into more details about gpt-2 and the transformers module.
4. Notably, you can modify and choose a few behavioral 'modes' (replace their name) according to your project. These will slightly influence the turn of the machine learning algorithm, as they will be given to the machine learning as context. For each of these modes, you have to come up with a few baseline sentences and overwrite modeSeeds. For each mode, choose also its probability (in probaMode).
5. As soon as you have fine-tuned and downloaded your own ML model, do not forget to change the parameter finetuned_ML_model=True. And check that the path pointing to your ML model is correct.

3. Mycroft Skills

No coding skills required

Some skills are installed by default. They are listed here along with their trigger commands. More official skills can be found in the skills marketplace. Pick the one(s) fitting your project. Here you can also find a corresponding Github repository. You can uninstall and install skills very easily, by voice or command line through the MSM, Mycroft Skill Manager once you launch mycroft.

1. Launch mycroft virtual env: $ source ./.venv/bin/activate
2. Add a skill: $ msm install skill-name
3. Remove a skill: $ msm remove skill-name

NOTE: We recommend uninstalling skills you do not want to use to avoid some unwanted triggers. For starters, we also advise to focus on fewer skills that could fit your project.

If you want to explore further, you can also have a peek at all the skills available on Github, including the ones that have not been officially tested/marketed. You can look at the ones in pending market and not in market, classified by themes. Of course you risk running into some issues, but you can try them out! You need to download them manually and put them in your folder home/opt/mycroft/skills/ , then restart mycroft services (./start-mycroft restart all)

4. Custom Triggers / Skill Hacks

No to little coding skills required

If you want to use an existing skill but modify how to trigger it, you can navigate to the specific skill's folder in /opt/mycroft/skills, go in the skill's /vocab subfolder, and modify the .voc or .intent files. *I.e. If I want that Mycroft laugh whenever I say 'human', I would go to the vocab file of Mycroft 'laugh' skill and add a line saying 'human' to the Laugh.intent*

The folder is not visible by default. In order to access it, open terminal, then do cd /opt/mycroft/skills, then type nautilus . (note the dot) and hit enter. This should open the folder in the graphic interface.

To better understand intent in more complex skills you may have to look at intent parser: the way Mycroft identifies and handles your intent. There are two possible parsers used by Mycroft which can be used in a Skill, among which Padatious is a bit easier to use. Adapt, a bit more flexible and widely used in the community. A tutorial about dealing with intents in Mycroft can be found here too.

You can also hack the skill further:

1. By modifying the .dialog files in the <skill-name>/dialog/ subfolder, you can change the way Mycroft will answer for some skills.
   I.e. If I want an answer "It's hot. ´Global warming, innit?" when I ask about the weather, I go to the current.hot.dialog and put the above line in. Actually, the weather skill has a whole bunch of other dialogs to check out.

2. If you change what the skill does by modifying the handle function(s) in _init_.py, you are changing how Mycroft reacts to a certain intent.
   I.e. If I want the Audio skill record several files, or that the remember skill spit something randomly from the VA memory, its a few lines of code ahead.

   More about the skill structure in Mycroft.

5. Create Your Skill

Some coding skills required, depending on skill complexity.

First, think in terms of "*If* Mycroft hears this …. then it should do/look-up/etc …. and reply ..."
The specific components to decide are:

1. What words will the Human need to speak to activate the Skill? >in /vocab/.
   NOTE: Except for a fallback Skill.
2. What data will the VA need to look for? > coded in handler function in _init_.py .
   Your Skill may need additional packages, dependencies, APIs, etc.
3. What will the VA say in response? > in /dialog/ .

In case you decide to venture in creating your own skill, you should have a look at the documentation and skill structure. Then, you can start with a template, or a simple 'hello world' skill (or with this to create a FallBack Skills). You can also look at some Github examples to get an idea of how skills are coded. Finally, there is a ready-made Mycroft tool

called mycroft-skills-kit (msk) to initialise skill templates. Here is an explanation. You can do it through the terminal directly, once you launch Mycroft.

NOTE:
1. Once created, the skill should be placed in /opt/mycroft/skills/MySKILL, *NOT* in /mycroft-core-mycroft/skills/MySKILL
2. The utterance string received from the speech-to-text engine is received in lowercase. For this reason, any string you are trying to issue should also be converted to lowercase. Compound words like "don't", "won't", "shouldn't" etc. are normalized by Mycroft - so they become "do not", "will not", "should not". You should use normalized words in your .voc files.

6. Custom Wake Word

This is possible with Mycroft but requires extra training. Alternatively, you can use a community-trained word such as “Hey, Computer!”

7. Additional References

Mycroft’s full documentation

Skill settings allow users to customize their experience or authenticate with external services. Learn how to create and use settings in your skill. Skill settings provide the ability for users to configure a Skill using a web-based interface

Conversation in skills