CS 598 JH (Fall 2025) — KG-RAG with Knowledge Graphs

Author: Matthew Handzel NetID: handzel4 Date: 2025-10-29

Introduction This report documents my reproduction and enhancement of the provided KG-RAG pipeline for multiple-choice biomedical question answering. I reproduced the baseline with Gemini-2.0-flash on the MCQ dataset at data/benchmark_data/mcq_questions.csv, then implemented and evaluated three strategies exposed by the assignment harness in kg_rag/rag_based_generation/GPT/run_mcq_qa.py (Modes 0–3). In addition, I designed and implemented a new bonus method (Mode 4) that filters the retrieved context to only statements explicitly involving one of the gene options from the question, reducing prompt length and concentrating evidence. I executed Modes 0–4 end-to-end and evaluated each run with the provided script; all outputs are saved to data/my_results/.

Method & Implementation Baseline (Mode 0) follows the original flow in run_mcq_qa.py. For each MCQ row, it builds a free-text context by concatenating the highest similarity statements returned by retr ieve_context(...) in kg_rag/utility.py20. It then constructs a prompt as "Context: " + context + "\n" + "Question: " + question and calls get_Gemini_response(...) with the MCQ_QUESTION system prompt. The script appends (question, correct_answer, llm_answer) to a list and continuously writes a CSV to dat a/my_results/ so partial progress is not lost.

Strategy 1 (Mode 1) replaces the free-text context with a structured JSON context by calling retrieve_co ntext_json(...) in kg_rag/utility.py. That function emits a compact JSON payload that records matched disease nodes and high-similarity statements with explicit fields for source, predicate, target, and provenanc e. By providing a structured context, the LLM sees clean triples and provenance instead of a long paragraph, which improves answer extraction and reduces ambiguity. The rest of the prompting is unchanged: the same MCQ_QUESTION system instruction is used, and the model returns { "answer": <option> }.

Strategy 2 (Mode 2) injects prior domain knowledge into the prompt body while keeping the baseline retrieval format. In run_mcq_qa.py, when --mode 2 is selected and --prior_knowledge or --prior_knowledge_pa th is provided, the script constructs the prompt as a three-part message that appends an "IMPORTANT NOTES" section between the retrieved context and the question. Concretely, it becomes Context + "\nIMP ORTANT NOTES:\n" + prior_knowledge + "\nQuestion: " + question. This design preserves the original retrieval and pruning while allowing targeted heuristics to steer tie-breakers. The call to Gemini and output format remain identical to Mode 0.

Strategy 3 (Mode 3) combines Strategy 1 and Strategy 2: the retrieved context is generated via retrieve_co ntext_json(...) and the same "IMPORTANT NOTES" block is injected before the question. This produces a compact, machine-readable context with an adjacent domain hint that can help disambiguate near-ties. The system prompt and output format again remain unchanged. I validated that the CLI accepts --mode 3 and reads prior knowledge either inline or from prior_knowledge.txt at the repository root.

Experimental Setup I ran all commands in a fresh environment created with Python 3.10.9 after pip instal l -r requirements.txt and python -m kg_rag.run_setup. I executed Mode 0 via sh run_gemini.sh (equivalent to python -m kg_rag.rag_based_generation.GPT.run_mcq_qa gemini-2.0-flash --mode 0) and Modes 1–4 via python -m kg_rag.rag_based_generation.GPT.run_mcq_qa gemini-2.0-flash --mode <1|2|3|4> [--prio r_knowledge_path prior_knowledge.txt]. Each run produces a CSV in data/my_results/ named as gemi ni_2.0_flash_kg_rag_based_mcq_<MODE>_<timestamp>.csv. I evaluated runs using python data/m y_results/evaluate_gemini.py --file_path <CSV>, which normalizes the model's JSON and computes the correct answer rate by matching the answer field to correct_answer.

Experimental Results Here are the experimental results

Mode	Description	Correct Answer Rate	File Location
0	Baseline (free-text context)	74.51%	data/my_results/gemini_2.0_flash_kg_rag_based_mcq_0.csv
1	Structured JSON context	72.88%	data/my_results/gemini_2.0_flash_kg_rag_based_mcq_1_1761777676

Mode	Description	Correct Answer Rate	File Location
2	Prior knowledge injection	76.80%	data/my_results/gemini_2.0_flash_kg_rag_based_mcq_2_1761782379
3	JSON context + prior knowledge	*77.12%*	data/my_results/gemini_2.0_flash_kg_rag_based_mcq_3_1761786043
4	Targeted option-aware filtering	**84.64%**	data/my_results/gemini_2.0_flash_kg_rag_based_mcq_4_1761788620

As you can see, adding prior knowledge injection (Modes 2 and 3) slightly improves over the baseline (Mode 0), while switching to structured JSON context alone (Mode 1) slightly degrades performance. The best of these is Mode 3, which combines JSON context with prior knowledge, achieving 77.12%. However, my bonus method (Mode 4) significantly outperforms all others, reaching 84.64% by filtering context to only statements mentioning option genes.

