Julius Caesar RAG Retrieval System using Groq API

Scene-level Retrieval-Augmented Generation over Shakespeare's *Julius Caesar*

Patenge Kanishka (MT2024111)
Gnanendhar Reddy (MT2024084)

AID 849: Advances in Natural Language Processing
Dr. Tulika Saha

International Institute of Information Technology Bangalore

November 17, 2025

Abstract

This report describes the design, implementation, and evaluation of a scene-aware Retrieval-Augmented Generation (RAG) system built over Shakespeare's *Julius Caesar*. The system answers questions about the play by retrieving dialogue-level chunks grouped at the Act–Scene level from a FAISS vector index and conditioning a Groq-hosted LLM on this context. We restrict the knowledge source to the Folger Edition PDF of the play and do not fine-tune any model. Instead, we design an explicit retrieval pipeline, an evaluation testbed of 35 questions (25 factual, 10 analytical), and simple RAGAs-inspired metrics for answer relevancy and faithfulness. The system achieves reasonably strong performance on many core scenes (Soothsayer, Calpurnia's dream, Antony's speech) while exposing limitations in late-play coverage and strict grounding. The project demonstrates both the strengths and fragility of RAG when applied to a single literary work.

1 Introduction

The objective of this project (Assignment 2) is to build a Retrieval-Augmented Generation (RAG) system over a single canonical text: Shakespeare's *The Tragedy of Julius Caesar* in the Folger Edition. The system must behave like an expert tutor for ICSE-level literature:

- answer factual questions such as "Who offers Caesar the crown?" and
- discuss analytical questions on themes, character motivation, and rhetoric.

Instead of training or finetuning an LLM, we:

1. extract and clean the play from the PDF;
2. build dialogue-level chunks annotated with Act and Scene;
3. index these chunks in a FAISS vector store using Nomic embeddings;
4. implement a scene-level retriever that aggregates relevant dialogue chunks; and
5. call a Groq-hosted Llama 3.1 model with a strict context-only prompt.

The assignment also requires a structured evaluation: a golden set of 25 factual questions (provided by the course) and at least 10 additional analytical/thematic questions, plus both quantitative metrics and qualitative error analysis. The rest of this report describes the system and its evaluation in a concise way aligned with these requirements.

2 System Design

2.1 Knowledge Source and ETL

The only knowledge source is the Folger Edition PDF of *Julius Caesar*. We do not use summaries, notes, or external web text. All preprocessing was performed in Jupyter notebooks (`etl.ipynb`, `phas2.ipynb`), which are not part of the runtime system but document the one-time ETL steps:

- Extract raw text from the PDF.
- Remove Folger-specific artefacts such as headers, footers, page numbers, and line labels (`FTLN xxxx`).
- Normalize whitespace and merge broken lines.

2.2 Chunking: Act–Scene and Dialogue Level

We treat the play as a structured hierarchy:

$$\text{Play} \rightarrow \text{Act} \rightarrow \text{Scene} \rightarrow \text{Dialogue / Speech.}$$

Chunking is performed at a *dialogue-level* granularity while retaining Act and Scene metadata:

- Each chunk is designed to contain a coherent piece of dialogue or narration (a few lines spoken by one or more characters).
- Each chunk has metadata: `act`, `scene`, `start_page`, `chunk_id`, and optionally speaker information.

Act–Scene information is used later to aggregate related dialogue chunks into a full-scene context for the LLM.

2.3 Vector Store: FAISS + Nomic Embeddings

For semantic search we use:

- Embeddings: `nomic-embed-text-v1.5`, via the `nomic` client.
- Wrapper: a `NomicEmbeddings` class implementing LangChain's `Embeddings` interface, providing:
  - `embed_documents(texts: List[str])`
  - `embed_query(text: str)`
  - `__call__(text: str)` (to work as a callable in FAISS).
- Vector Store: FAISS index from `langchain_community.vectorstores.FAISS`.

The index and accompanying docstore are persisted in `faiss_index/`. At runtime, `load_faiss_index()` loads this index once and caches it.

