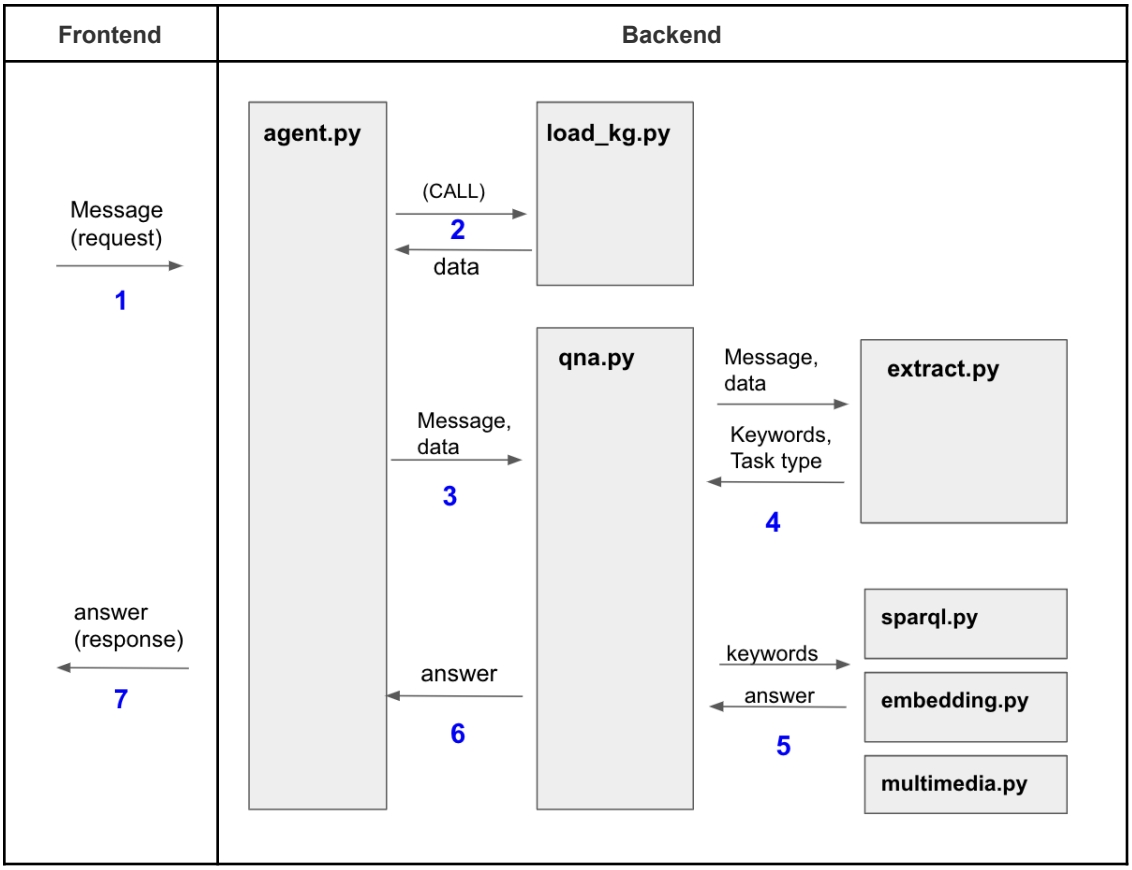
# 2022 HS ATAI Final Project Report
## Speakeasy - Building an Intelligent Conversational Agent

Songyi Han

### 1. Architecture

How to run my code : `$ python3 agent.py`

Once you run the code, agent.py will call load_kg.load() and load all the relevant data (e.g. knowledge graph, label to entity look up table, embeddings) and listen to the user message.
In this way, I could avoid loading graph each time and this made response time much faster. When user send a message, the agent will pass it to qna.answer(). What does qna.answer() is recognizing the type of question and then routing to the appropriate algorithms and get the answer consequently. To classify types of question, qna.answer() send the message to external function called extract.extract_question(). It will tokenize the message and classify the type of question and also extract the keywords (e.g. movie name, entity) and predicates (e.g relation). Finally, based on type of question, qna.answer() will send the keywords and predicate to either sparql.py, embedding.py or multimedia.py, get the proper answer accordingly.



## 2. Algorithm

### 2.1. Extracting question

In order to classify the type of questions, I had to use strong assumptions - for recommendation questions, I assumed the question would include “recommend” and for multimedia questions, the question includes “show me” or “look(s) like” explicitly. So the first step was to classify the type of question with these keywords then retrieve keywords and predicates with slightly different approaches.

Recommendation question string was splitted by either “similar to” or “like” then I only used the second half of the tokens to retrieve keywords. Since it is rare to have exactly the same keywords from a user, I post-processed these keywords by fuzzy matching with predefined labels listed in the graph.

Multimedia question was rather simple. I tokenized the whole sentence and posted it with spacy. Then only retrieved the token which is tagged as 'PROPN'. Again, the next step was fuzzy matching to make sure the keywords exist in the graph.

Most tedious task was extracting keywords and predicates from factual questions. I assumed there are two types of statements consisting of - 1) question words (who/when/what) + verb + keyword or 2) question words (who/when/what) + be-verb + predicate + of the + keyword. If the statement matches to type 1), I extracted the verb as a keyword, and if it matched to type2) I splitted sentence by “of” and retrieved the predicate from the first half of the sentence and keywords from the last half of the sentence.

### 2.2 Factual question and Embedding question

Factual question was answered by SPARQL running on the graph dataset. The query was simple triple `SELECT ?answer WHERE { <keyword> <predicate> ?answer}.` If the predicate and keyword was not able to match the existing label in the graph at the previous step, it will return a “not found” answer. Also even though there were matching keywords and predicates in the graph, but triple does not exist, the keywords and predicate were converted to similar labels using the embedding model and re-run the SPARQL query. If it does not work again, it will finally return the “not found” answer.

### 2.3. Multimedia question

There was a very useful data source called image.json which contains all the IMDB ID as a key and image resource as value. However, to use that, I had to get the IMDB ID of keywords in the knowledge graph with SPARQL `SELECT ?item WHERE { <keyword> <http://www.wikidata.org/prop/direct/P345> ?item . }` To decrease response time, instead of going over all json files, whenever there is a keyword(actor name) in the cast, I got the image resource and return the image immediately.

### 2.4. Recommendation question

Recommendation was fairly easy to answer using embeddings. The tricky part was how to design a more sophisticated calculation of similarity using all available information. For simplicity, I picked the one of the movies asked from the user and calculated the pairwise distance of the embeddings. Then I returned top 3 results except the one with distance zero which means identical.

**Appendix.**
Here is the dataset you need to test my code