# Building an Intelligent Conversational Agent

Noah Mamié[1]

**Abstract**
The challenge of creating an intelligent conversational agent is one of the current century's most daring enterprises in the world of AI. This report presents a possible approach for building an agent that is able hold a human-like conversation about the topic of movies. Note, since this is a mere overview of the agent's implementation details and not a formal paper, the report does not contain any formal references.

**Keywords**
Intelligent Agents — Artificial Intelligence — Large-Scale Knowledge Representation

[1] *Department of Computer Science, University of Zurich, Zurich, Switzerland*

## 1. Overview of the Agent

The agent consists of the following main classes, that are displayed in the form of a detailed class diagram in Figure 1:

**Movie Agent** This class is the ultimate implementation of the agent and really makes the agent ”come alive”. Its main task is the forwarding of pre-processed input sentences to the respective question type handler. The different question types are then built upon the pre-processing done by the agent (as elaborated in Section 2).

**Movie Bot** The movie bot serves as the connection of the movie agent to the Speakeasy interface. Additionally, this class enables the multi-threading capability of the agent and allows it to directly receive and store feedback from its users (more details in Section 4).

**Brain** As the name suggests, this class is the ”brain” of the agent and is responsible for the ”thinking”. Specifically, the main purpose of the brain class is the pre-processing of input sentences, including parsing, named entity recognition (NER), intent recognition, entity classification and pattern matching.

**SPARQL** This class is the main connection of the agent to its knowledge base (in the form of the knowledge graph (KG) that is loaded into memory in the KnowledgeGraph class).

**Response Formatter** The final formatting of the agent's responses is processed in this class, additionally providing several different options to answer each individual question type (to make the responses appear to be more ”human-like”).

## 2. Solving Different Question Types

This section introduces the implementation of the different question types in the agent:

**Factual** The simplest type of questions is handled in a very straightforward manner, by querying the KG with the recognized entities and intent.

**Embeddings** In case the answer is not contained in the KG, the respective embedding vectors of the entities (head) and the intent (relation) are computed and combined to derive the most likely answer to the user's question (tail).

**Crowdsourcing** The implementation is equivalent to the factual queries. The only difference is constituted by a check whether the crowdsourcing dataset was used to evaluate the correctness of this answer. Further, the user engagement of the crowd is presented here (per batch, no tasks skipped):

| Batch | Tasks answered | Throughput/min | Kappa |
|---|---|---|---|
| **7QT** | 21 | 0.41 | 0.24 |
| **8QT** | 20 | 0.32 | 0.04 |
| **9QT** | 20 | 0.32 | 0.20 |

**Table 1.** User Engagement Statistics.

**Recommender** The agent is able to provide the user with movie recommendations in two different ways. The first (and more sophisticated) option calculates a similarity matrix for each move given by the user. The features to derive this matrix are given by the movie plot, the genres and directors. The second option simply calculates the embedding vectors of the movies. In the end, both options combine the resulting matrices by simply computing the average.

**Multimedia** This task is implemented in the form of a dictionary based search, whereby the original 'images.json' file was first transformed into separate actor and movie dictionaries. These dictionaries contain, for each actor or movie, all appearances in an image that is present in the database.

## 3. Adopted Techniques

The main techniques that were adopted to create the movie agent are listed here. For the sake of brevity, this section does not include all techniques discussed during class, which were extremely helpful for implementing several major parts of the agent. Therefore, the following list is not exhaustive:

