GRCResponder

Brianna Steier, Liam Gass, Elijah Tavares, Cael Howard,
June Kim, Rish Sharma, Angel Li

accenture

UCI
Donald Bren
School of Information
& Computer Sciences

Table of Contents

Problem Statement

- What is a General Rate Case proceeding?
  - Used to address costs of operation and maintenance for utility companies
  - Carried out by the California Public Utilities Commission
- Utility companies need to be able to
  - Manage documents
  - Respond to regulatory inquiries
  - Ensure consistent responses across submissions
  - Meet strict deadlines and compliance requirements

Approach

- GRCResponder: an AI-powered tool that provides a solution in how utility companies can manage their General Rate Case processes.
  - Automatic processing of GRC documentation via web scraping
  - Semantic search across documents stored in DB to find relevant data
  - AI-powered assistant that generates responses to inquiries by user

Current Status

- Initial prototype using Llama 3.2 built for legal question answering
- Prompt engineering implemented with system prompt
- Developed a basic Retrieval-Augmented Generation (RAG) pipeline
- Created an Semi-automated testing framework for evaluating accuracy, consistency, and response quality

```python
def generate_response(query, vector_store, llm):
    # Retrieve relevant documents
    retrieved_text, _ = retrieve_documents(query, vector_store)
    system_prompt = (
        "You are an assistant for legal question answering. "
        "Use the following retrieved context to answer the question. "
        "If you don't know the answer, say that you don't know.\n\n" +
        retrieved_text
    )
```

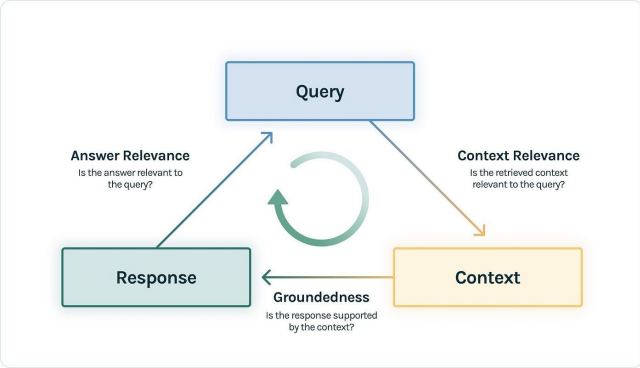
```python
def test_llm_response_quality(self):
    # Test that the LLM generates a non-empty response that includes context.
    test_query = "Explain how RAG works in legal document analysis."
    response = generate_response(test_query, self.vector_store, self.llm)
    self.assertIn("Source:", response, "LLM response does not include retrieved document
    self.assertTrue(len(response) > 20, "LLM response is too short; might indicate an err
```

Current Status

Researching automated RAG testing frameworks: ARES

Three Essential Components:

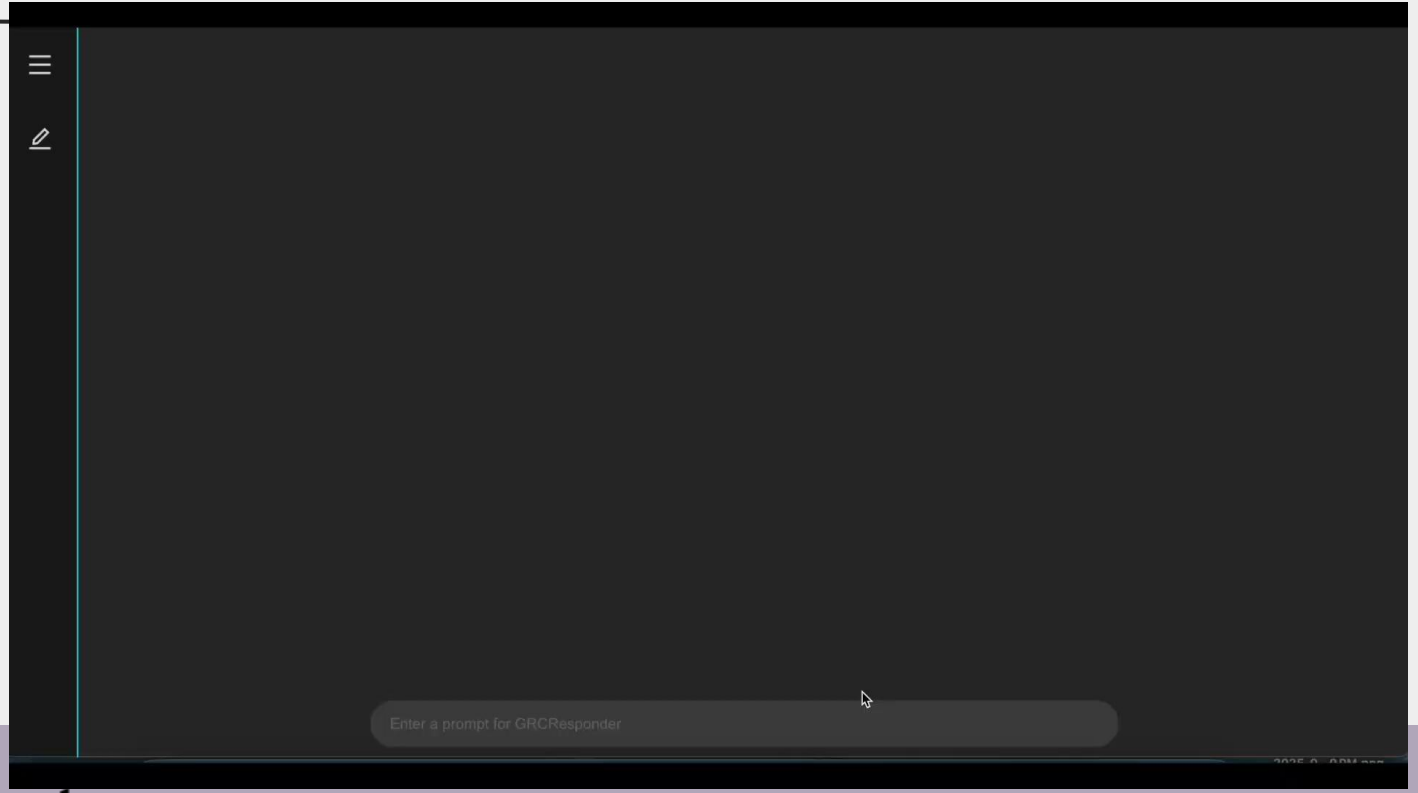
- **Human Preference Validation Set:**
  - Annotated query, document, & answer triples to assess context relevance, answer faithfulness, and answer relevance.
- **Few-Shot Examples:**
  - Curated examples for scoring the above criteria.
- **Large Unlabeled Dataset:**
  - Query-document-answer triples generated by your RAG system.



