Final Evaluation Event - *fierce-blade*

1st Adam Bauer (*adam.bauer*) 2nd Mete Polat (*mete.polat*)
3rd Matthias Mylaeus (*matthias.mylaeus*)*

December 9, 2024

1 Introduction

Our current agent can answer recommendation, multimedia and factual questions supported by crowd sourcing dataset. Additionally, our chat bot is able to engage in small talk with the user, such as responding to messages like "hi", "thank you", and similar phrases. Finally, if the user asks about an entity for which the label is the same for several entries in our graph, the bot engages in an interactive conversation to determine which entity exactly the user is asking about. Our factual questions are limited to those that include one relation and one movie entity. If the answer for knowledge questions is not found in the graph, our agent uses embeddings to compute a response. Our agent can also recommend movies on the base of movies the user provides. Multimedia questions recognize a person or a movie in the question and retrieve an appropriate image from MovieNet [2] .

If the answer cannot be found or computed, or if the entity or relation is not recognized, the agent politely asks the user to rephrase the question or to ask something else.

2 Capabilities

Our agent provides the following services: Agent Answering Service, Extraction Services, Image Finder, Recommendations, Question Classifier, Disambiguation and SPARQL Graph.

Firstly, the incoming question is classified into one of the following types: Multimedia, Knowledge, Smalltalk, or Recommendation. Afterwards, the question is processed through the appropriate answering service. Each answering service primarily performs an extraction of movie names, people names, and relations from the incoming message. Furthermore, each service processes the extracted data accordingly.

In the following subsections, each individual step is explained in more detail.

*Joined 13.11.2024

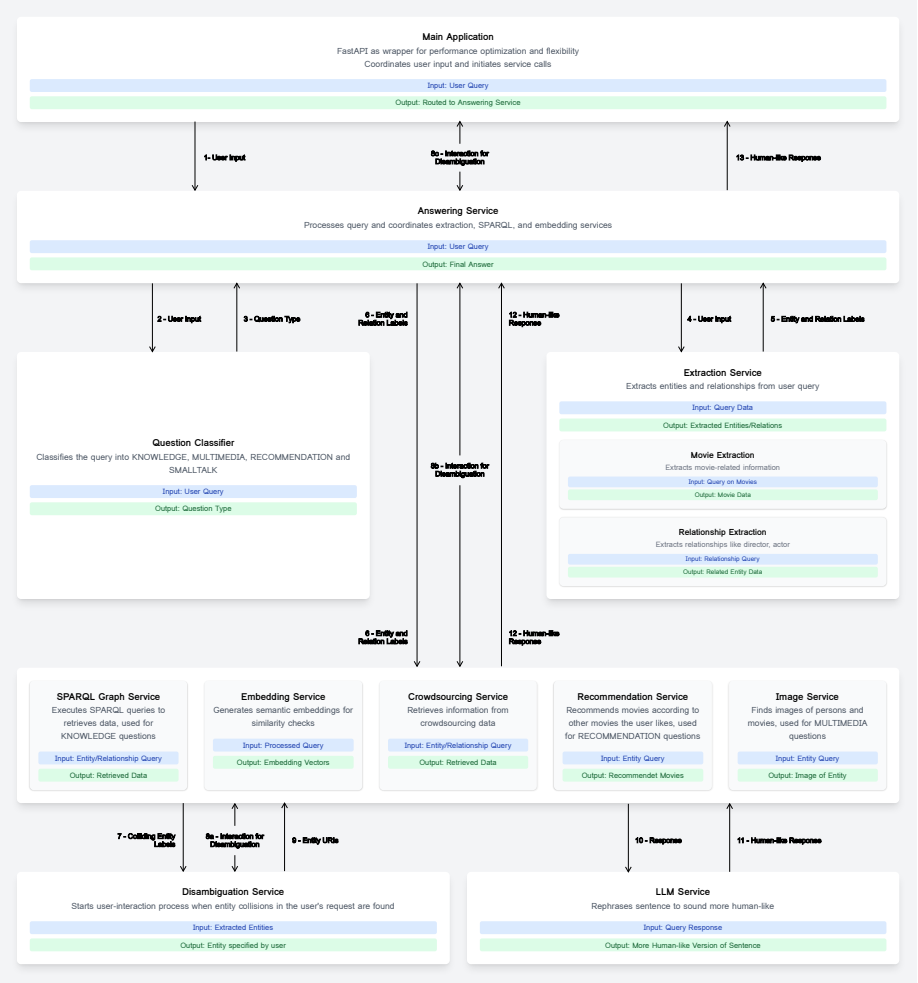

Figure 1: Diagram showing the input and output of each service.