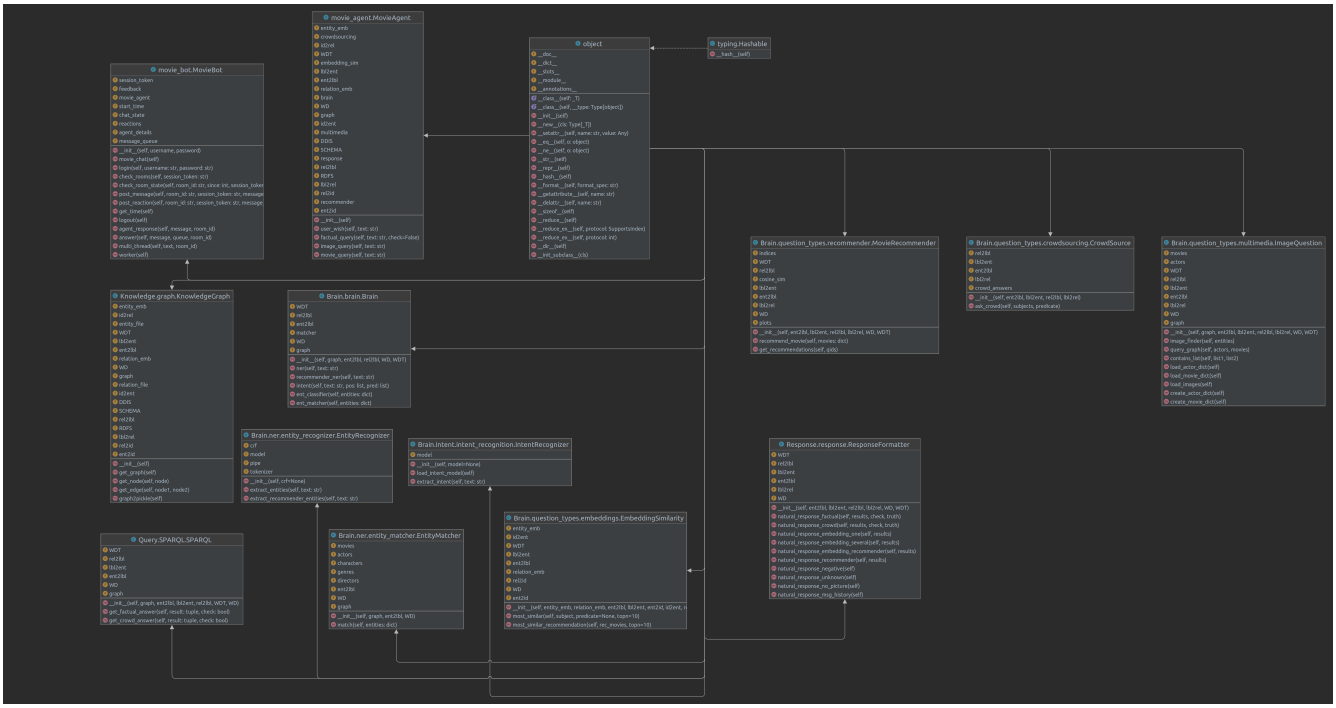


**Figure 1.** Agent Overview: Class Diagram

- NLTK for tokenization and POS tagging of sentences
- CRFs (cased and uncased) and BERT (uncased) for the agent's NER capabilities.
- Sentence-Transformers to create and compare embeddings of words and sentences.
- TF-IDF Vectorizer to convert raw data to features.
- Threading to enable thread-based parallelism when processing several tasks simultaneously.

## 4. Additional Insights

Lastly, the following points were relevant in shaping the current version of the movie agent's implementation.

In the early stages of this project, several experiments were conducted on the KG, in the hopes of generating additional insights and fixing potential errors. Interestingly, using PCA showed that the data contained in the KG contains most of its variance in its first two principal components (more than 80%), the 256-dimensional KG was reduced to a 2-dimensional representation (as displayed in Figure 2). Unfortunately, the recreation of the original KG proved to be unsuccessful and even the implementation of a deep autoencoder was not powerful enough to achieve this task. However, this process allowed the author a deeper understanding of the knowledge structure and, in the end, find and correct some errors in the KG. Furthermore, an additional question type was implemented, allowing users to check certain facts in the KG by reverse-engineering the factual question type. Also, when experimenting with CRFs and inspecting the weights more closely, it was observed that the model was overfitting the data from the MIT movie corpus, by simply learning individual names and terms. Therefore, the L1 and L2 regularization hyperparameters were optimized using cross-validation, leading to a much more generalizable result. Looking at the Speakeasy interface, the agent is able to listen to user reactions and, depending on the type, collect feedback from the users that is stored upon logout. Lastly, in case of emergency the agent has a built-in recovery state, allowing a successful reboot.

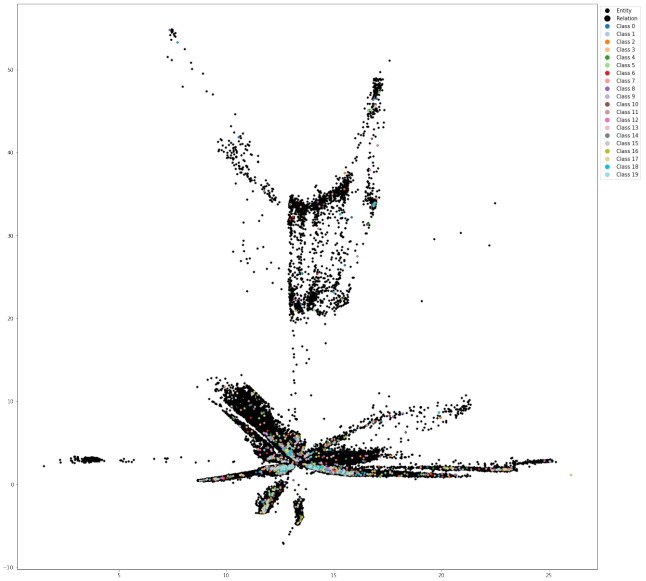

**Figure 2.** KG in a 2-dimensional representation