**CORRECTIVE RAG**

* CRAG focuses on improving the accuracy and relevance of generated responses by incorporating mechanisms for self-reflection and self-grading of retrieved documents.
* Methods:
  + retrieve\_documents(self, query: str) -> List[str]
    - Private method for retrieving relevant documents based on the input query.
    - Performs the following steps:
      * Verifies the initialization of the vector store, raising an error if uninitialized.
      * Executes a similarity search using the vector store to fetch relevant documents.
      * Extracts and returns the page content from the retrieved documents.
    - Logs errors and returns an empty list in case of failure.
  + grade\_relevance(self, query: str, documents: List[str]) -> bool
    - Private method for grading the relevance of retrieved documents based on a query.
    - Performs the following steps:
      * Checks if the document list is empty and immediately returns False if no documents are available.
      * Constructs a grading prompt by embedding the query and document content for evaluation.
      * Utilizes a model to generate a response based on the prompt, limited by a specified maximum length.
    - Returns True if the model response contains "YES", indicating relevance.
    - Returns False if no relevant documents are identified or if the response is "NO."
  + transform\_query(self, query: str) -> str
    - Private method for transforming a query into a search-friendly format.
    - Performs the following steps:
      * Constructs a prompt instructing the model to rewrite the query for better web search compatibility.
      * Embeds the original query within the prompt to provide context for transformation.
      * Uses a model to generate a transformed search query, constrained by a specified maximum length.
    - Returns the optimized search query generated by the model.
  + web\_search(self, query: str) -> str
    - Private method for performing a web search with error handling.
    - Performs the following steps:
      * Executes the search query using an integrated search tool.
      * Returns the search results if successful.
      * Captures and logs any exceptions encountered during the search process.
    - Returns an empty string if the search fails.
  + generate\_response(self, query: str, context: str) -> str
    - Private method for generating a final response based on the provided context and query.
    - Performs the following steps:
      * Constructs a response prompt by embedding the query and contextual information.
      * Instructs the model to generate a detailed answer using the given context.
      * Utilizes the model to generate a response constrained by a specified maximum length.
    - Returns the generated response as the final answer.
  + query(self, query: str) -> Dict[str, str>
    - Private method for executing the complete RAG pipeline with error handling.
    - Performs the following steps:
      * **Document Retrieval:** Retrieves relevant documents for the given query.
      * **Relevance Grading:** Evaluates the relevance of the retrieved documents.
      * Response Generation:
        + If relevant documents are found, generates a response using the retrieved content as context.
        + If no relevant documents are found, transforms the query and performs a web search, then generates a response based on the search results.
      * **Error Handling:** Logs any exceptions encountered during the process and provides a fallback response structure.
    - Returns a structured dictionary containing:
      * The original query
      * A flag indicating whether web search was used
      * Retrieved documents
      * The generated response
      * Error information (if any).

**FUSION RAG**

* Fusion RAG is an advanced paradigm of RAG that focuses on combining results from multiple retrieval sources or knowledge representations to create a more comprehensive and accurate response.
* Methods:
  + generate\_similar\_queries(self, query: str, num\_queries: int = 4) -> List[str]
    - Private method for generating similar queries.
    - Performs the following steps:
      * Iteratively generates multiple variations of the input query, up to the specified number of queries.
      * Utilizes a query generation chain to produce each similar query.
      * Appends each generated query to a result list.
    - Returns a list containing the generated similar queries.
  + reciprocal\_rank\_fusion(self, results\_lists: List[List[Dict]], k: int = 60) -> List[Dict]
    - Private method for implementing Reciprocal Rank Fusion to aggregate multiple result lists.
    - Performs the following steps:
      * Score Aggregation: Iterates over each result list and assigns a score to each document based on its rank using the formula 1 / (rank + k).
      * Sorting: Sorts the documents by their aggregated fusion scores in descending order.
      * Result Reconstruction:
        + Reconstructs the document list by preserving the original document data while adding the fusion scores.
        + Ensures each document in the final list includes the computed fusion score.
    - Returns a list of fused results with enhanced ranking based on the RRF scoring method.
  + search\_vector\_store(self, query: str, k: int = 5) -> List[Dict]
    - Private method for searching the vector store using a query.
    - Performs the following steps:
      * **Validation:** Checks if the vector store is initialized, raising an error if it is not available.
      * **Similarity Search:** Executes a similarity search on the vector store to retrieve the top k results.
      * **Result Formatting:** Constructs a list of dictionaries containing:
        + An identifier for each result
        + The retrieved document
        + The corresponding similarity score
    - Returns a structured list of search results with document and score information.
  + generate\_response(self, query: str, context: str) -> str
    - Private method for generating a response based on the query and context.
    - Performs the following steps:
      * Input Preparation: Constructs an input text by embedding the query and context in a structured format.
      * GPU Utilization: Transfers the tokenized input to the GPU if available for faster processing.
      * Model Inference: Generates the response with parameters for Maximum length, beam search (num\_beams), temperature, top-p sampling, and repetition penalty.
      * Decoding: Decodes the model's output to generate a human-readable response, skipping special tokens.
    - Returns the generated response as a string.
  + process\_query(self, query: str) -> str
    - Private method for processing a query through the Fusion RAG pipeline.
    - Performs the following steps:
      * Generate Similar Queries: Creates variations of the input query and combines them with the original query.
      * Search Vector Store: Retrieves search results for each query using the vector store and collects all results.
      * Reciprocal Rank Fusion**:** Aggregates the search results using the RRF technique to prioritize the most relevant documents.
      * Context Combination: Extracts the page content from the top fused results and combines them into a unified context.
      * Response Generation: Generates the final response using the combined context and the original query.
    - Returns the final generated response.

**REFEED RAG**

* REFEED represents a significant advancement in RAG techniques by providing a framework that leverages retrieval feedback to effectively refine LLM outputs.
* Methods:
  + generate\_answers(self, query: str, vectorstore: FAISS, num\_answers: int = 3) -> List[str]
    - Private method for generating multiple candidate answers to a query.
    - Performs the following steps:
      * Prompt Construction: Defines a detailed prompt template instructing the model to generate comprehensive answers based on the provided context.
      * LLM Chain Setup: Initializes a language model chain with the prompt template to facilitate response generation.
      * Document Retrieval: Performs a similarity search on the vector store with an increased k value to obtain a more extensive context for better answer quality.
      * Answer Generation: Iteratively generates multiple candidate answers by running the LLM chain on the query and context for the specified number of answers.
      * Result Collection: Collects and stores each generated response in the answers list.
    - Returns a list of multiple candidate answers generated by the LLM chain.
  + retrieve\_passages(self, answers: List[str], vectorstore: FAISS) -> List[str]
    - Private method for retrieving relevant passages for each generated answer.
    - Performs the following steps:
      * Iterative Retrieval: For each answer, performs a similarity search in the vector store to retrieve the top k passages.
      * Passage Extraction: Extracts the page content from each retrieved passage and adds it to the collection.
      * Duplicate Removal: Eliminates duplicate passages while preserving the original order of retrieval.
    - Returns a list of unique and relevant passages corresponding to the provided answers.
  + rank\_passages(self, query: str, passages: List[str>) -> List[str]
    - Private method for ranking passages based on their relevance to the query.
    - Performs the following steps:
      * Embedding Generation: Computes the embedding for the query and embeddings for each passage using the embeddings model.
      * Cosine Similarity Calculation: Computes cosine similarities between the query embedding and each passage embedding to determine relevance scores.
      * Sorting: Sorts the passages in descending order based on their similarity scores.
      * Result Compilation: Reorders the passages according to their ranked indices for output.
    - Returns a list of ranked passages, with the most relevant ones appearing first.
  + generate\_final\_answer(self, query: str, ranked\_passages: List[str>) -> str
    - Private method for generating the final answer using ranked passages.
    - Performs the following steps:
      * Context Construction: Combines the top k\_passages from the ranked passages into a single coherent context.
      * Prompt Creation: Defines a detailed prompt template instructing the model to generate a comprehensive answer by connecting and including all relevant information from the context.
      * LLM Chain Execution: Initializes an LLM chain with the prompt template and executes it to generate the final answer.
      * Answer Generation: Runs the chain using the provided query and context to produce the most complete and informative answer possible.
    - Returns the generated final answer as a string.

**SELF REFLECTIVE RAG**

* Self-Reflective RAG enhances an LM’s quality and factuality through retrieval and self-reflection.
* Methods:
  + retrieve(self, query: str, k: int = 5) -> List[Document]
    - Private method for retrieving relevant documents using the FAISS vector store.
    - Performs the following steps:
      * Query Execution: Searches the FAISS vector store with the given query to retrieve the top k relevant documents.
      * Result Compilation: Collects and returns the retrieved documents in a structured list format.
    - Returns a list of Document objects containing the most relevant search results.
  + grade\_documents(self, query: str, documents: List[Document]) -> List[Tuple[Document, float]]
    - Private method for grading the relevance of retrieved documents using cosine similarity.
    - Performs the following steps:
      * Embedding Generation: Computes the query embedding and embeddings for the content of each document.
      * Cosine Similarity Calculation: Calculates the cosine similarity between the query embedding and each document embedding to assess their relevance.
      * Result Pairing: Pairs each document with its corresponding similarity score.
    - Returns a list of tuples containing Document objects and their relevance scores.
  + check\_relevance(self, graded\_docs: List[Tuple[Document, float]]) -> bool
    - Private method for assessing the relevance of retrieved documents based on their graded scores.
    - Performs the following steps:
      * Empty Check: Immediately returns False if no graded documents are provided.
      * Threshold Evaluation: Iterates through the graded documents to check if any document's relevance score meets or exceeds the defined relevance threshold.
    - Returns True if at least one document is considered relevant, otherwise False.
  + rewrite\_query(self, original\_query: str, graded\_docs: List[Tuple[Document, float]]) -> str
    - Private method for rewriting the query based on the context of graded documents.
    - Performs the following steps:
      * Context Construction: Extracts relevant snippets from the graded documents along with their relevance scores and combines them into a cohesive context.
      * LLMChain Execution: Runs the LangChain rewrite chain using the original query and the constructed context to generate a refined query.
      * Result Cleanup: Strips any leading or trailing whitespace from the generated query.
    - Returns the rewritten query as a string.
  + generate\_response(self, query: str, relevant\_docs: List[Tuple[Document, float]]) -> str
    - Private method for generating a response using LangChain's LLMChain based on relevant documents.
    - Performs the following steps:
      * Context Construction: Combines the full content of relevant documents with their corresponding relevance scores to create a comprehensive context.
      * LLMChain Execution: Runs the response generation chain using the constructed context and the provided query to generate a detailed answer.
      * Result Cleanup: Removes any leading or trailing whitespace from the generated response.
    - Returns the final generated response as a string.

**ITERATIVE RAG**

* Iterative RAG increases the accuracy of retrieving the most relevant content from the knowledge base.
* Methods:
  + select\_exemplars(self, documents: List[str], state: RetrievalState) -> List[str]
    - Private method for selecting the most relevant exemplars from retrieved documents.
    - Performs the following steps:
      * History Check: If there is no document history, selects the first three documents as exemplars.
      * Scoring Evaluation: Computes a combined score for each document based on relevance and diversity.
      * Weighting Strategy: Applies a weighted scoring formula (70% relevance, 30% diversity).
      * Sorting and Selection: Sorts documents by their scores in descending order and selects the top three exemplars.
    - Returns a list of selected exemplar documents.
  + compute\_relevance(self, document: str, query: str) -> float
    - Private method for computing the relevance score between a document and a query.
    - Performs the following steps:
      * Embedding Generation: Generates embeddings for both the document and the query using the embeddings model.
      * Cosine Similarity Calculation: Computes the cosine similarity between the document and query vectors by calculating the dot product of their embeddings.
      * Relevance Score: Returns the computed relevance score as a floating-point value.
    - Returns a float representing the relevance score between the document and the query.
  + compute\_diversity(self, document: str, previous\_docs: List[str]) -> float
    - Private method for computing the diversity score of a document compared to previous documents.
    - Performs the following steps:
      * Embedding Generation: Generates embeddings for the current document and for each of the previous documents using the embeddings model.
      * Similarity Calculation: Computes the cosine similarities between the document and each of the previous documents.
      * Diversity Score: Calculates the diversity score. Higher similarity results in a lower diversity score.
    - Returns a float representing the diversity score of the document relative to the previous documents.
  + generate\_answer(self, query: str, exemplars: List[str]) -> str
    - Private method for generating an answer using an LLM with selected exemplars.
    - Performs the following steps:
      * Prompt Creation: Defines a prompt template that combines the query and context to instruct the model in generating the answer.
      * LLMChain Execution: Initializes and runs the LLM chain using the model and the prompt, providing the query and context (exemplars) as inputs.
      * Answer Generation: Generates a detailed answer based on the context provided by the exemplars.
    - Returns the generated answer as a string.
  + refine\_query(self, query: str, exemplars: List[str]) -> str
    - Private method for refining a query based on retrieved exemplars.
    - Performs the following steps:
      * Prompt Creation: Defines a prompt template that combines the original query and the context (exemplars) to guide the model in refining the query.
      * LLMChain Execution: Initializes and runs the LLM chain using the model and the prompt, providing the query and context (exemplars) as inputs.
      * Query Refinement: Refines the query based on the context provided by the exemplars.
    - Returns the refined query as a string.
  + grade\_document(self, query: str, document: str) -> float
    - Private method for grading the relevance of a document based on the query using a grading prompt.
    - Performs the following steps:
      * Prompt Creation: Defines a prompt template that combines the query and the document to guide the model in evaluating the relevance of the document.
      * LLMChain Execution: Initializes and runs the LLM chain using the model and the prompt, providing the query and document as inputs.
      * Result Parsing: Attempts to convert the model's output into a float to represent the relevance score. If the conversion fails, defaults to a score of 0.0.
    - Returns the relevance score as a float.