Evaluation Event III - *ancient-flame*

1st Lu Li (*lu.li*) 2nd Jiawei Pei (*jiawei.pei*)

December 9, 2024

1 Introduction

The current capabilities of our chat agent, Ancient-Flame, focus on delivering efficient and accurate responses to a range of queries regarding the Wiki film information. Ancient-Flame is equipped to answer factual questions that require precise and contextually relevant information. Additionally, the agent is capable of handling embedding-based queries, leveraging vector representations to understand and respond to more complex and nuanced questions. Lastly, the agent is able to answer recommend questions based on user's preferences by knowledge graph and vector similarities to recommend similar movies with shared attributes. These kinds of questions are augmented with crowd sourcing information.

2 Capabilities

Figure 1 illustrates the core components of the Ancient-Flame Agent and the flow of inputs and outputs. The agent processes natural language user queries, identifies the question type, and routes it to the appropriate processing module to generate a response.

The Ancient-Flame Agent supports two primary question types: factual or embedding-based questions, and recommend questions. When a user query is received, the *Intent recognizer* analyzes user's intent.

For factual or embedding questions, the *Query Processor* analyzes the input to extract the entity and relation in the query. Factual questions are directed to the *Factual Processor*, where construct a triple tuple to query retrieve structured data from a knowledge graph, allowing the agent to respond accurately to fact-based inquiries. For more context-dependent queries, the *Embedding Processor* leverages embeddings to capture semantic relationships, enabling the agent to answer questions requiring an understanding of similarity or nuance.

For recommend question, *Query Processor* extract the entities in user query, then the *Recommender* calculates the central vectors based on entities' embedding then finds the top 3 nearest movie entities, and discovers the shared attributes of the entities.

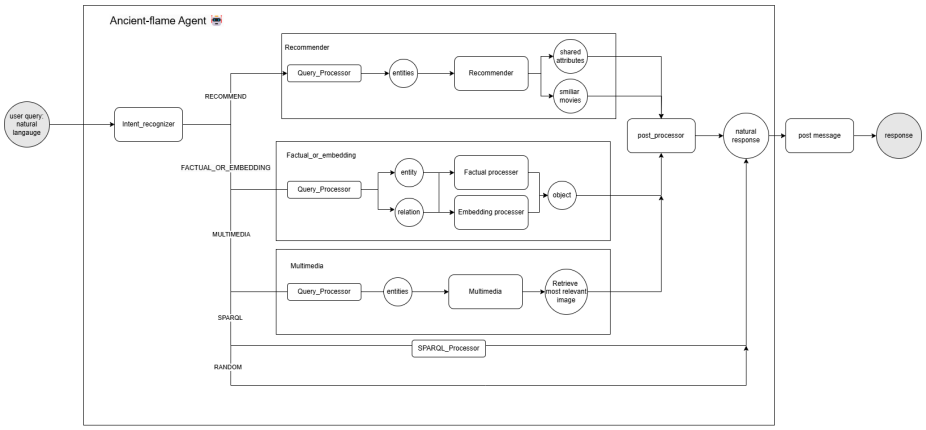


Figure 1: Chatbot Pipeline

For crowd sourcing question, *Query Processor* extract the entities in user query, then matching the first entity with *Crowd Sourcing* augmented data to get the Fleiss Kappa score, support votes and reject votes.

Following processing, the *Post Processor* refines the output into a coherent response, which is then delivered to the user by the *Post Message* component. This modular design ensures that Ancient-Flame can efficiently handle diverse queries with low latency, balancing accuracy with responsiveness.

- **Factual Questions**
  Our agent answers factual questions in three steps: 1) extract entity and relation through Spacy, BERT, and direct matching, 2) using fuzzywuzzy postprocess the extracted entity, mapping the extracted entity to knowledge graph entities and 3) construct a tuple(entity, relation, object) to query the knowledge graph.

