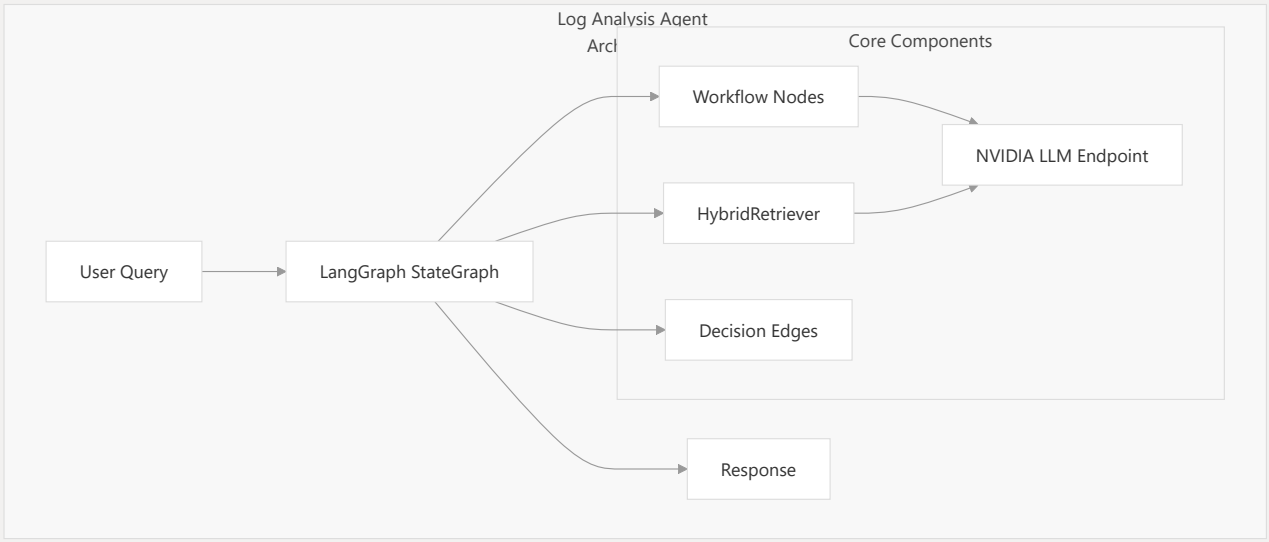
Log Analysis Agent

Relevant source files

The Log Analysis Agent, also known as BAT.AI (Bug Automation Tool), is a self-corrective multi-agent RAG system designed to help developers analyze log files and extract actionable insights using large language models. This document explains the architecture, components, and implementation details of this agent system located in the `community/log_analysis_multi_agent_rag` directory. For information about other multi-agent systems, see Multi-Agent Systems or Enterprise Chatbot with Glean.

1. System Overview

The Log Analysis Agent uses LangGraph to orchestrate a graph-based workflow that processes user queries about log files, retrieves relevant log entries, grades document relevance, generates responses, and self-corrects through query transformation. The system employs a hybrid retrieval approach that combines BM25 and FAISS with NVIDIA AI endpoints for embeddings and reranking.



Sources: `community/log_analysis_multi_agent_rag/README.md` `1-67`
`community/log_analysis_multi_agent_rag/bat_ai.py` `1-61`