2.1 Question Classifier

Our question classifier is based on the Sentence Transformer library. We created 50 examples per category and embedded them into the latent space. Initially, we experimented with linear regression, but it did not perform well for classifying similar queries and movie names were misleading sometimes as well. As a solution, we trained a custom single neural network (NN) to classify incoming sentences using embeddings from Sentence Transformers and the custom NN classifier. This approach ensures high accuracy in question classification across the categories: Multimedia, Knowledge, Smalltalk, and Recommendation.

2.2 Extraction of Entities and Movies

Common Approach - NLTK + Fuzzy Search

Our approach combines rule-based methods and fuzzy search for robustness:

- **NLTK-based Approach:** A lowercase preprocessing step is applied, and a pretrained NLTK rule-based model is used to identify entities, such as movie titles, persons, and relations, from the knowledge graph.
- **Fuzzy Search Integration:** Fuzzy search enhances the rule-based method by correcting for user misspellings or minor variations in entity names. For example, in the query "Who directed Godzilla vs Kong," the fuzzy search ensures the correct match with the knowledge graph entity "Godzilla vs. Kong," even though punctuation is omitted in the query.
- **Custom Matching Algorithm:** Challenges, such as handling shared terms in movie names (e.g., "Return" in "Star Wars: Return of the Jedi"), are addressed using a custom algorithm. This algorithm prioritizes the longest and most specific match, ensuring accurate identification of movie entities.

This approach is effective for extracting all types of entities, including movies, persons, and relations. For relations, a refined mechanism is employed to handle varying phrasing.

Relation Identification: Relations, unlike entities, may be expressed in various ways (e.g., "director," "directed," or "who directs"). For this reason, relations are identified using a Sentence Transformer-based approach:

- All relations from the knowledge graph associated with a given movie entity are embedded into the latent space using sentence transformers.
- The query sentence is masked (removing the entity) and embedded into the same latent space.
- The relation closest to the given relationship is selected based on cosine similarity, ensuring an accurate match. An threshold is applied to ensure that if there is no reasonable relation, no relation is determined and user is informed.

2.3 Disambiguation of Entity Labels

After successful NLP, the entities are fetched. If the request includes labels for which there are several entries, the user is engaged in an interactive conversation to specify the wanted entity. Namely, for each entity with the same label, the description is fetched from the knowledge graph and then returned as a list of possible candidates. The user is given the option to choose which entity is in fact desired. If this is the case for multiple entities in the prompt, this process is repeated. Help is given, in case the choice given by the user is not understood. The user also has the option to interrupt disambiguation at any point. After the ambiguities are solved, the prompt is executed normally. In addition, the user is informed about the choice they have made to further clarify which entity is used. This process works similarly for movies such as people. The following example illustrates the program flow:

1st Prompt: Who is the director of Titanic?

Response:
It seems there are several entries for Titanic:

1. 1953 film by Jean Negulesco
2. 1943 German propaganda film
3. 1997 American romantic disaster film directed by James Cameron
4. 1996 TV miniseries directed by Robert Lieberman

Let me know which one you're asking about or whether you'd like to stop.

2nd Prompt: The 3rd one. (This response is processed to enable the user linguistic freedom in choosing the desired item.)

Response: I understand you're asking about the 1997 American romantic disaster film directed by James Cameron. As far as I know, the director of Titanic is James Cameron.

2.4 Factual Questions

Factual questions are processed once the question is classified as a knowledge question, and the entities and relations are successfully extracted. The SPARQL service is then queried to retrieve the answer from the knowledge graph. The key steps include:

- Identifying entities (e.g., movies, persons) and relations using the approaches outlined above.
- Formulating a SPARQL query based on the extracted entities and relations.
- Returning the retrieved answer to the user.

The assumption is that the knowledge graph contains only valid answers. Therefore, results retrieved from the graph are not further validated.

2.5 Embedding-based Questions

If the answer cannot be retrieved from the knowledge graph, an embedding-based approach is used as a fallback mechanism. This method leverages embeddings derived from entities and relations to infer the most plausible answer.

- **Entity and Relation Identification:** The entity vector and relation vector are extracted from the query using the embeddings generated by Sentence Transformers.
- **Vector Combination with TransE Scoring:** These vectors are combined using the TransE scoring function to represent the query in the latent space of the knowledge graph embeddings.
- **Distance Calculation and Selection:** The combined query vector is compared against the latent space of possible entities. The distance is calculated, and the closest match is selected as the most plausible entity.
- **Answer Embedding and Return:** The identified entity is embedded into a meaningful response sentence, which is then returned to the user.