  In the first step, we employ a fine-tuned BERT [1], specifically the bert-large-cased-finetuned-conll03-english model, which is optimized for English named entity recognition tasks, to extract entities from the user query. We made the following changes to adapt BERT to the task:
  1. The BERT model processes at the token level, meaning the extracted entities are in subword form. To address this, we concatenate the extracted tokens into complete words.
  2. Some of the concatenated words may be incomplete due to missing characters in the extracted tokens. To improve accuracy, we prepared an entity file containing all entity URIs and labels from the knowledge graph. We then use Fuzzywuzzy to match the extracted words with entities in the knowledge graph, setting a similarity threshold of 88

- **Embedding Questions**
  Our agent answers embedding questions in three steps:

1) Extract entity and relation using the same method with processing factual questions
2) Match the embeddings with the extracted entity and relation
3) Calculate the pariewise distance between query entity and relation embeddings with the knowledge graph entity embeddings to get the most close entity.

- **Recommend Questions**
  Our agent answers recommend questions:
  1) Extract multi-entities using the Bert and fuzzy match, match the embeddings.
  2) Calculate the central/average vector of the entities embeddings.
  3) Find the Top 3 nearest movie entities(except the given movies) based on the distance of the average vector.
  4) Lookup the shared attributes of these entities in the knowledge graph using SPARQL, generate common features of user's preferences.

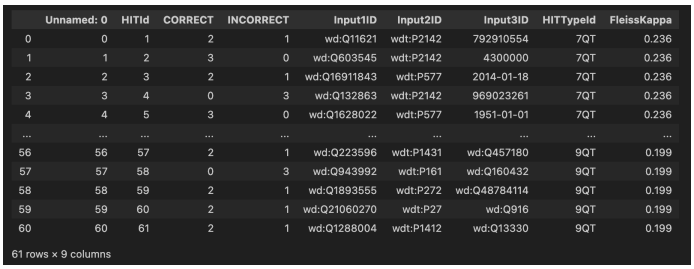
- **Multimedia Questions**
  Our agent answers Multimedia questions:
  1) Extract multi-entities using the Bert and fuzzy match, match the embeddings.
  2) Resolve entities to IDs using SPARQL queries and map them to metadata fields like movies and casts.
  3) Calculate image relevance scores by measuring the overlap between query-relevant IDs and image-associated IDs with a penalty for larger image associations.
  4) Retrieve the most relevant image by identifying the highest-scoring image or returning a default response if no matches are found.

- **Crowd Questions**
  Our agent answers crowd questions:
  1) Extract the first entity in the query.
  2) Match the entity with prepared crowd source data, as shown in 2 by subject id.
  3) Look up the corresponding Fleiss Kappa score, support votes and reject votes which have been calculated in advance.

3 Adopted Methods

The following methods, tools, and techniques were utilized in developing the Ancient-Flame Agent, with appropriate reasoning provided for each choice:

- **BERT** [1]
  We employed BERT for named entity recognition (NER) due to its superior performance in understanding contextual information and identifying



	Unnamed: 0	HITId	CORRECT	INCORRECT	Input1ID	Input2ID	Input3ID	HITTypeId	FleissKappa
0	0	1	2	1	wd:Q11621	wdt:P2142	792910554	7QT	0.236
1	1	2	3	0	wd:Q603545	wdt:P2142	4300000	7QT	0.236
2	2	3	2	1	wd:Q16911843	wdt:P577	2014-01-18	7QT	0.236
3	3	4	0	3	wd:Q132863	wdt:P2142	969023261	7QT	0.236
4	4	5	3	0	wd:Q1628022	wdt:P577	1951-01-01	7QT	0.236
...	...	...	...	...	...	...	...	...	...
56	56	57	2	1	wd:Q223596	wdt:P1431	wd:Q457180	9QT	0.199
57	57	58	0	3	wd:Q943992	wdt:P161	wd:Q160432	9QT	0.199
58	58	59	2	1	wd:Q1893555	wdt:P272	wd:Q48784114	9QT	0.199
59	59	60	2	1	wd:Q21060270	wdt:P27	wd:Q916	9QT	0.199
60	60	61	2	1	wd:Q1288004	wdt:P1412	wd:Q13330	9QT	0.199

61 rows × 9 columns

Figure 2: Processed Crowd Dataset

entities in natural language queries. A fine-tuned BERT model (bert-large-cased-finetuned-conll03-english) was selected to optimize NER tasks specifically for English language queries, which significantly enhances the accuracy of entity extraction.