2. Key Components

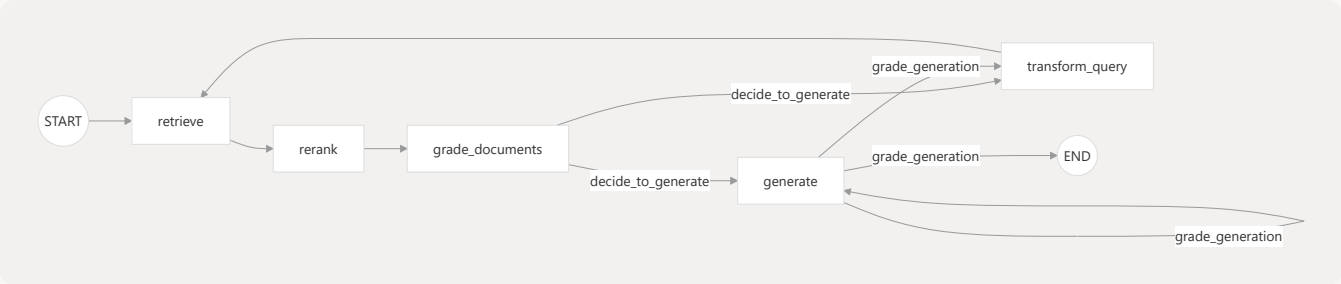
The Log Analysis Agent consists of several key components that work together to form the complete system:

Component	File	Purpose
StateGraph Definition	`bat_ai.py`	Defines the main workflow graph using LangGraph
Node Implementations	`graphnodes.py`	Contains the implementation of each workflow step
Edge Logic	`graphedges.py`	Contains the decision-making logic between nodes
Hybrid Retriever	`multiagent.py`	Implements document retrieval combining BM25 and FAISS
Output Models	`binary_score_models.py`	Defines structured output formats for grading
Utility Functions	`utils.py`	Sets up LLM prompts and chains for various operations

Sources: `community/log_analysis_multi_agent_rag/README.md` `23-29`

3. Graph Structure and Workflow

The Log Analysis Agent uses a directed graph to control the flow of information processing. The graph is defined in `bat_ai.py` with specific nodes for each step of the process and conditional edges to direct the flow based on the results of each step.



Sources: `community/log_analysis_multi_agent_rag/bat_ai.py` `29-61`
`community/log_analysis_multi_agent_rag/README.md` `58-67`

3.1 Graph State

The state that flows through the graph is defined as a `GraphState` TypedDict with the following structure:

```python
class GraphState(TypedDict):
    path: str               # Log file path
    question: str           # The current question being processed
    generation: str         # The current LLM generation
    documents: List[str]    # List of relevant documents retrieved
```