Bonus Method (Mode 4): Targeted option-aware context filtering For the bonus, I implemented Mode 4, which filters the retrieved context to only those statements that explicitly mention a gene from the question's options. Practically, after context pruning, I remove any statement that does not include one of the candidate gene strings. This reduces the prompt size and focuses the model exclusively on edges that directly connect the diseases in the question to one of the allowed answers. I ran Mode 4 across the full dataset and achieved 84.64% accuracy, significantly outperforming Modes 0–3. The evaluation was produced with python data/my_results/evaluate_gemini.py --file_path data/my_results/gemini_2.0_flash_kg_rag_based_mcq_4_1761788620403292088.csv.

Case study: reducing overload to expose the correct answer. Consider the question: "Out of the given list, which Gene is associated with psoriasis and myelodysplastic syndrome. Given list is: NOD2, CHEK2, HLA-B, GCKR, PKNOX2". For this item, the other methods give the model way too much context. A representative baseline snippet looks like the following, which is dominated by variant associations and many genes unrelated to the options:

```
Variant rs2853952 x rs13202464 associates Disease psoriasis.
Disease psoriasis associates Gene HLA-DQB1.
Disease psoriasis associates Gene HLA-B.
Disease psoriasis associates Gene GJB2.
Variant rs13203895 x rs4349859 associates Disease psoriasis.
Variant rs12191877 x rs6075938 associates Disease psoriasis.
Variant chr1: 152591953 associates Disease psoriasis.
Variant rs12191877 x rs13202464 associates Disease psoriasis.
Disease psoriasis associates Gene HLA-DRB1.
Disease psoriasis associates Gene DNAJB4.
Variant rs9262498 x rs12191877 associates Disease psoriasis.
Variant rs12660883 x rs2524052 associates Disease psoriasis.
Variant chr6: 159506600 associates Disease psoriasis.
Variant chr10: 75601596 associates Disease psoriasis.
Variant rs75851973 x rs13203895 associates Disease psoriasis.
Variant rs9262492 x rs12191877 associates Disease psoriasis.
Variant rs2844624 x rs2523619 associates Disease psoriasis.
Variant rs6911408 x rs2853953 associates Disease psoriasis.
Variant rs2853952 x rs9468932 associates Disease psoriasis.
Variant chr5: 96118852 associates Disease psoriasis.
Variant rs4358666 x rs6916062 associates Disease psoriasis.
Variant rs6911408 x rs2523619 associates Disease psoriasis.
Variant rs17728338 x rs13203895 associates Disease psoriasis.
Variant rs13191519 x rs4349859 associates Disease psoriasis.
```

```
Variant rs9366778 x rs4959062 associates Disease psoriasis.
...
```

In Mode 4, after filtering to only statements that include one of the gene options, the retained context becomes simply:

```
['Disease psoriasis associates Gene HLA-B']
```

With this targeted context, the model can directly pick HLA-B from the option set without distraction. This example illustrates how option-aware filtering surfaces decisive evidence while discarding irrelevant edges and variant chatter.

Case study: corrigibility under no-evidence conditions. Mode 4 is intentionally strict. On examples where the retrieval process fails to retrieve any statement that contains one of the gene options, the filtered context becomes empty. In these cases the model refrains from guessing and returns an explicit inability-to-answer message, for example:

```
"Out of the given list, which Variant is associated with autoimmune disease and anti-neutrophil
antibody associated vasculitis. Given list is: rs2310752,  rs7528604, rs6679677,  rs34691223,  rs296322
  ""answer"": ""Based on the context provided, I cannot definitively
  determine which variant is associated with autoimmune disease and
  anti-neutrophil antibody-associated vasculitis.""
```

This behavior is preferable to hallucination in safety-critical settings; it reflects corrigibility in the face of insufficient evidence. Importantly, errors here stem from retrieval gaps rather than over-filtering, because when evidence is present for an option gene, the method retains it.

Analysis of Mode 4. Compared to Mode 0, which concatenates high-similarity statements regardless of whether they mention an option gene, Mode 4 enforces an option-aligned relevance constraint. This sharply reduces prompt length and removes distractors, improving answer extraction accuracy. Relative to Mode 1, which restructures evidence as JSON but can still include many non-option statements, Mode 4 pairs relevance with brevity. Relative to Mode 2 and Mode 3, which can bias the model with prior notes, Mode 4 succeeds without domain hints by ensuring that only option-bearing evidence is visible. The method does not achieve a perfect score; the remaining errors largely occur when retrieval fails to surface any statement naming an option gene. In such instances, Mode 4 appropriately declines to answer rather than guessing, trading recall for precision by design. Overall, the empirical gains and qualitative behavior suggest option-aware filtering is an effective and robust enhancement for this MCQ task.

Reproducibility Notes Ensure GOOGLE_API_KEY is present in gpt_config.env so kg_rag/utility.py can initialize the Gemini client. Keep LLM_TEMPERATURE=0 in config.yaml for stable answers during evaluation. If a run is interrupted, run_mcq_qa.py writes the CSV incrementally under data/my_results/, so the existing partial file can be re-evaluated or extended by re-running the same mode.