CS 598 COURSE ASSIGNMENT

Miri Liu
NetID: miri3
miri3@illinois.edu
GitHub link

1 INTRODUCTION

The KG-RAG (Knowledge Graph-based Retrieval Augmented Generation) framework (Soman et al., 2023) applies RAG techniques to a biomedical knowledge graph. The framework is evaluated on multiple-choice biomedical questions. Each question asks which entity (e.g., gene, disease, or variant) from a given candidate list is associated with a specific set of biomedical conditions or properties, and given some knowledge graph context, the LLM must select the correct answer from the candidate list. While this kind of task clearly shows the value of additional context, since general-purpose LLMs do not have this kind of information in their parametric knowledge, it is also important that researchers carefully consider *what* should be included in that context, as well as how it should be *structured*. This report documents my exploration of how to address these questions through the implementation of four different modes.

2 METHOD

We are asked to build on the provided baseline of KG-RAG performance with three specified strategies.

First, we structure the retrieved context, which is currently passed as raw sentences to the LLM. We do so by parsing the sentences into a **JSON context**, which should be easier for the LLM to handle (Mode 1). Second, we provide **external domain knowledge** that the LLM may not have or may need a reminder for about how to interpret that context (Mode 2). Third, we **combine the two strategies** and observe how multiple improvement strategies can be used in tandem (Mode 3).

Additionally, I improve the quality of the retrieved context by adding an LLM prompt filter *over* the JSON context, which only keeps context that is possibly relevant to candidates from the list (Mode 4). The motivation here is that the JSON context of Mode 1 is actually still fairly noisy, meaning many snippets are not strongly relevant to the question, which may actually confuse the LLM.

3 IMPLEMENTATION

In my implementation, I changed only the files `run_mcq_qa.py` and `utility.py`. I produced five CSV files as model outputs, which are named `gemini_2.0_flash_kg_rag_based_mcq_{mode}.csv`, where `mode={0,1,2,3,4}`. All five are stored in the `data/my_results/` folder. In addition to these CSVs, in the same folder, I store `gemini_2.0_flash_kg_rag_based_mcq_4_contexts.csv`, which is a file comparing contexts with and without Mode 4.

For Mode 0, I did not write any code, but rather directly followed the instructions given on the GitHub to produce the baseline CSV. New files, along with all original files, can be found in my public GitHub repository (also linked at top of page).

The only change that is *not* reflected in this repository is that I also added my Google API key to `gpt_config.env`.

3.1 MODE 1: JSON

To implement this strategy, in `utility.py`, I created a duplicate function of `retrieve_context()` called `retrieve_context_jsonize()`. Instead of concatenating sentences, it groups them by disease and encodes their relationships in a structured dictionary. The helper function `json_disease()`, also in `utility.py`, handles this transformation: given the high-similarity sentences for a node, it uses regex matches to identify gene associations, variant mentions, disease–disease links, disease ontology identifiers, and any leftover miscellaneous relations, and places them into a consistent JSON schema. My first implementation of this mode used a similar structure to the one in the assignment document (where we have, say, `"Gene":"SLC29A3"`), but I found that this does not significantly improve performance. Instead, I opted for a flatter JSON schema, where we group all gene associations, variant associations, etc in one list (such as `"genes":["a","b","c"]`).

The only change made to Mode 0 evaluation is that we retrieve context with `retrieve_context_jsonize()` instead of the original function.

3.2 MODE 2: EXTERNAL DOMAIN KNOWLEDGE

To implement this strategy, I inspected the kinds of associations that came up in the context based on my work in Mode 1. For each stored association, I asked an LLM (Claude Sonnet 4.5) to write a sentence addressing the utility of that kind of context (i.e., "Ontology IDs are identifiers only" for the disease ontology ID association) and compiled these into one constant string stored in the `run_mcq_qa.py` file as `prior_domain_knowledge`. The string is as follows:

> Related diseases often share gene associations. Variants follow the disease patterns of their genes. Ontology IDs are identifiers only. Disease–disease relations imply shared biology. Symptoms and provenance do not affect gene associations. Always return exactly one answer. Use context evidence first; only when every candidate has zero context evidence should you rely on prior biological knowledge to choose the most plausible candidate.

For Mode 2 evaluation, instead of only giving the question and context, I also gave this prior domain knowledge string. I tested various orderings of the three and settled on `question + prior_domain_knowledge + context`.

3.3 MODE 3: COMBINED MODES 1 AND 2

To implement this strategy, I applied my implementations from Modes 1 and 2. We still use the `question + prior_domain_knowledge + context` ordering for evaluation (as in Mode 2), but the context part of it is now supplied by `retrieve_context_jsonize()` rather than `retrieve_context()`.

3.4 MODE 4: COMBINED MODES 1 AND 2 + FILTER

To implement this strategy, I introduce a second LLM prompt whose sole job is to filter the raw JSON into a smaller, more focused subset. I initially considered using a regex-based filter, similar to what I did in Mode 1, to keep only entries that explicitly mention one of the candidate answers. However, this is possibly too brittle. For example, the context might say "Disease X is associated with gene Y," and even though gene Y isn't in the candidate list, an LLM could infer that gene Y is closely related to gene Z (something the prior domain knowledge from Mode 2 strongly encourages). A strict regex filter would remove these useful indirect associations, but since we rely on the same LLM to make these inferences at evaluation time, it is reasonable to let it make them during filtering too.

The prompt takes the JSON context and the candidate list, which we find through a oneliner regex match from the question text at runtime.

Mode	Strategy	Correct Answer Rate	Improvement over Baseline
Mode 0	Baseline	73.53%	—
Mode 1	JSON Context	81.37%	+7.84%
Mode 2	Prior Domain Knowledge	85.29%	+11.76%
Mode 3	Combination	86.93%	+13.40%
Mode 4	Combination + Filter	87.91%	+14.38%

Table 1: Performance comparison across modes.

4 Experimental Results

As the table above shows, the baseline Mode 0 has an already fairly high correct answer rate of 73.53%. It is clear that the model is already doing substantially better than random guessing, especially since the LLM sometimes refuses to guess at all.

Mode 1, JSON context, improves on the baseline by 7.84%, achieving a correct answer rate of 81.37%. Mode 2, prior domain knowledge, improves on the baseline by 11.76%, achieving a correct answer rate of 85.29%. Finally, Mode 3, a combination of Modes 1 and 2, improves on the baseline by 12.42%, achieving a correct answer rate of 85.95%.

I theorize that this relatively small Mode 3 improvement (over Modes 1 and 2) is because the "easy" cases to improve performance on are already completely covered by either Mode 1 or Mode 2.

Finally Mode 4, which adds an LLM filter on the JSON context on top of the other improvements, improves on the baseline by 14.38%, achieving a correct answer rate of 87.91%. While the improvement over Mode 3 is, like Mode 3's improvement over Mode 3, comparatively minor, this kind of mode could also bring value when the downstream task requires interpretable or compact evidence. In this assignment we evaluate on multiple-choice questions, but in a generation setting, where the model must justify, summarize, or otherwise show its reasoning, a smaller, more intentionally curated context could be extremely beneficial.

On the following page, I show an example of why this is the case. (It is on the next page because the minipage package otherwise distorts the spacing.)

For example, here is how Mode 4 transforms the context for the question ”Out of the given list, which Gene is associated with psoriasis and Takayasu’s arteritis. Given list is: SHTN1, HLA-B, SLC14A2, BTBD9, DTNB”:

Original JSON Context


```
{
  "Diseases": {
    "psoriasis 13": {
      "genes": [
        "TRAF3IP2."
      ]
    },
    "psoriasis 1": {
      "genes": [
        "HLA-C."
      ]
    },
    "psoriasis 4": {
      "Disease Ontology ID": "0111280"
    },
    "psoriasis": {
      "genes": [
        "HLA-B",
        "HLA-DQB1"
      ],
      "variants": [
        "rs13203895 x rs4349859",
        "rs11652075",
        "rs41298997",
        ... 28 more variants
      ]
    },
    "psoriasis 15": {
      "genes": [
        "AP1S3."
      ]
    }
  }
}
```


Filtered JSON Context


```
{
  "Diseases": {
    "psoriasis": {
      "genes": [
        "HLA-B",
        "HLA-DQB1"
      ]
    }
  }
}
```


The second is clearly more tractable, not to mention easier for humans to read, should they want to audit the LLM’s use of the context and understand how the final answer was derived.

REFERENCES

Karthik Soman, Peter W Rose, John H Morris, Rabia E Akbas, Brett Smith, Braian Peetoom, Catalina Villouta-Reyes, Gabriel Cerono, Yongmei Shi, Angela Rizk-Jackson, et al. Biomedical knowledge graph-enhanced prompt generation for large language models. *arXiv preprint arXiv:2311.17330*, 2023.