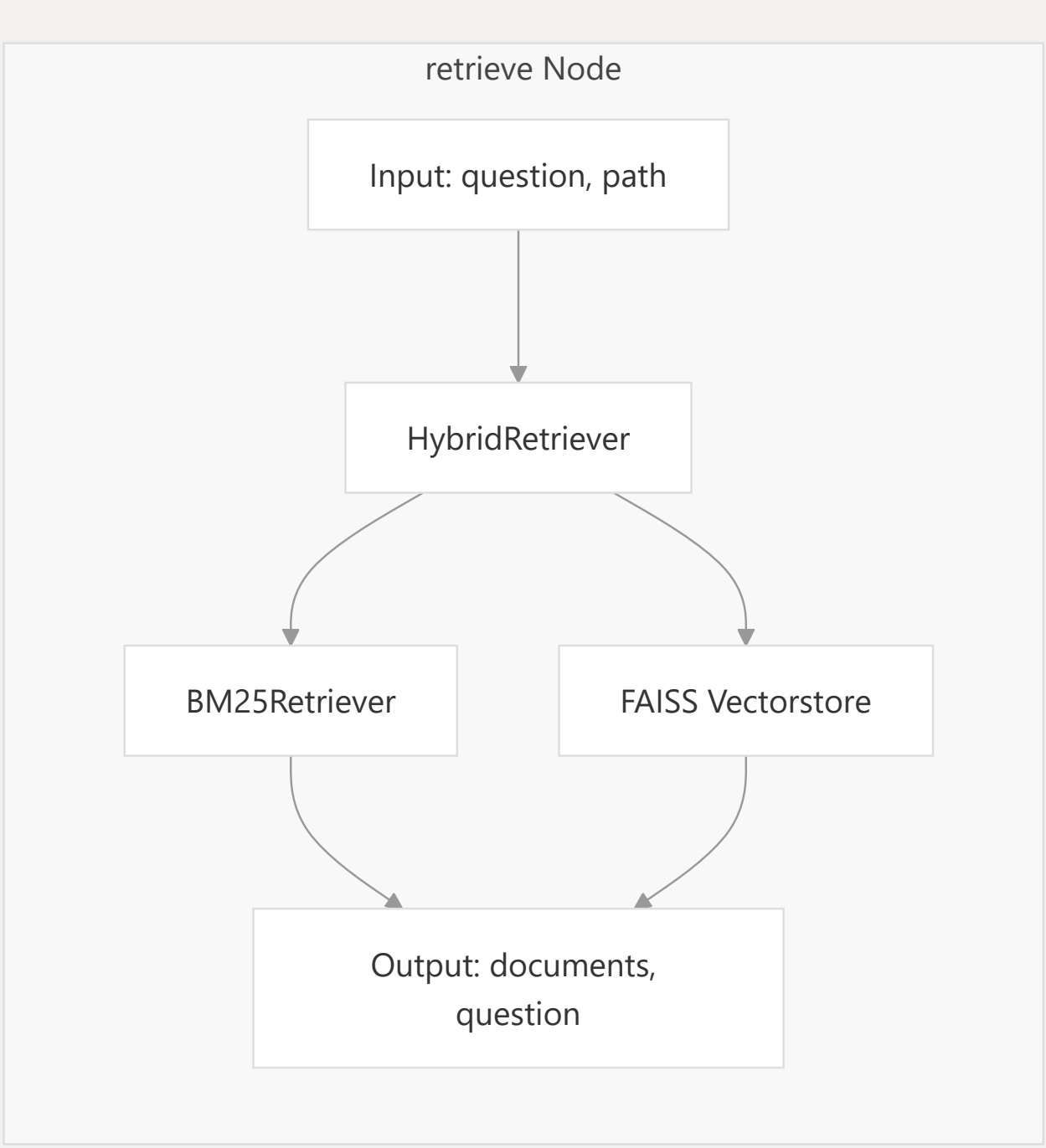
Sources: `community/log_analysis_multi_agent_rag/bat_ai.py` `7-26`

4. Node Implementations

The system defines five key node functions in the `Nodes` class that perform the core operations of the workflow:

4.1 Document Retrieval

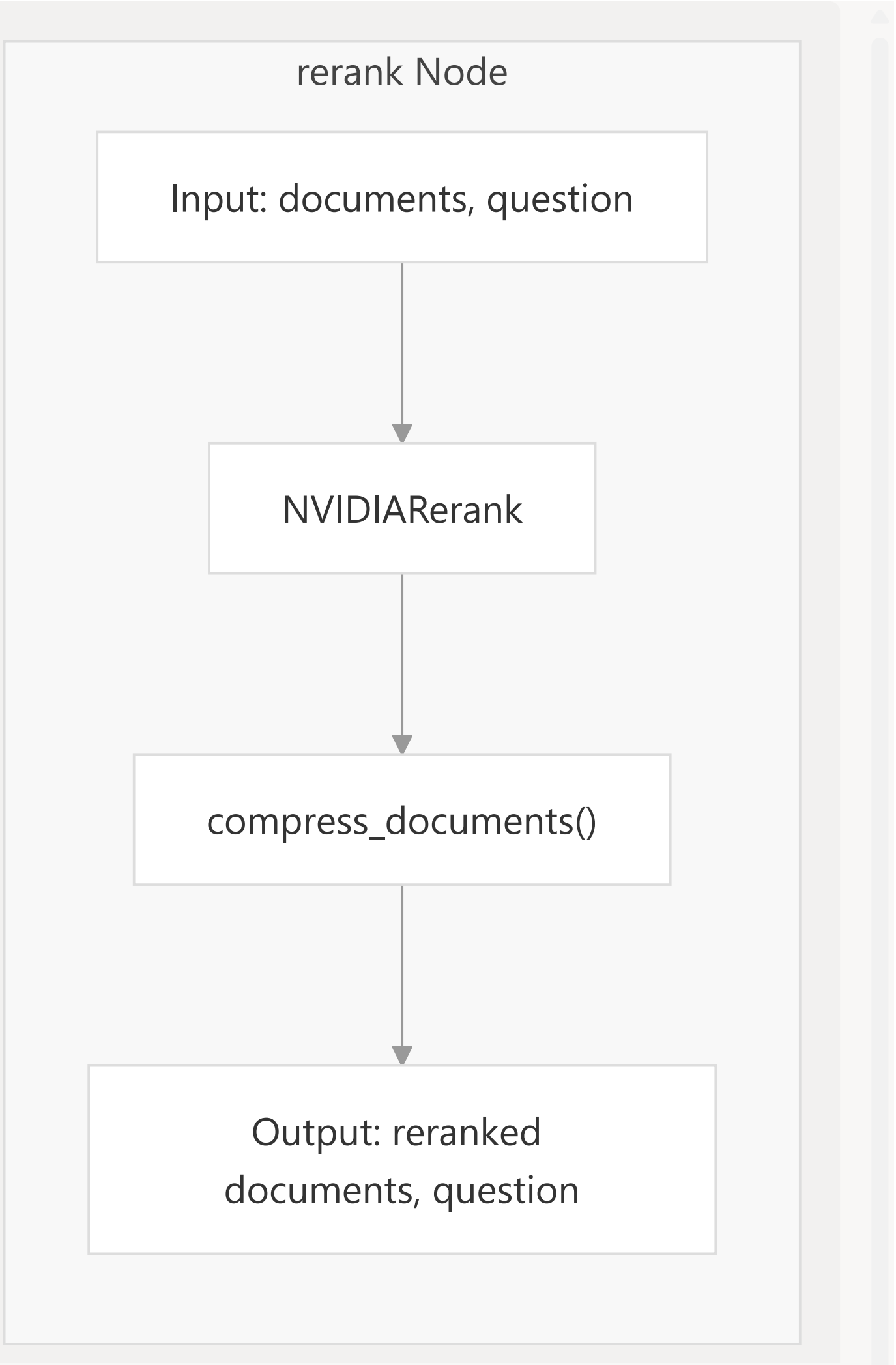
The `retrieve` node loads the log file and uses a hybrid retrieval approach to find relevant log entries:



Sources: `community/log_analysis_multi_agent_rag/graphnodes.py` `11-22`
`community/log_analysis_multi_agent_rag/multiagent.py` `11-43`