2.4 Retrieval Logic: Scene-aware Aggregation

The core retriever is implemented as `retrieve_scene_context(query, top_k)` in `app/rag_grog.py`. The logic is:

1. Perform FAISS similarity search:

   `base_docs = store.similarity_search(query, k=top_k).`

2. For each retrieved doc, read its `act` and `scene` metadata.

3. Apply a majority vote over (`act`, `scene`) pairs to identify the dominant scene.

4. Collect *all* dialogue-level chunks from the docstore that match this Act–Scene.

5. Sort by `start_page` and `chunk_id` to reconstruct the full scene in reading order.

This design reflects the observation that many exam-style questions are about a specific scene (e.g., Calpurnia's dream, the funeral speeches) rather than arbitrary parts of the play. For debugging and ablations, we also support a pure top-k mode (without scene expansion).

2.5 LLM and Prompting via Groq

The generation component uses Groq's hosted Llama 3.1 model:

- Model: `llama-3.1-8b-instant`.
- Client: `ChatGroq` from `langchain-groq` with `GROQ_API_KEY`.

We use a strict system prompt to reduce hallucinations:

> You are a strict tutor for Shakespeare's *Julius Caesar*. You must answer **only** using the provided context. If the answer is not clearly in the context, reply with exactly:
> `"Context insufficient to answer this question."`

The final prompt template is:

> `<context>`
> *[scene-level retrieved dialogue text]*
> `</context>`
> Question: *[user question]*
> Answer:

2.6 Context Length Control and Token Limits

During experimentation, we hit Groq's Tokens-Per-Minute (TPM) limit when sending entire scenes. To avoid 413 *request too large* errors, we implemented a context limiter:

- Concatenate retrieved chunks into a single text.
- Approximate tokens by whitespace splitting.
- Stop once we reach a budget of about 2500 tokens.

This keeps the request within safe bounds while still providing enough context for most questions.

2.7 FastAPI RAG Endpoint

For interaction and evaluation, we built a small API in `app/api.py` using FastAPI:

- Endpoint: `POST /query`
- Request:

```
{
  "question": "Who addresses the plebeians after Caesar is killed?",
  "top_k": 10
}
```

- Response:

```
{
  "answer": "...",
  "sources": [
    {
      "act": 3,
      "scene": 2,
      "text_preview": "PLEBEIAN Peace, silence! Brutus speaks."
    },
    ...
  ]
}
```

This API reuses `retrieve_scene_context`, the LLM prompt, and the context limiter, and exposes enough metadata for debugging which scene was retrieved.

3 Evaluation Setup

3.1 Golden Testbed: 25 + 10 Questions

Following the assignment specification, we created an `evaluation.json` file with:

- The official 25 factual questions given in the assignment, each with an *ideal answer* string.
- 10 additional analytical/thematic questions designed by us, for example:
  - "What are Brutus's internal conflicts as shown in his soliloquy in Act 2, Scene 1?"
  - "How does Cassius manipulate Brutus into joining the conspiracy?"
  - "How does Antony use rhetoric and irony in his funeral speech to turn the crowd against the conspirators?"
  - "In what ways do the plebeians in Act 3, Scene 2 illustrate the theme of mob mentality?"
  - "Why can Brutus be seen as both a patriot and a tragic hero?"

In total, the testbed contains 35 questions: 25 factual, 10 analytical.

3.2 Evaluation Script and Procedure

The script `evaluation.py` automates the evaluation:

1. For each question in `evaluation.json`:
   (a) Call `retrieve_scene_context(question, top_k=10)` to get candidate dialogue chunks.
   (b) Build a limited context (approximately 2500 tokens) from those chunks.
   (c) Invoke the Groq LLM with the strict RAG prompt.
   (d) Record the system answer.
2. Compute two RAGAs-inspired metrics:
   - **Answer relevancy:** approximate lexical overlap between the system answer and the ideal answer.
   - **Faithfulness:** fraction of answer tokens that also appear in the retrieved context.
3. Save detailed results (question, ideal answer, system answer, metrics) to `evaluation_results.json`.

These metrics are simple heuristics inspired by RAGAs' notions of answer relevancy and faithfulness, not a full integration of the library, but they are sufficient to compare behaviour across the testbed.

4 Results and Discussion

4.1 Quantitative Summary

On the 35-question testbed, the aggregated metrics are:

```
"overall": {
  "avg_answer_relevancy": 0.9103937589318558,
  "avg_faithfulness": 0.653395344957812,
  "num_questions": 35
}
```

Although our relevancy heuristic can exceed 1.0 for individual answers (due to repeated token overlap), the high average indicates that the system's answers usually share substantial overlap with the ideal answers. The average faithfulness of around 0.65 suggests that a majority of answer tokens are grounded in the retrieved context, but some speculative content remains.

4.2 Success Cases

Soothsayer's Warning. For "What does the Soothsayer say to Caesar?", the system responds with: *"Beware the ides of March."*, achieving perfect relevancy and faithfulness. Retrieval selects the correct early scene, and the LLM simply repeats the quoted line.

Calpurnia's Dream. For "Why does Calpurnia urge Caesar to stay home?", the system gives a detailed and accurate explanation, referencing her dream of Caesar's statue spouting blood, strange omens in the streets, and the augurers' warnings. This aligns well with the ideal answer and shows that scene-level aggregation around Act 2, Scene 2 works as intended.

Antony's Speech and Mob Mentality. For questions about Antony's funeral speech and the plebeians' reaction, the system produces rich analytical answers describing:

- Antony's repeated use of "Brutus is an honorable man" as verbal irony.
- Caesar's refusal of the crown and his generosity, used to contradict the charge of ambition.
- The plebeians' shift from calm acceptance to rage, illustrating mob mentality.

These answers are well matched to ICSE-level literature expectations and are strongly supported by the retrieved Act 3, Scene 2 context.

4.3 Failure Cases

Caesar's First Entrance. For "How does Caesar first enter the play?", the ideal answer is a triumphal procession after defeating Pompey's sons. Instead, the system incorrectly describes Caesar entering in his nightgown during a storm, mixing up scenes. This suggests that either retrieval failed to focus on the opening scene or the LLM overrode the context with its own prior knowledge.

Who Addresses the Plebeians First. For "After the assassination of Caesar, which of the conspirators addresses the plebeians first?", the ideal answer is Brutus. The system answers "Cassius," even though the retrieved context includes lines like "Brutus goes into the pulpit" and plebeians saying "I will hear Brutus speak." This is a generation error despite adequate retrieval.

Late-Play Events. Several questions about late-play events (Portia's death, the ghost of Caesar, details of the battle, Brutus's and Cassius's suicides) often yield: `"Context insufficient to answer this question."` This indicates that our semantic search plus scene voting does not always give good coverage for Acts 4 and 5, leading to genuine retrieval failures.

Instruction Drift. In some answers, the LLM begins with the mandated string "Context insufficient to answer this question." and then continues with speculative analysis beginning with "However...". This violates our intended hard constraint and introduces hallucinated content, lowering faithfulness.

4.4 Overall Assessment

Overall, the system performs strongly on:

- famous, central scenes (Soothsayer, Calpurnia, assassination, funeral speeches);
- analytical questions about rhetoric, mob mentality, and themes when the right scene is retrieved.

It struggles more with:

- early entrance and late battle/ghost scenes (coverage issue);
- strict enforcement of the "context-only" rule (prompting and post-processing issue).

These observations are consistent with the quantitative metrics: relatively high answer relevancy and moderate faithfulness.

5 Conclusion and Future Work

We implemented a complete scene-aware RAG system over Shakespeare's *Julius Caesar*, using only the Folger Edition as the knowledge source. Dialogue-level chunks with Act–Scene metadata are embedded with Nomic embeddings and stored in FAISS. A majority-vote scene retriever aggregates the most relevant dialogue chunks, and a Groq-hosted Llama 3.1 model generates answers under a strict context-only prompt. An evaluation testbed of 35 questions (25 factual, 10 analytical) and RAGAs-inspired metrics provides a realistic picture of the system's strengths and weaknesses.

Future work can improve the system in several ways:

- Combine scene-level aggregation with cross-scene top-k retrieval, especially for questions that span multiple Acts.
- Strengthen enforcement of the "context insufficient" rule, including post-processing that truncates any text after this phrase.
- Integrate the official RAGAs framework for more robust evaluation of relevancy, faithfulness, and answer correctness.
- Analyse retrieval coverage by Act and Scene to systematically correct underrepresented parts of the play.

Despite its limitations, the project successfully demonstrates how RAG can be applied to a single literary work, and how careful evaluation reveals both where such systems excel and where they mislead.