Limitations: Embedding-based approaches are less accurate than knowledge graph queries and are therefore used only as a backup. The knowledge graph is assumed to contain valid answers, so embeddings are not used to validate graph-based results. If no plausible answer can be inferred, the agent requests the user to rephrase the question or to ask a different one.

2.6 Multimedia Questions

When a question is classified as a multimedia question, a relationship extraction process is initiated to identify any persons mentioned in the query. This process follows the standard approach used in our application, which combines an NLTK rule-based method with enhancements through fuzzy search for improved accuracy.

Once the person is identified, their entity is retrieved from the Knowledge Graph using their corresponding IMDB ID. This ID is then used to query a custom preprocessed dataset. The dataset is structured as a mapping in the format `imdb_id:(array of all pictures where the person is alone)`.

Finally, the appropriate image is retrieved from the dataset and returned as a string in the format supported by the platform, `image:image_identifier`, accompanied by an additional description for context.

As a refinement, we also extracted all posters of the movies. In case a person is not found in the query, we fall back on a movie search and then return its poster. We decided to show only posters for movies, as we consider them to be a proper form of identification.

2.7 Recommendations

When a question is classified as a recommendation request, it is forwarded to the recommendation service. The service begins by querying the knowledge graph for all possible attributes associated with the movies mentioned in the question. Among these attributes, priority is given to *genre* (P136), *director* (P57), and *part of a series* (P179), as these are considered the most significant for generating relevant recommendations.

The system then verifies whether the *part of a series* or *genre* and *director* attributes are shared among all the user-provided movies. If a match is found, the knowledge graph is queried to identify movies with these matching attributes.

If no matches are found for the primary attributes, the system broadens the scope to search for any other shared attributes. If at least five secondary attributes match, a query is performed using those attributes.

In cases where no attributes match, the system defaults to recommending a movie from the same *genre* as the first movie mentioned by the user. This fallback mechanism ensures that a recommendation is always provided, even when attribute information is limited.

In all the methods described above, the recommended movies are sorted by *review score* (P444). The top three movies with the highest ratings that meet the requirements are selected. Additionally, the system provides an explanation in the recommendation message, highlighting the attributes that matched across the movies. Finally, the system searches for an image of the recommended movies, selecting the first matching image to accompany the recommendation and explanation.

2.8 Crowdsourcing Questions

Filtering Out Malicious Workers:

- **Loading Dataset:** The initial dataset contains **305 records**.
- **Removing Workers with Low Lifetime Approval Rates:** Workers with a lifetime approval rate below 50% are excluded. *Result:* The dataset is reduced to **244 records**.
- **Removing Workers with Homogeneous Contributions:** Workers who contributed only with identical labels are removed. For instance, Worker `2133U7HKDLO` contributed 21 times with label: *CORRECT* but was already excluded in the previous step due to a low lifetime approval rate. *Result:* No further changes to the dataset; it remains at **244 records**.
- **Removing Workers with Unrealistically Fast Completion Times:** Workers with a *WorkTimeInSeconds* of less than 10 seconds are excluded, as it is assumed they randomly clicked answers to earn rewards. *Result:* The dataset is reduced to **183 records**.

Fleiss' Kappa for Batches:
The agreement among workers (Fleiss' kappa) for each batch is summarized below:

Batch	Kappa
7QT	0.236364
8QT	0.040000
9QT	0.199110

Implementing into current architecture: To ensure the data is accurately reflected when answering questions through the Knowledge Graph, we enhance searches that match the filtered crowdsourced data by incorporating information from its associated batch, including rejections and approvals.

3 Adopted Methods

- **NLTK**
  We used the NLTK library[1] to implement a rule-based approach for identifying labels, specifically for movies and relations. Additionally, NLTK is used to remove stop words from input sentences, helping to improve the accuracy of relation extraction by reducing noise.

- **SentenceTransformer Library**
  The SentenceTransformer library[2] is used to embed relations into a vector space, allowing us to compare the input sentence with known relations. We specifically utilize the 'all-mpnet-base-v2' model[3] from Hugging Face, which is well-suited for semantic similarity tasks. The input sentence is preprocessed by removing stop words to ensure the embeddings focus on key terms, enhancing the relevance of similarity comparisons. We leverage SentenceTransfomer also for embedding of the initial question and as an default embedding for an NN classifier.