4.2 Document Reranking

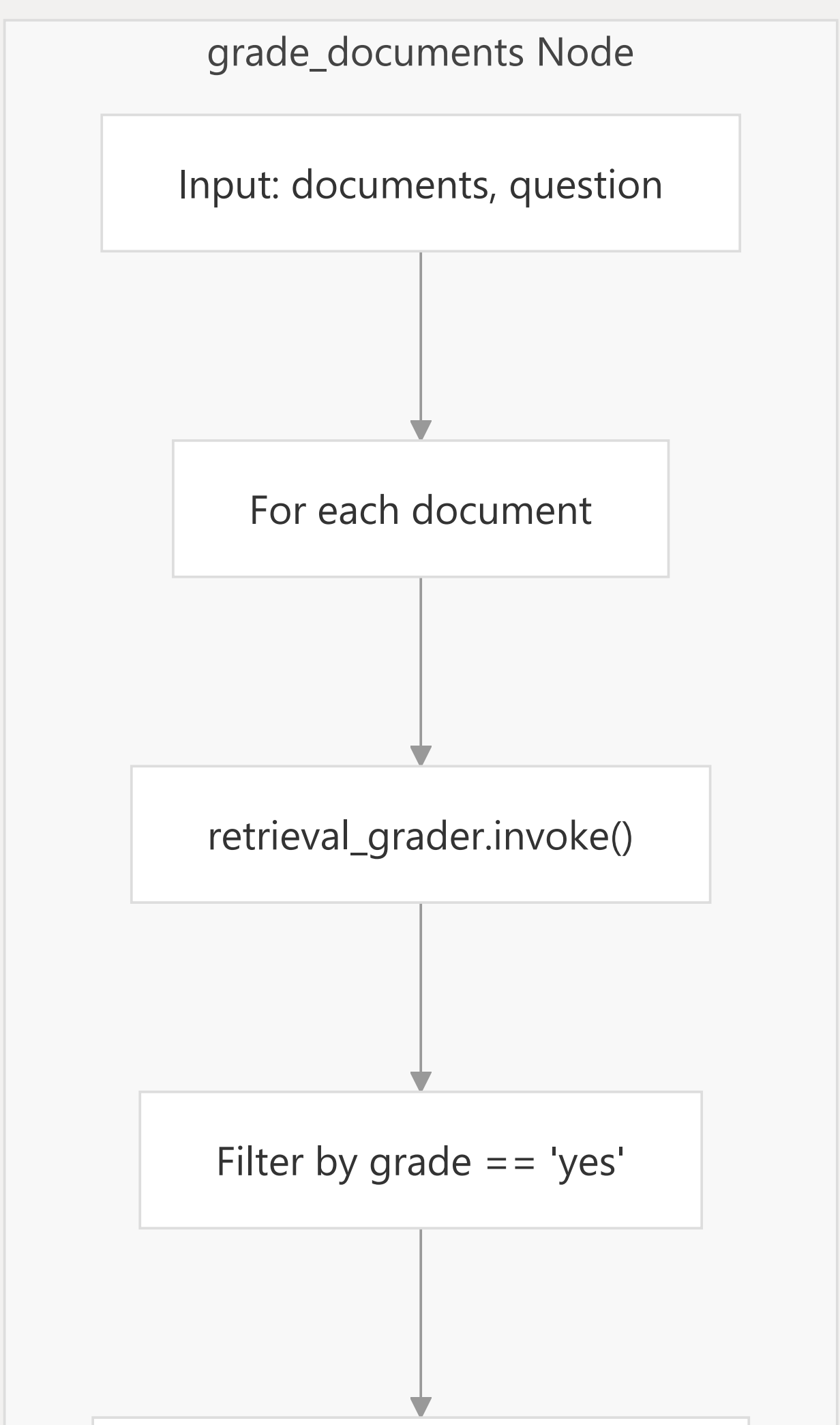
The `rerank` node uses the NVIDIA reranking model to prioritize the most relevant documents:


rerank Node
Input: documents, question
NVIDIARerank
compress_documents()
Output: reranked documents, question

Sources: `community/log_analysis_multi_agent_rag/graphnodes.py` `24-31`

4.3 Document Grading

The `grade_documents` node evaluates each document for relevance to the query using the LLM:

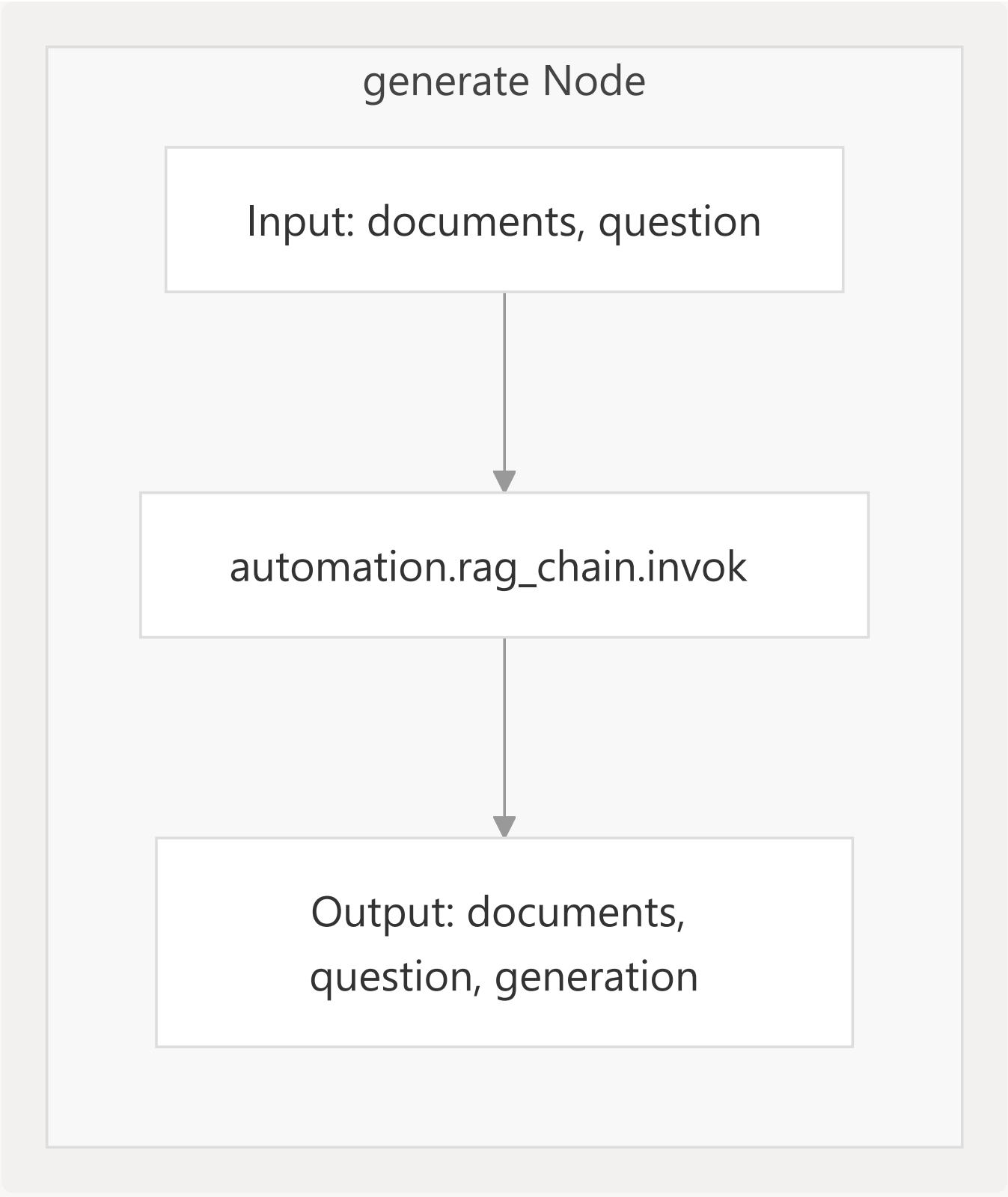

grade_documents Node
Input: documents, question
For each document
retrieval_grader.invoke()
Filter by grade == 'yes'

