ATAI Final Evaluation Report

Haokun He, Haoyang Liang

August 19, 2024

Github Link

1 An overview of the agent

Our agent is an end2end chatbot which can receive both **SPARQL style or natural language query** from user and respond with suggested answer generated by corresponding solvers in the chatbot. Moreover, we implement several algorithms to leverage the graph dataset which is mainly about movie from wikipedia including pair-wise distance, rule-based mapping, name entity recogniton, content-based recommendation and so on. Here is our pipeline for the whole agent:

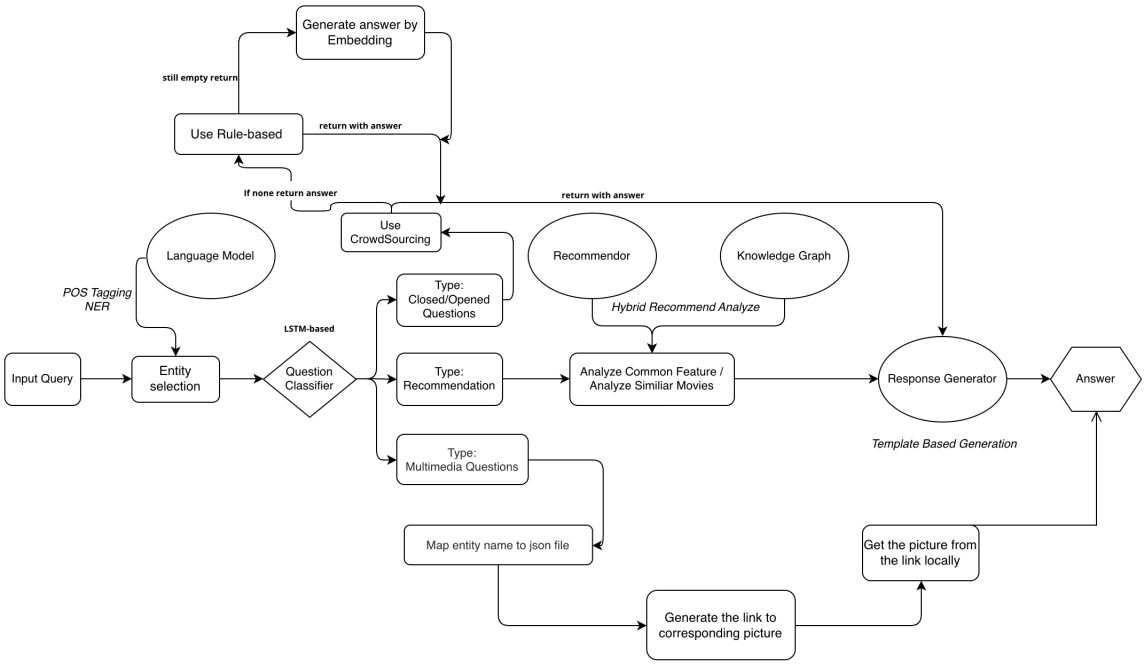

Figure 1: Pipeline

For each input query, it will be sent to NER module except pure SPARQL query. Then with the process of query intent classification, corresponding method will be called to generate suggested answer from this chatbot.

2 How questions are answered by the agent

2.1 General Module

Besides how to handle questions, another challenging aspect is classifying the question type. Here we trained a simple classifier model with the **LSTM** backbone to classifier the user input into thress types:

- 0: Querier Questions
- 1: Multimedia Question
- 2: Recommendation Question

Additionally, the training data is generated by GPT4 with the size of 3k sentences. Once the chatbot can tell the question type, it can consequently call corresponding module to generate corresponding responses.

2.2 Querier Module

In this project, we assembled crouwdsourcing, factual and embedding question answering as an entire module of "Querier", which is especially to handle knowledge QAs.

And further, all question queriers in **Querier** Module share the same process of entity and relation extraction. Firstly, the querier will try to parse the user input by language models. We utilized NER and POS here for sentence parsing. If there is still unknown information for either entity or relation, the chatbot will try to match each possible word combination with the entity set and relation set, which is so-called **rule-based**. Then if the rule-based method fails as well (usually means the given entity/relation is not in the dataset), the chatbot will finally try to find the closest entity/relation in the dataset for the extracted entity/relation with word embeddings calculated by the language model.

2.2.1 Crowdsourcing Questions

Crowdingsourcing querier performs as the most first level of this model. It is **placed beyond the rule-based and embedding methods**. In this case, when the chatbot acquires entity and relation names, it will first try to generate answers from the crowdsourcing dataset. If no answer returned or the agreement rate is low, then rule-based or embedding methods will be applied.

As for the concrete implementation for crowdsourcing, in the initial state, the dataset will be loaded into a "numpy.dataframe" with related columns.Then map the entity and relation names to index which can be used to select corresponding data from the dataframe. Lastly, count the votes, calculate required probabilities for inter-rater agreement respectively and then response to the user with suggested answer and those evaluation indexes.

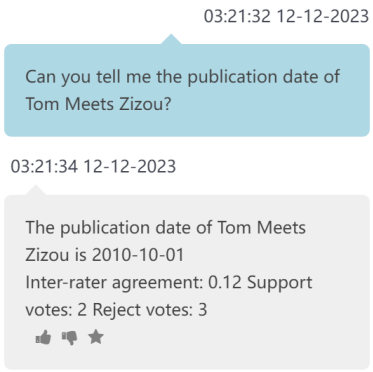

Figure 2: Example of Crowdsourcing Query

2.2.2 Factual Questions

Once crowdsourcing fails to find the answer for users, the Querier module will attempt to utilize the RDF knowledge graph to figure out the question.

Factual querier directly looks for the answer stored in the provided graph dataset. We create a **rule-based answer extracting function** for this question, the key is to select the movie entity name and relation name from the query correctly. In this case, we use pre-trained language model to achieve NER process then **fill the entity and relation nodes into a sparql query** which can directly return the required answer immediately. The output of sparql retrieval is independent words, so those will be sent to a respond generator which can **convert words into sentence** in natural language style. Also, to increase our chatbot's robustness, a synonym mapping dictionary is created which can **handle different representations of user input for the same relationship entity** and map them to the corresponding relation names.

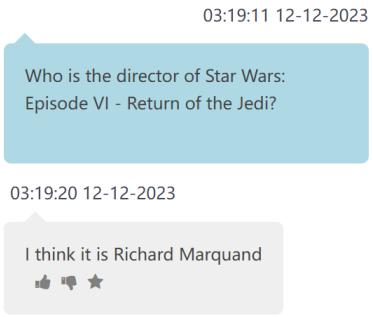

Figure 3: Example of Factual Query

2.2.3 Embedding questions

The embedding querier usually acts as the final solution if both the crowdsourcing querier and the factual querier fail. Given the extracted entity and the extracted relation, the embedding querier will look up their pre-trained graph embedding and calculate an expected object embedding with **TransE** formula, which is given by :

$$Emb_{\{tail\}} = Emb_{\{head\}} + Emb_{\{predicate\}}$$

Then the closest existing entity/relation in our dataset will be considered as the query result.

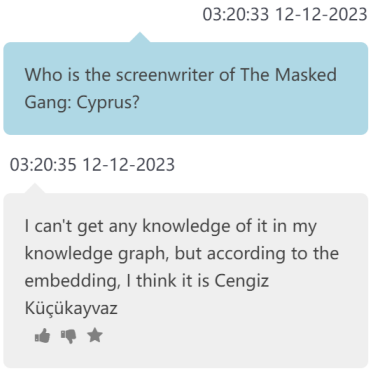

Figure 4: Example of Embedding Query

2.3 Multimedia

2.3.1 Multimedia questions

The multimedia module is mainly aiming to understand the user input and return a corresponding image from the dataset *MovieNet*. *MovieNet* provides a dictionary with substantially three essential information:

1. **IMDB ids** for involved movie(s);
2. **IMDB ids** for involved cast(s);
3. a unique image id

By combining the first two items, we can obtain a list of involving entities for each image.

Later, all entities will be extracted with a language model for NER. And extracted entities will be passed to the knowledge graph for querying corresponding **IMDB ids**. Once the **IMDB ids** list is determined, we can compare the given list with **IMDB ids** lists of each image and determine the optimal image with the following metric:

$$\text{image index} = \arg max_i\{\#\text{overlap entity}_i - \alpha \times length_i\{\text{IMDB ids}\}\}$$

where α is a penalty coefficient to prevent the module from always returning the image with more entities.

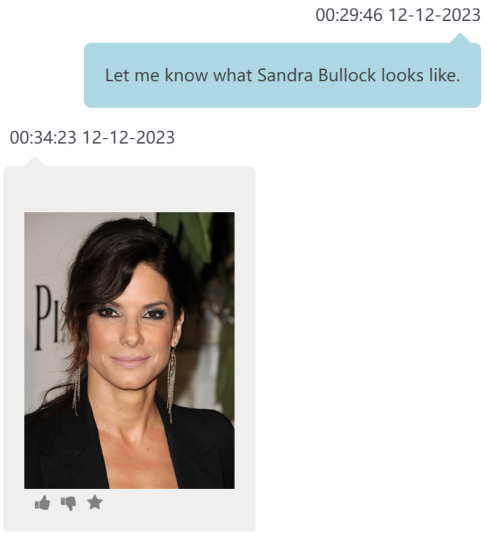

Figure 5: Example of Multimedia Query

2.4 Recommendation

Similar to other types of questions, several language models will be utilized at first for paring the sentence and extracting entities involved.

Given involved entities, the "Recommendor" module is capable of providing two methods of recommendation, which are *1. Collaborative Filter (CF)* and *2. Content-Based (CB)*. The CF method supposes that movies have multiple attributes, and users will love the movie with similar features to the provided movies. Therefore it grabbed features of the provided movies and clustered them to extract and recommend several common features. While the CB methods are based on the movie graph embedding. Since embeddings can represent the topology structure and content to some extent, we utilized it to calculate a similarity matrix of movies. And directly recommend movies from the top K-th closest movies.

Subsequently, the bot now can organize the returned recommends from two methods as an integral response for the user.

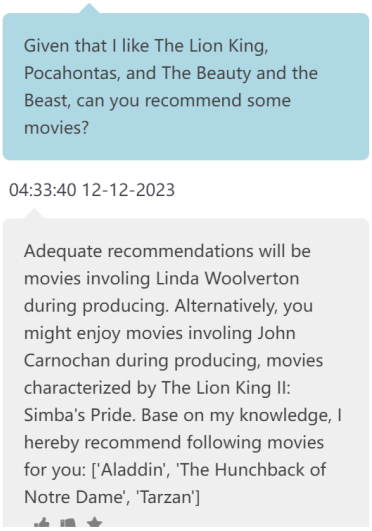

Figure 6: Example of Recommendation Query

3 List of adopted techniques

- **Sparknlp**: pre-trained language models for the NER task and the POS tagging task. Utilized JAVA and Spark for higher efficiency in both memory management and computing acceleration.
- **KMeans (sklearn)**: unsupervised learning derived by *sklearn*. used to find centroids from the spatial pattern of given features in the graph embedding space.
- **Concrete Movie RecSys based on Pairwise distance**: compute the similarity between input and all stored movie embeddings by Pairwise distance.
- **LSTM model for question classifier**: Generate a dataset of queries with intents by GPT and train a LSTM-based intent classification model that can classify input queries into factual questions, multimedia questions, and recommendation questions.
- **Fasttext**: pretrained model for getting word embedding, more powerful in handling **OOV** words