- **RapidFuzz and Levenshtein Distance**
  We leverage the RapidFuzz library[4] for fuzzy string matching, which allows us to match movie names even when they are not exact matches. RapidFuzz relies on the Levenshtein distance algorithm[5] to calculate the similarity between strings. For improved accuracy, we set a similarity threshold of 90%, ensuring that only closely matching terms are identified as movie names, even if they contain minor variations.

[1] `https://www.nltk.org/`
[2] `https://sbert.net/`
[3] `https://huggingface.co/sentence-transformers/all-mpnet-base-v2`
[4] `https://github.com/rapidfuzz/RapidFuzz`
[5] `https://en.wikipedia.org/wiki/Levenshtein_distance`

4 Examples

4.1 Factual Questions

Given a question such as, "*Who is the director of Good Will Hunting?*" our agent follows a structured process to retrieve the answer:

First, the agent classifies the incoming query "Knowledge" the movie entity in this case, *Good Will Hunting*—using the NLTK library for movie extraction. Once the movie entity is detected, it is masked in the question to focus on identifying relationships. In this example, the relationship *director* is extracted as the key relation.

Next, these extracted labels are mapped to their corresponding entity and predicate in the knowledge graph. Once the mapping is complete, the agent constructs and executes a query in the knowledge graph, returning the result as the answer.

4.2 Embedded Questions

If the question cannot be answered using the graph, the answer is calculated using embeddings. Since the graph is assumed to contain only correct information, embeddings are used solely as a backup. Therefore, no cross-checking is performed, and the answer with the highest score is selected and returned.

4.3 Multimedia Questions

Given a question such as "What does Denzel Washington look like?", the question is classified as a Multimedia question. The system then proceeds to identify the correct entity, in this case, "Denzel Washington," using the Person Image Finder. The entity is queried in the Knowledge Graph to retrieve the corresponding IMDB ID.

Using this IMDB ID, a custom preprocessed dictionary is accessed to locate all images associated with the person. From this collection, one image is randomly selected and returned to the user. The final output includes the image formatted as a string, supported by the platform, and optionally accompanied by additional descriptive information.

4.4 Crowdsourcing questions

Given the question, "Who is the executive producer of X-Men: First Class?," we first query the data to determine whether it is present in the crowdsourcing dataset. For questions of this nature, similar to knowledge-based queries, we avoid using embeddings for the answers since verification is not feasible. Instead, we incorporate relevant information from the crowdsourcing dataset, including the calculated inter-rater agreement and the distribution of support and rejection votes (e.g., inter-rater agreement of 0.199, with 2 support votes and 1 reject vote). This approach ensures that the response is grounded in crowd consensus data.

4.5 Recommendation Questions

Given a question such as "Given that I like *Avengers: Endgame* and *Thor: Ragnarok*, can you recommend some movies?" it is first classified as a recommendation request and forwarded to the recommendation service. The service queries the knowledge graph to retrieve relevant attributes for the provided movies. For *Avengers: Endgame*, attributes such as genre (Action, Adventure, Science Fiction), director (Anthony Russo and Joe Russo), and part of a series (Marvel Cinematic Universe) are retrieved. Similarly, for *Thor: Ragnarok*, attributes like genre (Action, Adventure, Fantasy), director (Taika Waititi), and part of a series (Marvel Cinematic Universe) are identified.

Next, the system prioritizes the attributes genre, director, and part of a series for filtering. Upon analyzing the data, it finds that the attribute part of a series (Marvel Cinematic Universe) matches for both movies. Using this shared attribute, the system queries the knowledge graph to identify other movies within the Marvel Cinematic Universe. Among the retrieved results, Doctor Strange appears as the first relevant match. Consequently, the system recommends Doctor Strange as the answer to the user's request. Furthermore, the system selects a random image from the Doctor Strange movie and also explains how the recommendation was made by listing all matching attributes.

5 Additional Features

To enhance the timeliness of our agent, we preload all necessary models and libraries upon startup on the server. For development purposes, we implemented a lazy-loading approach, allowing for a faster application startup during testing and debugging. Additionally, we pre-exported all movie labels and relationships to avoid redundant processing, loading them into memory as predefined data for quicker access. We also preprocessed lookup array for image finder to ensure much faster search of images given a person.

The *humanness* of our agent is achieved through a response rendering stage, where we define distinct response templates. The templates are used to inform the user when a prompt was received, and about what choices the user has made. Furthermore, we use templates to dynamically fill with the answer to produce responses that feel more natural and conversational. Additionally, we add manual checks for *Welcome* and *Thanks* messages, to which we respond without performing query. Furthermore, we began utilizing the open-source *llama3.2* to rephrase our sentences, aiming to make the answers sound more natural and human-like. To ensure that the language model consistently behaves as intended and refrains from adding any additional information, we experimented with various system prompts.

One challenge we addressed involved the varying wording of relationships and the use of synonyms. For example, a user might ask, "When was a movie released?" whereas the corresponding relationship in the knowledge graph is labeled as *publication date*. To bridge this gap, we initially used embeddings to

match synonyms more accurately. However, as this approach did not yield sufficient precision, we implemented a manual mapping specifically for *publication date* to correctly align it with *release date* in the user's query.

Another challenge we encountered involved hyphens and their variants. To improve search accuracy, whenever the incoming query contains a hyphen, we automatically replace it with different variants—such as em and en dashes—to improve matching accuracy. This solution has significantly reduced potential user errors related to hyphenation, making the user experience more resilient to variations in the dataset.

In the beginning, when pre-exported all movie labels, movies with the same label were simply overwritten. Therefore, when users ask for a movie for which there are several entries all with the same label, we only ever had a single option. To overcome this issue we adjusted our pre-exporting process and saved all entries more precisely. Entries with the same label were aggregated and are now used to start a user-interaction process to identify which entity the user is requesting.

We also experimented with BERT [1] for Named Entity Recognition (NER) to enhance entity extraction accuracy. Although this approach required additional training on our dataset, it still relied on matching with fixed labels in the knowledge graph—primarily through fuzzy search. Given these dependencies, we ultimately opted for a pure fuzzy search approach combined with rule-based matching using NLTK, which provided more consistent results with less complexity.

6 Conclusions

Throughout this project, we developed a sophisticated chatbot system capable of addressing a variety of user queries, including factual questions, multimedia requests, recommendations, and small talk. By leveraging cutting-edge tools and methodologies, such as the SentenceTransformer library, NLTK, and fuzzy search, we ensured robust entity and relation extraction for high-accuracy responses.

A significant aspect of our work was the implementation of a dynamic question classifier, which categorized queries into appropriate types for streamlined processing. Additionally, we designed a recommendation system that intelligently prioritizes attributes to generate relevant suggestions. For multimedia queries, our custom pipeline effectively identified and retrieved high-quality images associated with movie characters or posters, enhancing the user experience.

To improve the chatbot's conversational quality, we integrated the open-source *llama3.2* model to rephrase sentences, making responses more natural and human-like. Challenges such as synonym handling, hyphenation variations, and the alignment of user phrasing with knowledge graph labels were addressed through innovative solutions, including manual mappings and rule-based enhancements.

Overall, this project exemplifies the integration of advanced natural language

processing techniques with practical engineering solutions. The result is a user-centric chatbot system that demonstrates accuracy, adaptability, and a focus on delivering high-quality interactions on a specific domain.

7 Contributions

For the project, the contributions are divided as follows:

- **Adam Bauer:**
  - Implemented and trained the Question Classifier.
  - Developed the complete Multimedia Service.
  - Developed NER for entity and relation recognition.
  - Cleaned and preprocessed crowd sourcing data.
- **Mete Polat:**
  - Implemented the Factual Answers.
  - Implemented the Embedding Answers.
  - Designed and implemented the Recommendation System.
  - Integrated the Large Language Model (LLM) into the Answering and Small Talk Service.
  - Addressed the encoding issue.
  - Architecture Diagram Base Template.
- **Matthias Mylaeus:**
  - Implemented the Crowd Answers.
  - Designed and implemented the Disambiguation System.
    - Generated mappings to include all URIs which share the same label. This mapping is used after extraction to find possible collisions of the user's request.
    - Implemented a user-interaction process to engage the users in specifying the correct entity. This process saves the state for every chat room separately to enable users to inteact easily with the bot and enables a more human conversation. Thanks to the saved state, rather than only having the flow request and response, users now can interact with the bot concerning a previous request.
  - Addressed the encoding issue.
  - Implemented user-friendlier responses to inform the users about what's going on (i.e., when a request is received, what option was picked for disambiguation).
  - Architecture Diagram Design and Structure.

References

[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[2] Q. Huang, Y. Xiong, A. Rao, J. Wang, and D. Lin. Movienet: A holistic dataset for movie understanding. *CoRR*, abs/2007.10937, 2020.