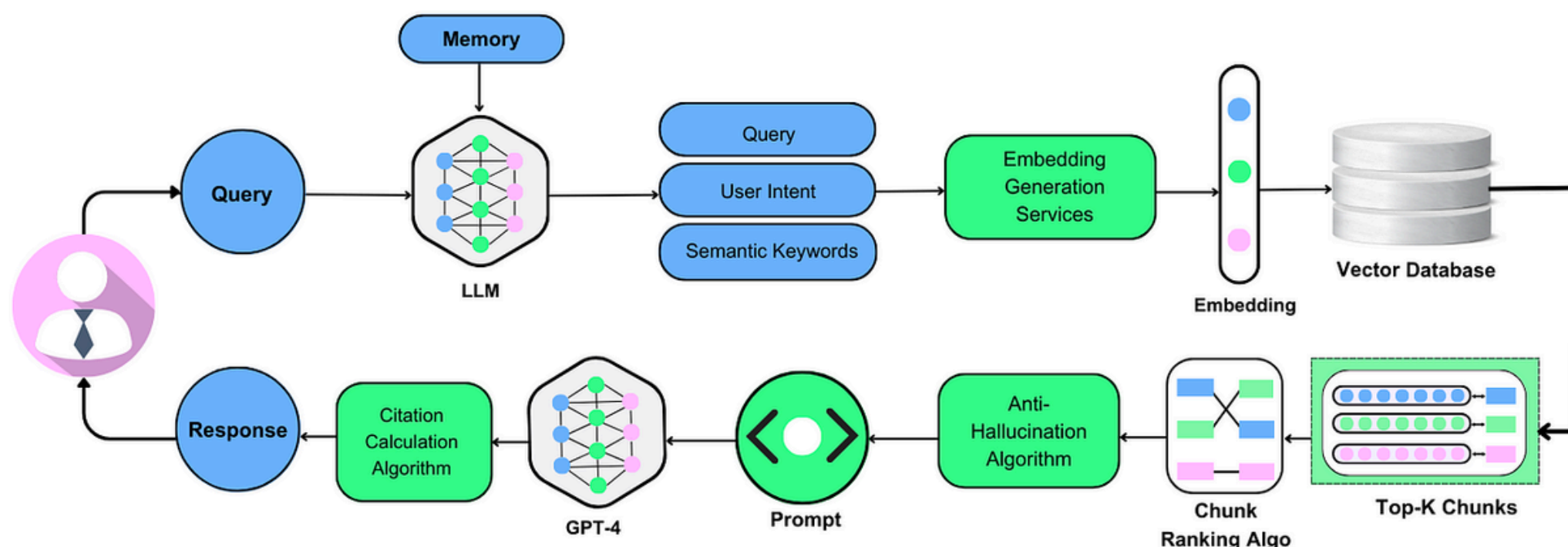


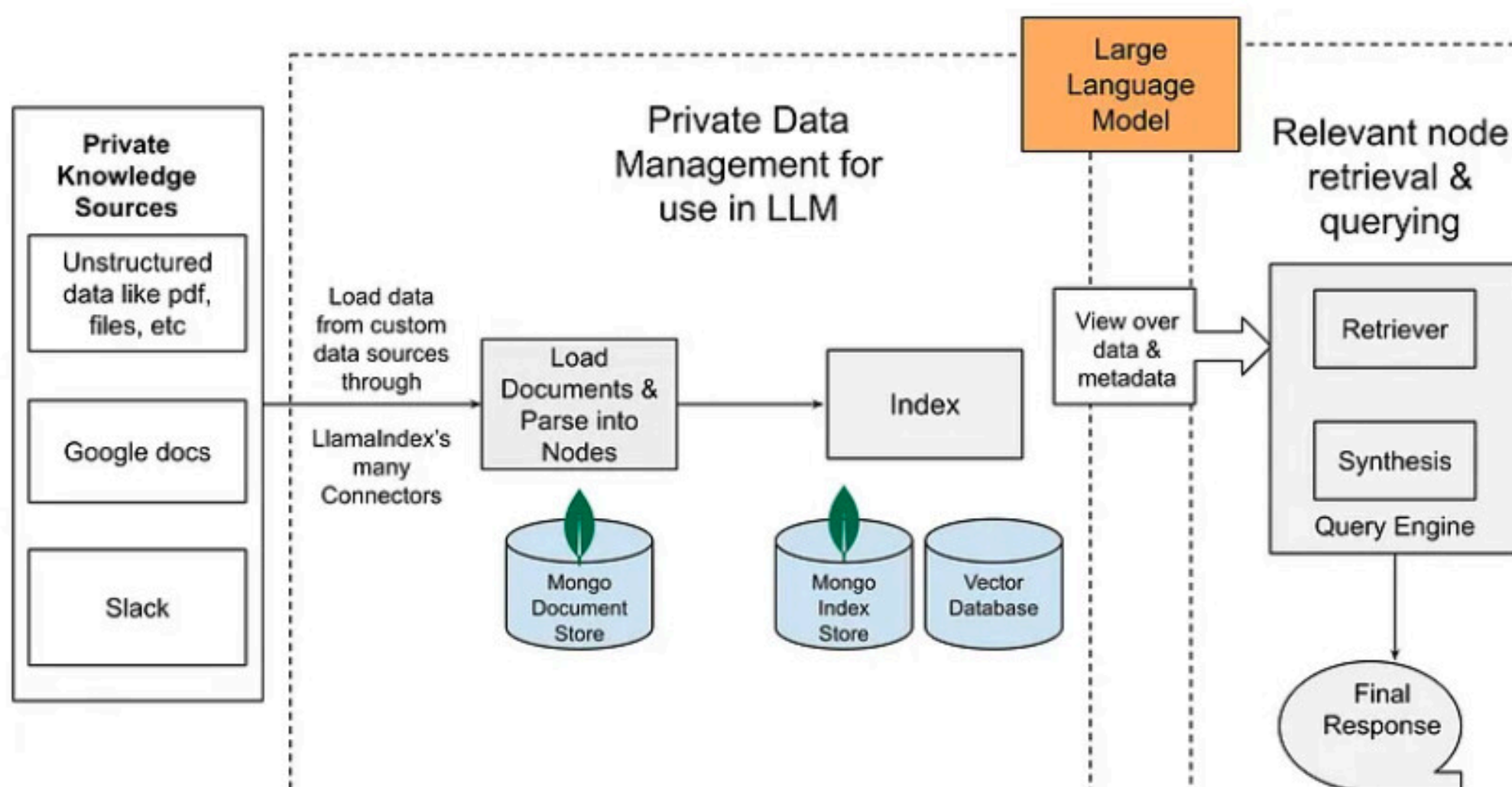
Mastering RAG

Mitigating Hallucinations in RAG

Controlling Hallucinations with Retrieval Augmented Generation (RAG)



Improving response quality with query pre-processing, semantic expansion, chunk ranking, anti-hallucinations and citations.



Why Do Hallucinations Occur in RAG?

Hallucinations in RAG often stem from:

1. Poor Document Retrieval

- The retriever fetches irrelevant or low-quality documents.
- The context window includes non-informative or misleading passages.

2. LLM Overgeneralization

- The language model attempts to "fill in the gaps" when relevant information is missing.
- The model generates content based on learned patterns, not actual facts.

3. Lack of Source Verification

- RAG pipelines often do not verify the credibility of retrieved information.
- This leads to inclusion of outdated or incorrect knowledge.

4. Context Window Limitations

- The retriever might fetch too many or too few documents.
- LLMs truncate or misinterpret the retrieved context.



Techniques to Mitigate Hallucinations

1. Optimizing the Retriever for High-Quality Context

- ✓ Use Hybrid Search – Combine semantic retrieval (vector search) with keyword-based retrieval to ensure relevance.
- ✓ Diversity-aware Retrieval – Include multiple perspectives from various sources.
- ✓ Document Re-ranking – Score retrieved documents based on relevance.

2. Improving Prompt Engineering

- ✓ Explicit Constraints – Guide the model with prompts like: "Answer only based on the given documents. If unsure, say 'I don't know.'"
- ✓ Chain-of-Thought (CoT) Prompting – Force the LLM to break down reasoning.
- ✓ Reference-Only Mode – Ask the model to cite retrieved documents for verification



3. Implementing Source Verification & Filtering

- ✓ Metadata Filtering – Use document timestamps, authorship, and credibility scores.
- ✓ Multi-Step Fact-Checking – Cross-verify retrieved documents before generation.
- ✓ Threshold-based Filtering – Reject documents with low similarity scores.

4. Context Length Optimization

- ✓ Chunking Strategies – Instead of full documents, retrieve smaller, high-relevance passages.
- ✓ Dynamic Context Adaptation – Adjust context size dynamically to balance relevance vs. token limits.

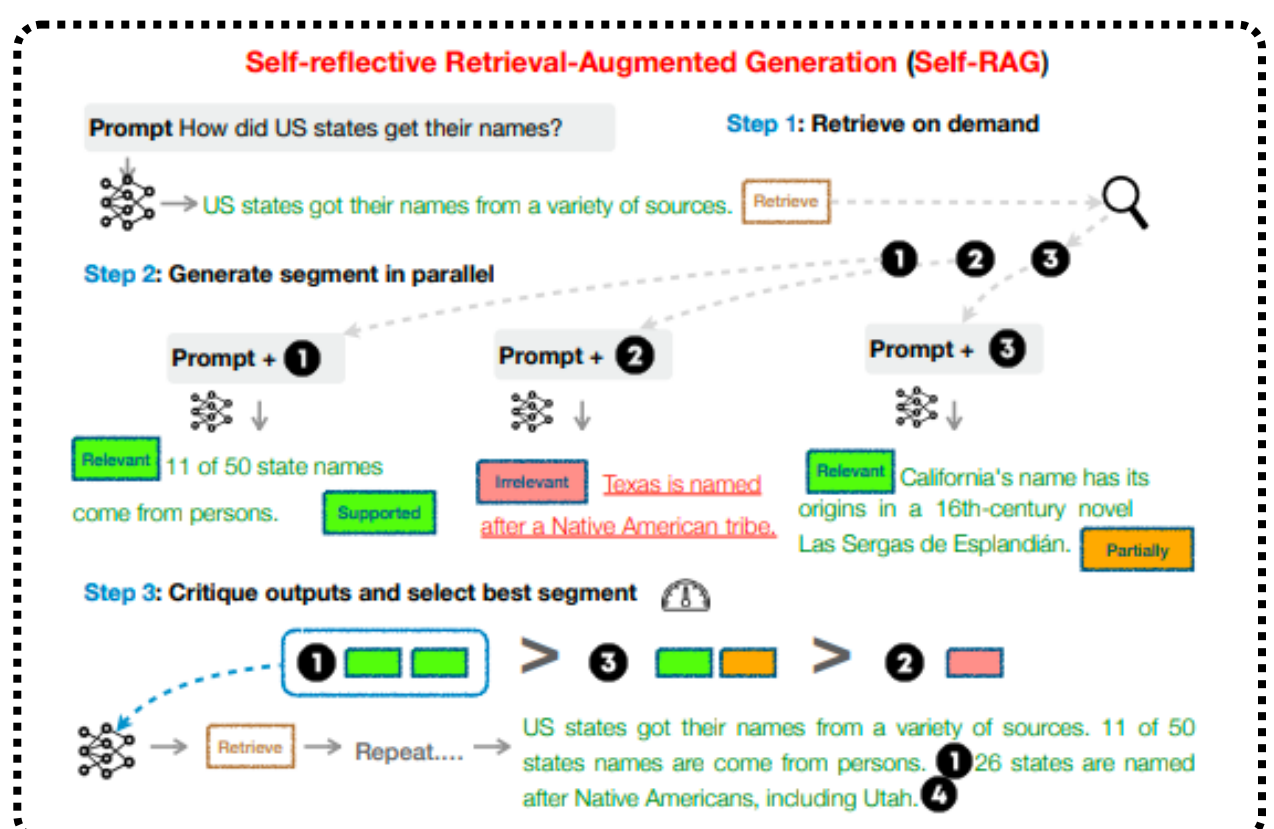
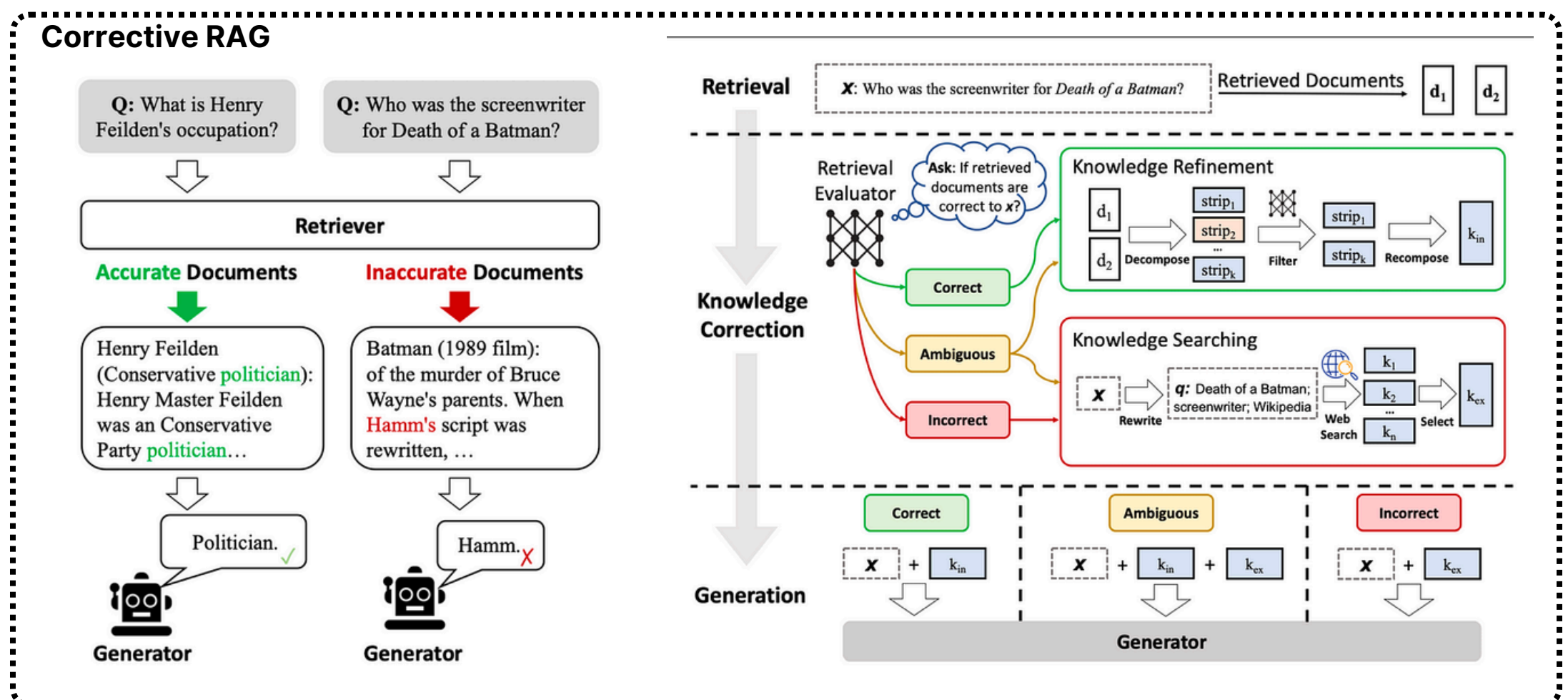
5. Post-Processing & Human-in-the-Loop Feedback

- ✓ Automatic Response Validation – Implement models that score generated responses for factual accuracy.
- ✓ Human-in-the-Loop Review – Allow domain experts to validate and fine-tune responses.
- ✓ Reinforcement Learning from Human Feedback (RLHF) – Use human preferences to improve future generations.



6. Using a powerful LLM like GPT-4, Gemini 2.0, Llama 3.3, Claude 3.7 or even reasoning LLMs like DeepSeek to prevent hallucinated answers.

7. Using Agentic RAG workflows like Agentic Corrective RAG, Self-Reflective RAG to fact-check generated response, answer relevancy and context relevancy.




Hands-On Python Implementation

Step 1: Install Required Libraries



```
pip install transformers faiss-cpu langchain openai
```

Step 2: Implementing a Hybrid Retriever



```
from langchain.vectorstores import FAISS
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.document_loaders import TextLoader

# Load documents
doc_loader = TextLoader("knowledge_base.txt")
docs = doc_loader.load()

# Convert documents into embeddings
embedding_model = OpenAIEmbeddings()
vector_db = FAISS.from_documents(docs, embedding_model)

def hybrid_search(query, top_k=3):
    """Combines keyword search with semantic retrieval"""
    semantic_results = vector_db.similarity_search(query, k=top_k)
    keyword_results = [doc for doc in docs if query.lower() in doc.page_content.lower()]
    return list(set(semantic_results + keyword_results))
```



Step 3: Implementing Context Filtering & Ranking

```
from langchain.llms import OpenAI

llm = OpenAI(model_name="gpt-4")

def generate_response(query):
    retrieved_docs = hybrid_search(query)
    ranked_docs = sorted(retrieved_docs, key=lambda x: x.metadata.get("score", 0), reverse=True)
    context = "\n".join([doc.page_content for doc in ranked_docs[:2]]) # Select top 2 docs

    prompt = f"""
    Given the following context:
    {context}
    Answer the question: {query}
    If the answer is not found, say 'I don't know.'"""

    response = llm(prompt)
    return response
```

Step 4: Implementing a Fact-Checking Model

```
from transformers import pipeline

fact_checker = pipeline("text-classification", model="facebook/bart-large-mnli")

def verify_response(response, context):
    """Validates if the generated response aligns with retrieved documents."""
    results = fact_checker([{"text": response, "context": context}])
    return results[0]['score'] > 0.8 # Threshold-based validation
```



Step 5: Testing the Pipeline



```
query = "What is FlashRAG?"
gen_response = generate_response(query)
retrieved_docs = hybrid_search(query)
validity = verify_response(gen_response, retrieved_docs)

print("Generated Response:", gen_response)
print("Is the response factually valid?", validity)
```