ARES Functionality:

- Automatically scores your RAG system using these components.
- Enables standardized comparisons and guides continuous improvement

Current Status



Current Status

API Endpoints

- Function: populating sidebar and viewing past conversations
  - POST ‘/conversations’
  - POST ‘/conversations/{conversation_id}/messages’
  - GET ‘/conversations’
  - GET ‘/conversations/{conversation_id}/messages’
- Function: communication with LLM
  - POST ‘/query’
  - GET ‘/response/documents/{conversation_id}’

Technical Details: Search

Document Retrieval

Semantic Search

- Vector Storage: ChromaDB for text embedding storage and retrieval
  - Used to locate relevant documents used to address user queries
- Embedding Function: sentence-transformers/all-MiniLM-L6-v2

Topic Based Navigation

- postgreSQL: document and proceeding metadata storage
  - Uses GIN indexing to map unique words to text chunks
  - Allows for retrieval of only chunk text containing query words

Technical Details

Tech Stack:

- TypeScript with React – frontend
- Python (FastAPI) – backend
- Microsoft Azure

Document Retrieval

Semantic Search

- Vector Storage: ChromaDB for text embedding storage and retrieval
  - Used to locate relevant documents used to address user queries

Topic Based Navigation

- postgreSQL: document and proceeding metadata storage
  - Uses GIN indexing to map unique words to text chunks
  - Allows for retrieval of only chunk text containing query words

Technical Details: LLM

Model

- Q&A LLM: Llama 3.2 Instruct 3B with 8 bit quantization
- Embedding Model: all-MiniLM-L6-v2

Naive RAG

- Indexing
  - Chunk and encode data
- Retrieval
  - Cosine similarity to find chunks relevant to query
- Generation
  - Pass retrieved data to LLM as context with query

Example

- Sample on RAG papers

```
================================ Human Message =================================

Hello!
================================== Ai Message ==================================

Hello! How can I assist you today? Do you have a specific question or topic you'd like to discuss? I'll do my best to
 help based on the context I've been trained on.
================================ Human Message =================================

Explain how RAG works
================================== Ai Message ==================================

Based on the retrieved context, it appears that RAG (Retrieval-Augmented Generation) is a technology that enhances the
 capabilities of Large Language Models (LLMs) by integrating parameterized knowledge from language models with extensive
  non-parameterized data from external knowledge bases.

Here's a high-level overview of how RAG works:

1. **Knowledge Retrieval**: RAG uses an external knowledge base to retrieve relevant information related to a given
 input question or task.
2. **Parameterized Knowledge Integration**: The retrieved knowledge is then integrated with the LLM using parameterized
 knowledge, which allows for flexible and dynamic updating of the model's knowledge graph.
3. **Soft Prompting**: RAG employs soft prompting, which involves providing the LLM with a prompt that encourages it to
 generate answers based on both its internal knowledge and the external knowledge base.
4. **LLM Generation**: The LLM generates a response to the input question or task using its internal knowledge and the
 integrated parameterized knowledge.

The goal of RAG is to leverage the strengths of both LLMs and external knowledge bases to improve the accuracy and
robustness of language understanding tasks, such as question-answering and text generation.
```

Evaluating Performance

The Task

- Proceedings date all the way back to 1981
- Associated files are only available for proceedings issued from: July 2000 - Present
- Proceeding Search with 0 parameters provided 126 pages each with 100 proceedings - resulting in 12,660 proceedings
- each of these proceedings ranges from 2 or 3 associated documents to as many as 100+ from what I've seen - averaging around 10-12 documents (cases have grown bigger in the more recent years than they were in the old days)
- A conservative estimate is 100k total pdfs to process

Next Steps…

- **Refine RAG Testing & Evaluation:**
  - Continue research and then implement a fully-automated testing framework
  - Evaluation metrics:
    i. Search response time < 2 seconds
    ii. Document processing accuracy > 95%
    iii. >90% accuracy in relevant document retrieval
- **Develop Citation Management:**
  - Design a structured approach for handling citations beyond document-based retrieval
- **Enhance User Experience:**
  - Develop settings so users can upload documents or set date constraints for data to better support their upcoming queries
- **Optimize Embedding & Search:**
  - Identify meaningful metadata and unit test for search performance

Questions?