- **Fuzzywuzzy**[1]
  Fuzzywuzzy was used to perform fuzzy matching for entity recognition, allowing the agent to handle variations and minor inaccuracies in user input when mapping entities to the knowledge graph. This approach enables more robust entity resolution, especially when exact matches are not available, improving response precision.

- **Relation Matching Mechanism**
  A custom matching mechanism was developed for relation extraction. Using a pre-defined file containing relation IDs and labels from the knowledge graph, we implemented a mapping process to link relations in the query with those in the knowledge graph, ensuring efficient and accurate relation extraction.

- **Sklearn Pairwise Distances**[2]
  To determine similarity in embedding-based questions, we used sklearn's pairwise distance calculation to measure the semantic closeness between query embeddings and knowledge graph embeddings. This approach enhances the agent's capability to handle complex, context-sensitive queries.

- **Statsmodels**[3]
  Using the *fleiss kappa* function in *statsmodels.stats.inter rater* to calculate the Fleiss Kappa score in each batch to evaluate the crowd evaluator's reliability.

[1] `https://github.com/seatgeek/fuzzywuzzy`

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise_distances.html`

[3] `https://www.statsmodels.org/dev/generated/statsmodels.stats.inter_rater.fleiss_kappa.html`

- **OpenAI API**[4]
  The OpenAI API was used to generate natural language responses based on retrieved answers, adding fluency and coherence to the final output, specifically it does not have the role to modify the answer, we use few-shots learning prompt engineering method to fine-tune its response. This step ensures that responses are not only informative but also conversationally appropriate while keeping with the retrieved answer from our pipeline.

4 Examples

Embedding Questions: As shown in Figure 3, given the question, "*Who is the director of Good Will Hunting?*" our agent first extracts the entity "Good Will Hunting" and the relation "director." It then retrieves their embeddings from the embedding file and uses the TransE model to combine the entity and relation embeddings. Pairwise distance is then applied to calculate the similarity between the combined embedding and the embeddings of potential entities, ultimately identifying the most likely entity as the answer. As a result, our agent returns the result "Harmony Korine" but the correct director is "Gus Van Sant", the reason is that Harmony Korine also contributed to the movie he wrote the draft script and had many cooperations with Gus Van Sant, so their embeddings are close to each other which causes this result.

Recommend Question: As shown in Figure 4, for the query *Given that I like The Lion King, Pocahontas, and The Beauty and the Beast, can you recommend some movies?*, our agent recognizes the question type is *RECOMMEND* firstly, then it extracts multi-entities within the query "The Lion King", "Pocahontas", and "The Beauty and the Beast", matching the entities with embeddings then calculates the average vector, based on the average vector, our agent recommends the three nearest movies which are "'Aladdin", "The Hunchback of Notre Dame", "The Little Mermaid", and extract their shared attribute such as "animated movies", "real-life remakes of Disney movies".

5 Additional Features

Challenges:

Extracting the correct entities required significant effort. Initially, we used Spacy, a traditional and popular NER toolkit. While it performed adequately on common terms like "director" and "screenwriter," it struggled with rare entities or relation terms, such as "MPAA film rating" or "Star Wars: Episode VI - Return of the Jedi." We then pivoted to a pre-trained BERT model, shown in figure 6 and figure 7. However, BERT processes text at the subword level, which created challenges; for example, as shown in figure 5 it segmented the entity "Good Neighbors" as "Good Nei" and "bors," resulting in the loss of

[4] `https://platform.openai.com/`

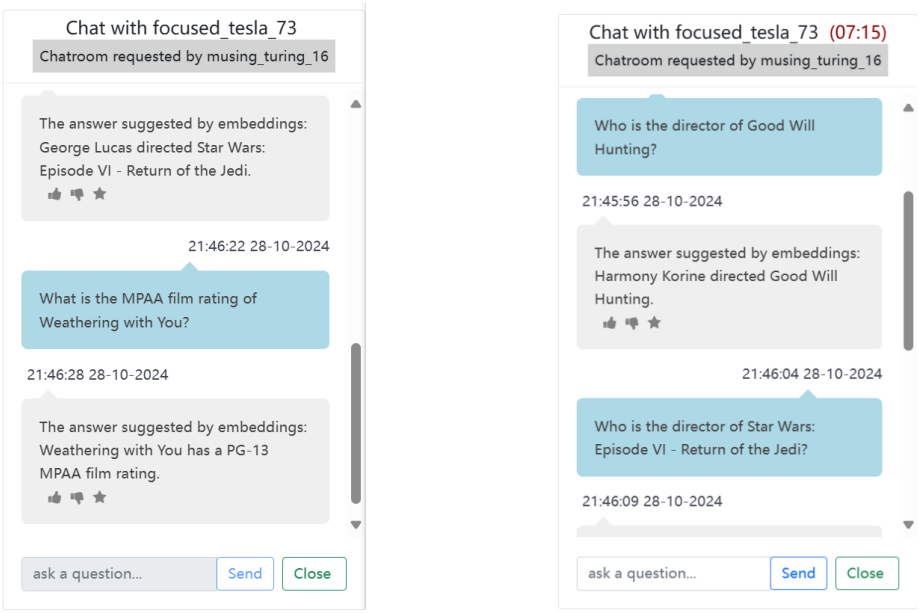


Figure 3: Example Q&A of Embedding Questions

characters like "gh." To address this, we introduced a fuzzy matching mechanism. Using Fuzzywuzzy in figure 8 we matched the entities extracted by BERT against a pre-prepared entity file based on the knowledge graph. This approach ensured both the accuracy of entity recognition and alignment with entities in the knowledge graph, helping us handle null values effectively.

Achievements:

To improve the efficiency of entity fuzzy matching, we upgraded our toolkit, reducing matching latency from 2.7 seconds to 0.5 seconds. Additionally, to make the agent's responses more "human-like," we integrated the OpenAI API to generate more natural responses for users.

6 Conclusions

To conclude, our agent is capable of answering factual and embedding-based questions, recommend questions, crowd questions. We also integrate more advanced LLMs, such as the OpenAI API, into our response, leverage the robust capabilities of LLMs to make the interaction more user-friendly and natural.

References

[1] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein,

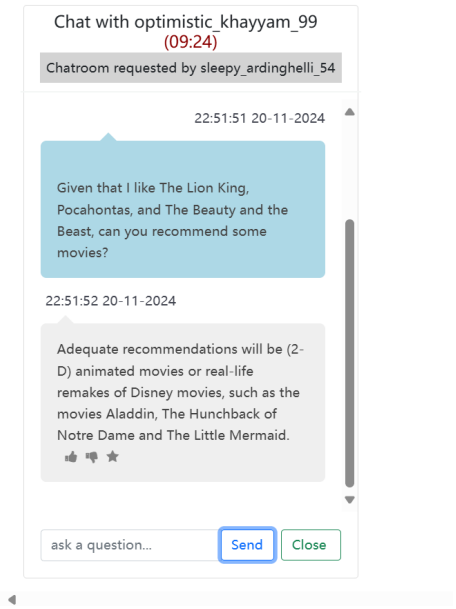
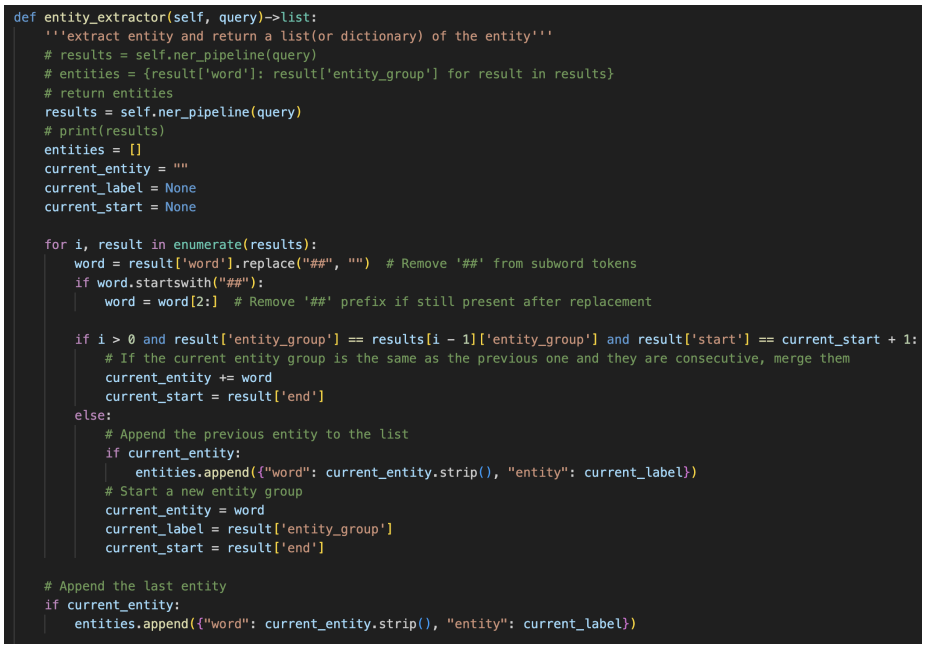


Figure 4: Example Q&A of Recommend Question

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.

```
query:What is the genre of Good Neighbors?
entity:{'MISC': 'Good Nei bors'}
ralation:{'relation': 'genre'}'
```

Figure 5: Missing characters issue



```python
def entity_extractor(self, query)->list:
    '''extract entity and return a list(or dictionary) of the entity'''
    # results = self.ner_pipeline(query)
    # entities = {result['word']: result['entity_group'] for result in results}
    # return entities
    results = self.ner_pipeline(query)
    # print(results)
    entities = []
    current_entity = ""
    current_label = None
    current_start = None

    for i, result in enumerate(results):
        word = result['word'].replace("##", "")  # Remove '##' from subword tokens
        if word.startswith("##"):
            word = word[2:]  # Remove '##' prefix if still present after replacement

        if i > 0 and result['entity_group'] == results[i - 1]['entity_group'] and result['start'] == current_start + 1:
            # If the current entity group is the same as the previous one and they are consecutive, merge them
            current_entity += word
            current_start = result['end']
        else:
            # Append the previous entity to the list
            if current_entity:
                entities.append({"word": current_entity.strip(), "entity": current_label})
            # Start a new entity group
            current_entity = word
            current_label = result['entity_group']
            current_start = result['end']

    # Append the last entity
    if current_entity:
        entities.append({"word": current_entity.strip(), "entity": current_label})
```

Figure 6: Entity Extraction Process-1

```python
    # Convert list of entities to dictionary format for return
    merged_entities = {}
    for entity in entities:
        if entity['entity'] in merged_entities:
            merged_entities[entity['entity']] += " " + entity['word']
        else:
            merged_entities[entity['entity']] = entity['word']

    # 匹配现有的entity
    if merged_entities:
        results = []

        for entity_value in merged_entities.values():
            matching_results = self._entity2id(str(entity_value))
            if matching_results:
                results.append(matching_results)

    return results
```

Figure 7: Entity Extraction Process-2

```python
def _entity2id(self, entity: str)-> dict:
    threshold = 88

    matching_results = {}

    best_match = process.extractOne(entity, self.data_entities.values(), scorer=fuzz.ratio, score_cutoff=threshold)

    if best_match:
        matched_title, best_score = best_match[0], best_match[1]
        entity_uri = next((uri for uri, title in self.data_entities.items() if title == matched_title), None)
        entity_id = entity_uri.split('/')[-1:][0]
        if entity_uri:
            matching_results[entity_id] = matched_title

    return matching_results
```

Figure 8: Fuzzy matching