Output: filtered documents, question

Sources: `community/log_analysis_multi_agent_rag/graphnodes.py` `42-59`

4.4 Response Generation

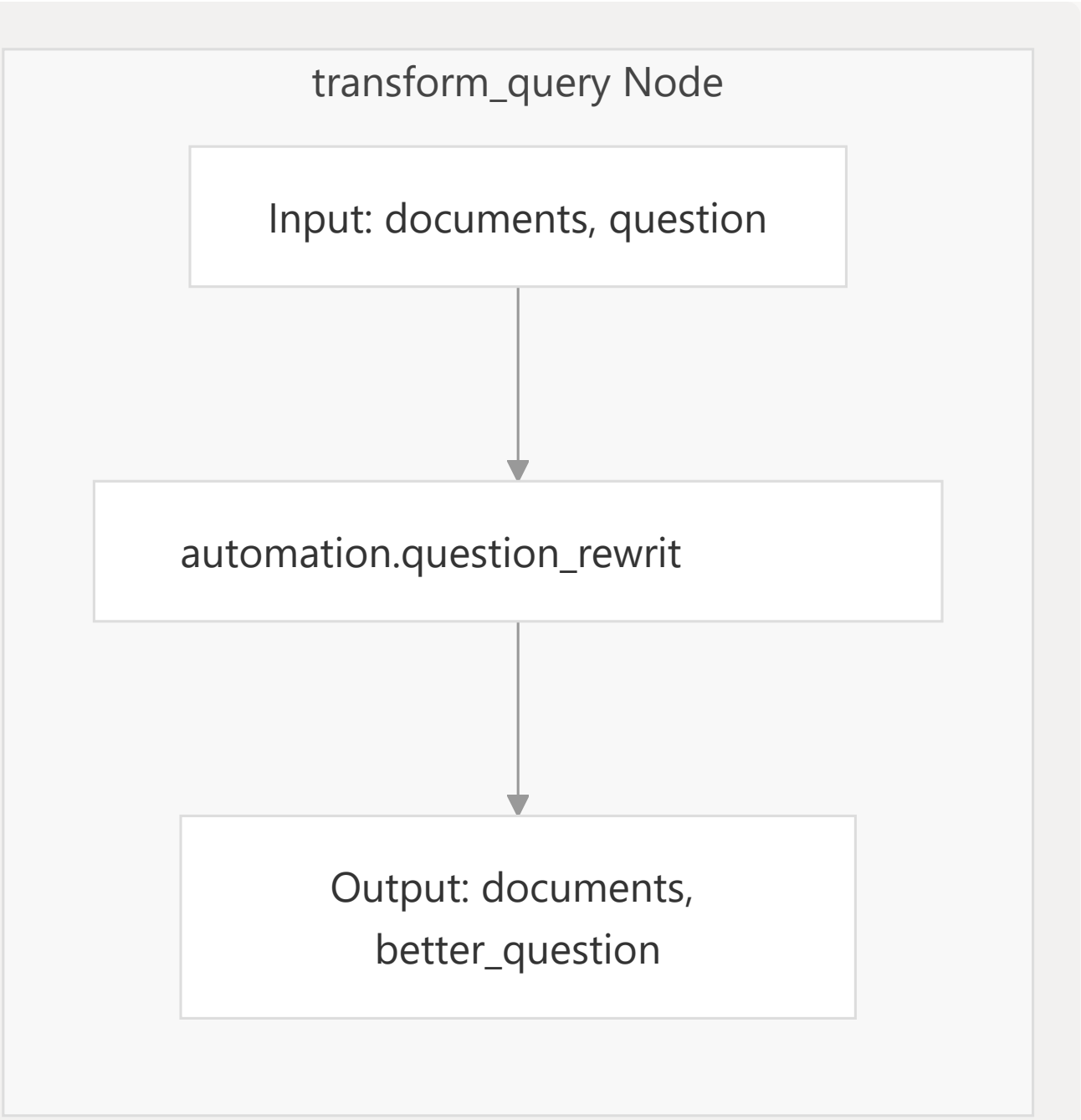
The `generate` node creates a response using the filtered documents and the original question:



Sources: `community/log_analysis_multi_agent_rag/graphnodes.py` `33-40`

4.5 Query Transformation

The `transform_query` node rewrites the query to improve retrieval when necessary:



Sources: `community/log_analysis_multi_agent_rag/graphnodes.py` `61-70`

5. Edge Decision Logic

The system uses conditional edges to determine the flow between nodes based on the current state:

5.1 Document Relevance Decision

The `decide_to_generate` edge function decides whether to generate an answer or transform the query: