6 RAG Enhancement Techniques

Overview

This notebook demonstrates **6 RAG Enhancement Techniques** (plus a baseline) that improve upon basic Retrieval-Augmented Generation (RAG) systems. These techniques address common bottlenecks in traditional RAG pipelines including poor query formulation, irrelevant retrieval, suboptimal indexing, and lack of response validation.

Important Note: These are **non-agentic implementations** using LangChain LCEL (LangChain Expression Language). They represent advanced RAG patterns that enhance retrieval quality without requiring complex agent workflows or LangGraph state machines.

Table of Contents

---

1. Basic RAG (Baseline)

What is it?

The foundational RAG pattern that serves as our baseline for comparison.

How it works:

```
User Query → Retrieve Documents → Generate Answer
```

What we'll build:

- Simple vector store retrieval using FAISS
- Basic embeddings with Google Generative AI
- Straightforward generate-from-context approach

Use case:

- Simple Q&A over documents
- Proof-of-concept implementations
- When query and document structures are well-aligned

Limitations addressed by other techniques:

- ❌ Poor query formulation
- ❌ Single retrieval strategy
- ❌ No relevance filtering
- ❌ No response validation

---

2. Query Expansion

What is it?

A technique that transforms a single user query into multiple query variations to improve retrieval coverage and capture different aspects of the information need.

How it works:

```
User Query → Generate Multiple Query Variations → Retrieve for Each → Combine Results → Generate Answer
```

What we'll build:

- Multi-query generation using LLM
- Parallel retrieval for each query variation
- Result aggregation and deduplication
- Comprehensive answer generation from combined results

Use cases:

- **Broad and diverse context** - When the knowledge base covers wide-ranging topics
- **Ambiguous queries** - When the user's intent could be interpreted multiple ways
- **Comprehensive coverage** - When you want to ensure all relevant documents are retrieved
- **Different perspectives** - When you need to capture multiple viewpoints on a topic

Example:

Original Query: "How does attention work in transformers?"

Expanded Queries:

1. "What is the attention mechanism in transformer models?"
2. "Explain self-attention in neural networks"
3. "How do transformers use attention for sequence processing?"
4. "What are the key components of transformer attention?"

Benefits:

- ✅ Increased retrieval coverage
- ✅ Captures different phrasings and perspectives
- ✅ Reduces risk of missing relevant documents
- ✅ Better handles ambiguous queries

Trade-offs:

- ⚠️ Increased latency (multiple retrievals)
- ⚠️ Potential for redundant results
- ⚠️ Higher computational cost

---

3. Hypothetical Document Embeddings (HyDE)

What is it?

Instead of searching with the query directly, HyDE generates a hypothetical answer first, then uses that answer's embedding to search for similar documents.

How it works:

```
User Query → Generate Hypothetical Answer → Embed Answer → Search with Answer Embedding → Retrieve Real Documents → Generate Final Answer
```

What we'll build:

- Hypothetical document generation using LLM
- Embedding-based similarity search using the generated document
- Final answer generation from retrieved real documents

The Key Insight:

Sometimes the answer we're looking for is more similar to other answers than the question is similar to other questions. By searching with a hypothetical answer, we find documents that contain answer-like content.

Use cases:

- **Technical Q&A** - When questions and answers use different vocabulary
- **Complex reasoning** - When the answer requires synthesis of information
- **Domain-specific queries** - When expert answers use specialized terminology
- **When queries are vague** - Generate a concrete hypothetical answer to search with

Example:

Query: "Why do neural networks need activation functions?"

Hypothetical Answer Generated: "Neural networks need activation functions to introduce non-linearity into the model. Without activation functions, multiple layers would collapse into a single linear transformation, limiting the network's ability to learn complex patterns..."

Search: Use this hypothetical answer's embedding to find similar real documents

Benefits:

- ✅ Bridges vocabulary gap between questions and answers
- ✅ Finds documents with answer-like content
- ✅ Works well for complex reasoning queries
- ✅ Leverages LLM's knowledge to improve retrieval

Trade-offs:

- ⚠️ Extra LLM call adds latency
- ⚠️ Hypothetical answer might lead search astray if incorrect
- ⚠️ Works best when LLM has relevant knowledge

---

4. Contextual Compression

What is it?

A technique that extracts only the most relevant portions of retrieved documents, removing irrelevant content before passing to the LLM for answer generation.

How it works:

```
User Query → Retrieve Documents → Extract Relevant Portions Only → Discard Irrelevant Parts → Generate Answer
```

What we'll build:

- Document compression using LLM or extractive methods
- Relevance-based filtering
- Token-efficient context creation
- Preservation of key information while reducing noise

The Problem it Solves:

Retrieved documents often contain large amounts of irrelevant information. Passing entire documents to the LLM:

- Wastes tokens
- Adds noise that can confuse the model
- Increases latency and cost
- May exceed context window limits

Use cases:

- **Long documents** - When retrieved documents are too lengthy

- **Token limitations** - When approaching context window limits
- **Cost optimization** - Reducing tokens processed by expensive LLMs
- **Noisy retrieval** - When documents contain mixed relevant/irrelevant content
- **Multi-document synthesis** - Combining information from many sources

Example:

Retrieved Document (1000 tokens): "Introduction... Background... [500 tokens of context]... The attention mechanism allows models to focus on relevant parts of the input... [specific explanation]... Future work... Conclusion..."

Compressed (200 tokens): "The attention mechanism allows models to focus on relevant parts of the input by computing weighted sums based on learned similarity scores..."

Benefits:

- ✅ Reduces token usage and costs
- ✅ Removes irrelevant information
- ✅ Stays within context limits
- ✅ Improves answer quality by reducing noise
- ✅ Faster processing with smaller context

Trade-offs:

- ⚠️ Risk of removing important context
- ⚠️ Additional processing step adds some latency
- ⚠️ Requires careful tuning to balance compression vs information loss

---

5. Maximum Marginal Relevance (MMR)

What is it?

A retrieval strategy that balances relevance and diversity when selecting documents, avoiding redundant information by considering both similarity to the query AND dissimilarity to already-selected documents.

How it works:

```
User Query → Initial Retrieval → Rank by: (Relevance to Query) - λ × (Similarity to Selected Docs) → Return Diverse Set
```

What we'll build:

- MMR scoring algorithm
- Diversity-aware document selection
- Configurable relevance vs. diversity trade-off (λ parameter)
- Iterative document selection process

The Problem it Solves:

Standard similarity search often returns many nearly-identical documents. This leads to:

- Redundant information in the context
- Wasted token budget on repetitive content
- Missing diverse perspectives
- Reduced answer quality

The MMR Formula:

```
MMR = argmax[λ × Sim(Di, Query) - (1-λ) × max(Sim(Di, Dj))]
where:
- Di = candidate document
- Dj = already selected documents
- λ = relevance vs diversity trade-off (0 to 1)
```

Use cases:

- **Many similar results** - When search returns too many redundant documents
- **Comprehensive answers** - Need to cover multiple aspects of a topic
- **Diverse perspectives** - Want different viewpoints or approaches
- **Exploratory search** - User wants to see breadth of information
- **Summary generation** - Covering multiple angles of a topic

Example:

Query: "What are transformer architectures?"

Without MMR (top 3):

1. "Transformer architecture uses self-attention..."
2. "The transformer model uses self-attention..."
3. "Transformers rely on self-attention mechanisms..."

(All saying similar things!)

With MMR (λ=0.5):

1. "Transformer architecture uses self-attention..."
2. "BERT is a bidirectional transformer for NLP..."
3. "GPT uses transformer decoders for generation..."

(More diverse coverage!)

Benefits:

- ✅ Reduces redundancy in retrieved documents
- ✅ Provides diverse information coverage
- ✅ Better use of token budget
- ✅ Improves answer comprehensiveness
- ✅ Exposes different perspectives

Parameters:

- **λ = 1.0**: Pure relevance (like standard retrieval)
- **λ = 0.5**: Balanced relevance and diversity
- **λ = 0.0**: Pure diversity (may lose relevance)

Trade-offs:

- ⚠️ May sacrifice some relevance for diversity
- ⚠️ Slower than simple top-k retrieval
- ⚠️ Requires tuning λ parameter for your use case

---

6. Source Attribution

What is it?

A technique that tracks and provides information about the source of each piece of information in the generated answer, including specific paragraphs, page numbers, or document references.

How it works:

```
User Query → Retrieve Documents → Generate Answer → Link Answer Parts to Source Documents → Present with Citations
```

What we'll build:

- Source tracking during answer generation
- Citation formatting
- Document/paragraph/page reference system
- Linkage between answer claims and source evidence

The Problem it Solves:

Without source attribution:

- Users can't verify information
- Hard to trust AI-generated answers
- Difficult to find more details
- No accountability for claims
- Users can't distinguish facts from hallucinations

Use cases:

- **Sensitive topics** - Medical, legal, financial information
- **High-stakes decisions** - When accuracy is critical
- **Trust requirements** - When users need to verify claims
- **Regulatory compliance** - When traceability is required
- **Academic/research** - Proper citation is essential
- **Fact-checking** - Need to trace back to original sources

Example Output:

Query: "What are the side effects of aspirin?"

Answer with Attribution: "Aspirin can cause several side effects. Common side effects include stomach upset and heartburn [Source: Medical Guide, Page 42]. More serious side effects include bleeding and allergic reactions [Source: FDA Drug Sheet, Section 3.2]. Long-term use may increase the risk of stomach ulcers [Source: Clinical Study 2023, Paragraph 5]."

vs Without Attribution: "Aspirin can cause stomach upset, heartburn, bleeding, and allergic reactions..." *(How do I verify this? Where did this come from?)*

Implementation Approaches:

6.1 Inline Citations

- Reference documents inline: "[Source: Document A]"
- Simple and clear
- Easy to implement

6.2 Paragraph-level Attribution

- Track which paragraphs support each claim
- More granular than document-level
- Helps users find exact information

6.3 Page Numbers

- For PDF/book sources
- "As stated on page 42..."
- Familiar citation format

6.4 Structured Metadata

```
{
  "answer": "The answer text...",
  "sources": [
    {
      "claim": "Specific claim",
      "source": "Document name",
      "page": 42,
      "confidence": 0.95
    }
  ]
}
```

Benefits:

- ✅ Increases trust and credibility
- ✅ Enables verification

- ✅ Helps users find more information
- ✅ Reduces hallucination risk (model knows it must cite)
- ✅ Provides accountability
- ✅ Meets regulatory requirements
- ✅ Better user experience for research

Trade-offs:

- ⚠️ Slightly more complex to implement
- ⚠️ May make answers more verbose
- ⚠️ Requires careful prompt engineering
- ⚠️ Need to handle cases where sources disagree

---

7. Self-consistency Checking

What is it?

A technique that generates multiple independent answers to the same query, then selects the most consistent or frequently occurring answer as the final output. This leverages the wisdom of multiple model runs to improve reliability.

How it works:

```
User Query → Generate Multiple Answers (N times) → Analyze Consistency → Select Most Frequent/Consistent → Return Final Answer
```

What we'll build:

- Multiple answer generation with temperature sampling
- Consistency analysis across answers
- Majority voting or semantic clustering
- Confidence scoring based on agreement

The Problem it Solves:

Single LLM generations can be:

- Inconsistent or unstable
- Prone to hallucinations
- Affected by random sampling
- Less reliable for critical applications

By generating multiple answers and checking consistency, we can:

- Identify the most reliable information
- Detect when the model is uncertain
- Filter out random hallucinations
- Increase confidence in the output

Use cases:

- **Sensitive topics** - Medical diagnoses, legal advice, financial decisions
- **High trust requirements** - When accuracy is paramount
- **Critical applications** - Safety-critical systems
- **Fact verification** - Checking controversial or disputed claims
- **Quality assurance** - Enterprise applications requiring reliability
- **Complex reasoning** - Multi-step problems where errors compound

Example:

Query: "What is the capital of Australia?"

Generation 1: "The capital of Australia is Canberra..." **Generation 2:** "Canberra is the capital city of Australia..." **Generation 3:** "Australia's capital is Canberra..." **Generation 4:** "The capital is Canberra, located in the ACT..." **Generation 5:** "Canberra serves

as Australia's capital..."

Consistency Analysis: 5/5 answers agree on "Canberra" **Confidence:** Very High ✓ **Final Answer:** "The capital of Australia is Canberra"

Comparison Example (Low Consistency):

Query: "Will quantum computing break current encryption?"

Generation 1: "Yes, quantum computers will break RSA encryption..." **Generation 2:** "Current encryption is safe for now, but vulnerable in the future..." **Generation 3:** "Only certain types of encryption are vulnerable..." **Generation 4:** "Post-quantum cryptography is being developed..." **Generation 5:** "It depends on when large-scale quantum computers are built..."

Consistency Analysis: Answers vary significantly **Confidence:** Low ⚠ **Action:** Flag as uncertain, present multiple perspectives, or request more specific query

Implementation Approaches:

7.1 Majority Voting

- Generate N answers
- Extract key facts from each
- Select most frequent answer
- Simple but effective

7.2 Semantic Clustering

- Generate N answers
- Embed each answer
- Cluster similar answers
- Select largest cluster's representative

7.3 Weighted Consistency

- Consider confidence scores
- Weight by retrieval quality
- Penalize outliers
- More sophisticated scoring

7.4 Hybrid Approach

```
1. Generate N answers (typically 3-7)
2. Extract key claims from each
3. Calculate agreement score
4. If high agreement (>70%): Return consensus answer
5. If medium agreement (40-70%): Synthesize common elements
6. If low agreement (<40%): Flag as uncertain, show diversity
```

Benefits:

- ✅ Significantly increased reliability
- ✅ Detection of model uncertainty
- ✅ Reduced hallucinations
- ✅ Better error detection
- ✅ Quantifiable confidence scores
- ✅ Suitable for high-stakes applications

Trade-offs:

- ⚠ **Higher latency** - Need N generations (3-7x slower)
- ⚠ **Increased cost** - Multiple LLM calls
- ⚠ **More complex** - Consistency analysis required
- ⚠ **Token intensive** - Uses more API quota

When to Use:

- **Always use** for: Medical, legal, financial applications

- **Strongly recommended** for: High-stakes decisions, safety-critical systems
- **Consider** for: Important business decisions, content requiring high accuracy
- **Skip** for: Casual queries, exploratory questions, non-critical applications

Parameters:

- **N (number of generations)**: Typically 3-7
  - N=3: Minimum for consistency checking
  - N=5: Good balance of reliability vs cost
  - N=7+: Maximum reliability, high cost
- **Temperature**: Use 0.3-0.7 for diversity without wild variation
- **Consistency Threshold**: 60-80% agreement for high confidence

---

Technology Stack

Core Libraries:

- **LangChain**: RAG pipeline orchestration
- **LangChain Community**: Additional integrations
- **FAISS**: Vector similarity search
- **Google Generative AI**: Embeddings and LLM (Gemini 2.0 Flash)

Key Components:

- `ChatGoogleGenerativeAI` : LLM for generation
- `GoogleGenerativeAIEmbeddings` : Text embeddings
- `FAISS` : Vector store
- `ChatPromptTemplate` : Prompt engineering
- `StrOutputParser` : Output parsing
- LCEL (LangChain Expression Language): Chain composition

Implementation Approach

Each technique is implemented as:

1. **Standalone function/class** - Can be used independently
2. **Clear examples** - Demonstrates usage with sample data
3. **Comparison to baseline** - Shows improvement over basic RAG
4. **Composable design** - Can combine multiple techniques

Chain Composition Example:

```
enhanced_rag = (
    query_rewriter          # Technique 2: Query Transformation
    | router                # Technique 3: Routing
    | retriever             # Enhanced with Technique 5: Better Indexing
    | reranker              # Technique 6: Post-Retrieval
    | generator
    | verifier              # Technique 6: Response Verification
)
```

When to Use Which Technique?

Technique	Use When...	Complexity	Impact
Basic RAG	Simple Q&A, proof-of-concept	Low	Baseline
Query Expansion	Broad/diverse context, need multiple query versions	Medium	High
Hypothetical Document Embeddings (HyDE)	Want to search by hypothetical answer, not query	Medium-High	High
Contextual Compression	Documents too long, token limits	Medium	High
Maximum Marginal Relevance (MMR)	Too many similar results, need diverse information	Medium	Medium-High

Technique	Use When...	Complexity	Impact
Source Attribution	Sensitive topics, high trust/accuracy requirements	Low	Medium
Self-consistency Checking	Sensitive topics, high trust requirements	Medium-High	High

Key Takeaways

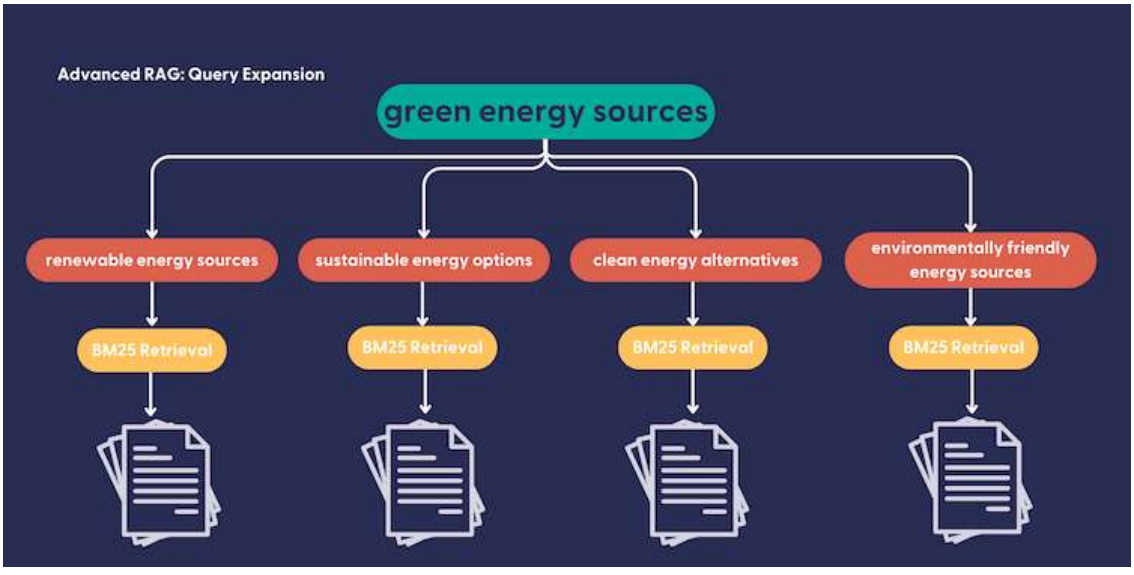
1. **These are enhancements, not replacements** - Start with basic RAG, add techniques as needed
2. **Linear pipelines, not agents** - No feedback loops or state machines (that's where LangGraph comes in)
3. **Composable design** - Mix and match techniques based on your use case
4. **Production considerations:**
   - Query Transformation: Adds latency but improves quality
   - Routing: Reduces irrelevant retrieval
   - Advanced Indexing: Higher preprocessing cost, better retrieval
   - Post-Retrieval: Adds final quality check
5. **Next steps to Agentic RAG:**
   - Add conditional logic (if retrieval fails, try different strategy)
   - Implement feedback loops (self-correction)
   - Use LangGraph for complex workflows
   - Add tool orchestration and state management

Make sure you load the API keys for cloud providers!

You can set your environment keys yourself or use a script. Please note that since keys are private, they are not included in the repository.

In [ ]:
```
pip install langchain-google-genai langchain-community langchain-core langchain jq faiss-cpu
```

Query Expansion

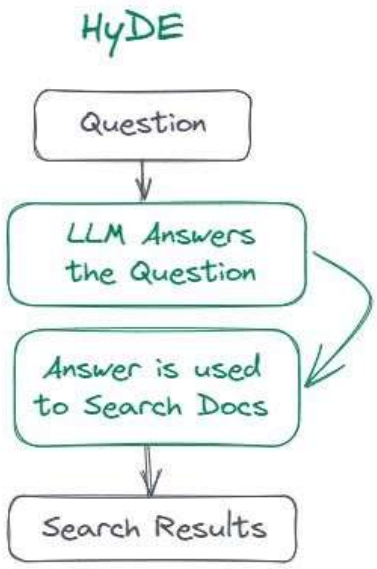


In [ ]:
```
from langchain.prompts import PromptTemplate
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.output_parsers import StrOutputParser

# Define the prompt template
expansion_template = """Given the user question: {question}
Generate three alternative versions that express the same information need but with different wording:
1."""

expansion_prompt = PromptTemplate(
    input_variables=["question"],
```

```
    template=expansion_template
)

# Use Google's Gemini model via LangChain
llm = ChatGoogleGenerativeAI(
    model="gemini-2.0-flash",
    temperature=0.7,
    google_api_key="AIzaSyAMAYxkjP49QZRCg21zImWWAu7c3YHJ0a8"  # ✅ Pass it directly here
)

# Create the chain
expansion_chain = expansion_prompt | llm | StrOutputParser()

# Generate expanded queries
original_query = "What are the effects of climate change?"
expanded_queries = expansion_chain.invoke({"question": original_query})

print(expanded_queries)
```

```
Here are three alternative versions of the question "What are the effects of climate change?":

1. **How does climate change impact the world?** (This version is slightly broader and focuses on the impact in g
eneral.)
2. **What are the consequences of a changing climate?** (This version uses more formal language and emphasizes th
e results of the change.)
3. **What happens as a result of climate change?** (This version is more conversational and focuses on the direct
outcomes.)
```

Hypothetical Document Embeddings (HyDE)



In [ ]:

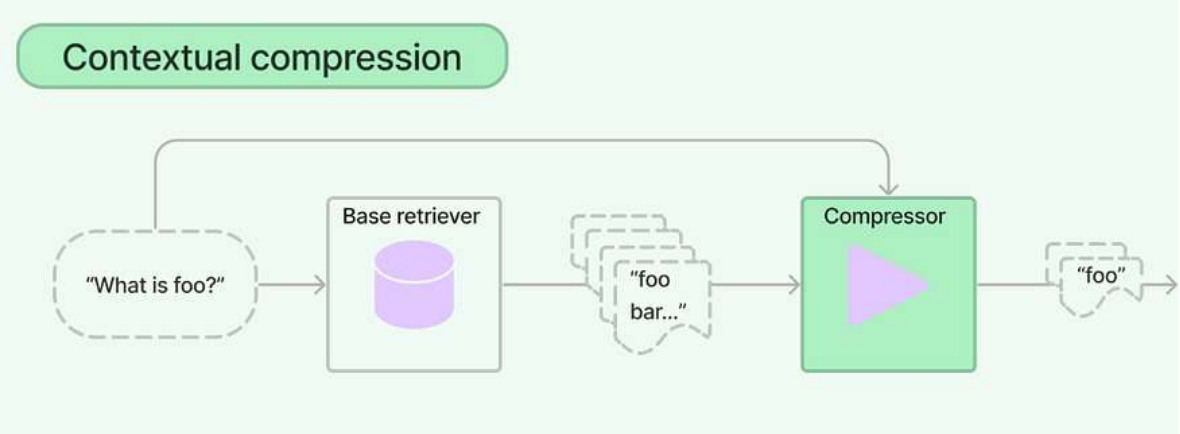
```
import json

knowledge_base = [
    {
        "content": "Dietary changes that reduce carbon footprint include eating more plant-based foods, reducing
    },
    {
        "content": "Reducing red meat intake can significantly lower greenhouse gas emissions because livestock f
    },
    {
        "content": "Increasing consumption of seasonal fruits and vegetables can decrease the environmental impac
    },
    {
        "content": "Minimizing food waste through meal planning and proper storage helps reduce the carbon footpr
    },
    {
        "content": "Switching to organic farming products can reduce synthetic fertilizer use, which lowers nitro
    }
]

with open("knowledge_base.json", "w") as f:
    json.dump(knowledge_base, f, indent=2)

print("knowledge_base.json saved successfully!")
```

```
knowledge_base.json saved successfully!
```

In [ ]:
```python
from langchain_google_genai import ChatGoogleGenerativeAI, GoogleGenerativeAIEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_community.document_loaders import JSONLoader
from langchain.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

# Load documents from JSON file
loader = JSONLoader(
    file_path="knowledge_base.json",
    jq_schema=".[].content",
    text_content=True
)
documents = loader.load()

# Initialize Google Generative AI embeddings
embedder = GoogleGenerativeAIEmbeddings(
    model="models/embedding-001",
    google_api_key="AIzaSyAMAYxkjP49QZRCg21zImWWAu7c3YHJ0a8"
)

# Create FAISS vector store from documents using Google embeddings
vector_db = FAISS.from_documents(documents, embedder)

# Prompt template for generating hypothetical document (Hyde method)
hyde_template = """Based on the question: {question}
Write a passage that could contain the answer to this question:"""

hyde_prompt = PromptTemplate(
    input_variables=["question"],
    template=hyde_template
)

# Initialize Google Gemini chat model
llm = ChatGoogleGenerativeAI(
    model="gemini-2.0-flash",
    temperature=0.7,
    google_api_key="AIzaSyAMAYxkjP49QZRCg21zImWWAu7c3YHJ0a8"  # ✅ Pass it directly here
)

hyde_chain = hyde_prompt | llm | StrOutputParser()

# Generate hypothetical document
query = "What dietary changes can reduce carbon footprint?"
hypothetical_doc = hyde_chain.invoke({"question": query})

# Embed the hypothetical document for similarity search
embedded_query = embedder.embed_query(hypothetical_doc)
results = vector_db.similarity_search_by_vector(embedded_query, k=3)

print(results)
```

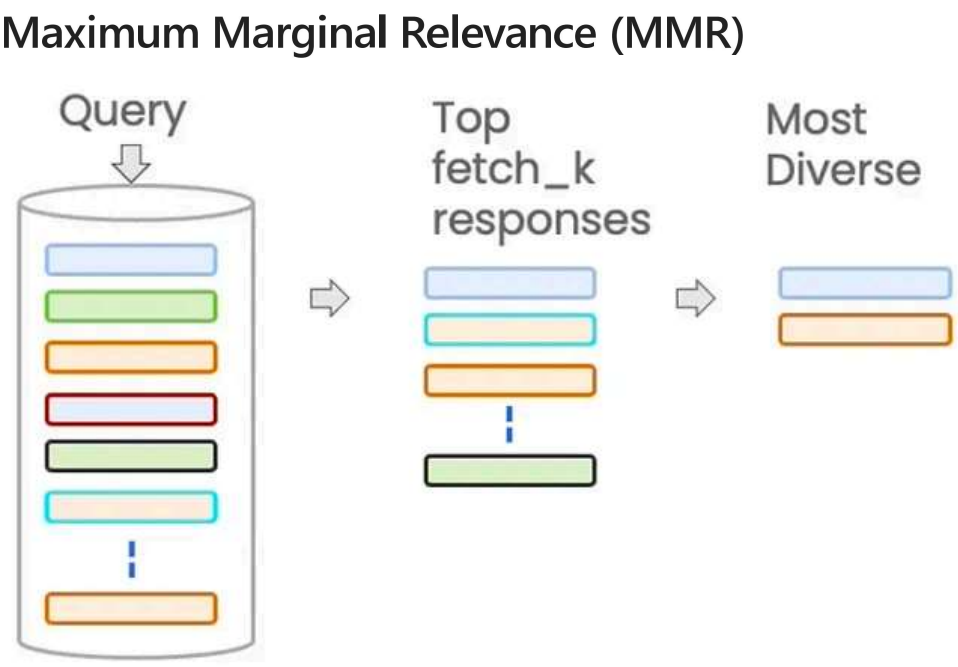
```
[Document(id='c2eb5523-6577-43f9-a212-8fda7d17b2a2', metadata={'source': '/content/knowledge_base.json', 'seq_num': 1}, page_content='Dietary changes that reduce carbon footprint include eating more plant-based foods, reducing meat and dairy consumption, and choosing locally sourced produce to minimize transportation emissions.'), Document(id='25fab36e-190e-4413-95ab-7586f053bae6', metadata={'source': '/content/knowledge_base.json', 'seq_num': 2}, page_content='Reducing red meat intake can significantly lower greenhouse gas emissions because livestock farming produces methane, a potent greenhouse gas.'), Document(id='2f8425e9-f496-438b-ac77-9f93d2868dbf', metadata={'source': '/content/knowledge_base.json', 'seq_num': 4}, page_content='Minimizing food waste through meal planning and proper storage helps reduce the carbon footprint associated with producing, transporting, and disposing of food.')]
```

Contextual Compression



In [ ]:
```python
from langchain.retrievers.document_compressors import LLMChainExtractor
from langchain.retrievers import ContextualCompressionRetriever
from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(
    model="gemini-2.0-flash",
    temperature=0.7,
    google_api_key="AIzaSyAMAYxkjP49QZRCg21zImWWAu7c3YHJ0a8"  # ✅ Pass it directly here
)
compressor = LLMChainExtractor.from_llm(llm)

# Create a basic retriever from the vector store
base_retriever = vector_db.as_retriever(search_kwargs={"k": 3})

compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=base_retriever
)
compressed_docs = compression_retriever.invoke("What dietary changes can reduce carbon footprint?")

print(compressed_docs)
```

```
[Document(metadata={'source': '/content/knowledge_base.json', 'seq_num': 1}, page_content='Dietary changes that r
educe carbon footprint include eating more plant-based foods, reducing meat and dairy consumption, and choosing l
ocally sourced produce to minimize transportation emissions.'), Document(metadata={'source': '/content/knowledge_
base.json', 'seq_num': 2}, page_content='Reducing red meat intake can significantly lower greenhouse gas emission
s because livestock farming produces methane, a potent greenhouse gas.'), Document(metadata={'source': '/content/
knowledge_base.json', 'seq_num': 4}, page_content='Minimizing food waste through meal planning and proper storage
helps reduce the carbon footprint associated with producing, transporting, and disposing of food.')]
```

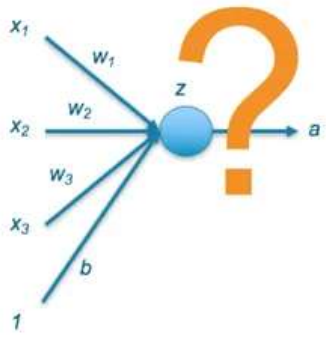


In [ ]:
```python
from langchain_community.vectorstores import FAISS

embeddings = GoogleGenerativeAIEmbeddings(
```

```
    model="models/embedding-001",
    google_api_key="AIzaSyAMAYxkjP49QZRCg21zImWWAu7c3YHJ0a8"
)

vector_store = FAISS.from_documents(documents, embeddings)

mmr_results = vector_db.max_marginal_relevance_search(
    query="What are transformer models?",
    k=5, # Number of documents to return
    fetch_k=20, # Number of documents to initially fetch
    lambda_mult=0.5 # Diversity parameter (0 = max diversity, 1 = max relevance)
)
print(mmr_results)
```

```
[Document(id='3d26e368-4fe5-450a-9b4b-743b49885842', metadata={'source': '/content/knowledge_base.json', 'seq_nu
m': 5}, page_content='Switching to organic farming products can reduce synthetic fertilizer use, which lowers nit
rous oxide emissions and improves soil health.'), Document(id='8c76e370-a50c-41b5-92bc-c278db36aea8', metadata=
{'source': '/content/knowledge_base.json', 'seq_num': 4}, page_content='Minimizing food waste through meal planni
ng and proper storage helps reduce the carbon footprint associated with producing, transporting, and disposing of
food.'), Document(id='1c20576c-3e6c-4fc5-9762-24030360b696', metadata={'source': '/content/knowledge_base.json',
'seq_num': 3}, page_content='Increasing consumption of seasonal fruits and vegetables can decrease the environmen
tal impact associated with growing and transporting non-seasonal produce.'), Document(id='f0bf89b6-d2b8-4d90-8165
-8f4d5ad34d6b', metadata={'source': '/content/knowledge_base.json', 'seq_num': 2}, page_content='Reducing red mea
t intake can significantly lower greenhouse gas emissions because livestock farming produces methane, a potent gr
eenhouse gas.'), Document(id='54d35fe8-261d-45a7-b4e0-049cb926682a', metadata={'source': '/content/knowledge_bas
e.json', 'seq_num': 1}, page_content='Dietary changes that reduce carbon footprint include eating more plant-base
d foods, reducing meat and dairy consumption, and choosing locally sourced produce to minimize transportation emi
ssions.')]
```

Source Attribution



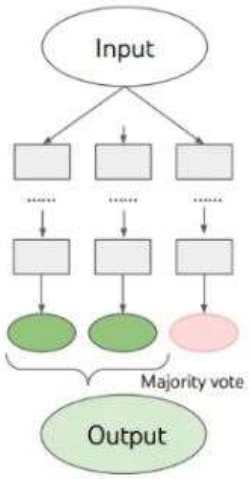
In [ ]:

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_community.vectorstores import FAISS
from langchain_google_genai import ChatGoogleGenerativeAI, GoogleGenerativeAIEmbeddings
from langchain_core.documents import Document

# Example documents
documents = [
    Document(
        page_content="The transformer architecture was introduced in the paper 'Attention is All You Need' by Vas
        metadata={"source": "Neural Network Review 2021", "page": 42}
    ),
    Document(
        page_content="BERT uses bidirectional training of the Transformer, masked language modeling, and next ser
        metadata={"source": "Introduction to NLP", "page": 137}
    ),
    Document(
        page_content="GPT models are autoregressive transformers that predict the next token based on previous to
        metadata={"source": "Large Language Models Survey", "page": 89}
    )
]

# Create a vector store and retriever
embeddings = GoogleGenerativeAIEmbeddings(
    model="models/embedding-001",
    google_api_key="AIzaSyAMAYxkjP49QZRCg21zImWWAu7c3YHJ0a8"
)
vector_store = FAISS.from_documents(documents, embeddings)
retriever = vector_store.as_retriever(search_kwargs={"k": 3})
```

```python
# Source attribution prompt template
attribution_prompt = ChatPromptTemplate.from_template("""
You are a precise AI assistant that provides well-sourced information.
Answer the following question based ONLY on the provided sources. For each fact or claim in your answer,
include a citation using [1], [2], etc. that refers to the source. Include a numbered reference list at the end.

Question: {question}

Sources:
{sources}

Your answer:
""")

# Create a source-formatted string from documents
def format_sources_with_citations(docs):
    formatted_sources = []
    for i, doc in enumerate(docs, 1):
        source_info = f"[{i}] {doc.metadata.get('source', 'Unknown source')}"
        if doc.metadata.get('page'):
            source_info += f", page {doc.metadata['page']}"
        formatted_sources.append(f"{source_info}\n{doc.page_content}")
    return "\n\n".join(formatted_sources)

# Build the RAG chain with source attribution
def generate_attributed_response(question):
    # Retrieve relevant documents
    retrieved_docs = retriever.invoke(question)

    # Format sources with citation numbers
    sources_formatted = format_sources_with_citations(retrieved_docs)

    # Create the attribution chain using LCEL
    attribution_chain = (
        attribution_prompt
        | ChatGoogleGenerativeAI(model="gemini-2.0-flash", temperature=0.7, google_api_key="AIzaSyAMAYxkjP49QZRCg
        | StrOutputParser()
    )

    # Generate the response with citations
    response = attribution_chain.invoke({
        "question": question,
        "sources": sources_formatted
    })

    return response

# Example usage
question = "How do transformer models work and what are some examples?"
attributed_answer = generate_attributed_response(question)
print(attributed_answer)
```

```
Transformer models, introduced in 2017 by Vaswani et al. [1], are used in models such as BERT [2] and GPT [3]. BE
RT uses bidirectional training of the Transformer [2]. GPT models are autoregressive transformers that predict th
e next token [3].

References:
1. Neural Network Review 2021, page 42
2. Introduction to NLP, page 137
3. Large Language Models Survey, page 89
```

Self-consistency Checking



In [ ]:
```python
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_google_genai import ChatGoogleGenerativeAI
from typing import List, Dict
from langchain_core.documents import Document

def verify_response_accuracy(
    retrieved_docs: List[Document],
    generated_answer: str,
    llm: ChatGoogleGenerativeAI = None
) -> Dict:
    """
    Verify if a generated answer is fully supported by the retrieved documents.
    Args:
        retrieved_docs: List of documents used to generate the answer
        generated_answer: The answer produced by the RAG system
        llm: Language model to use for verification
    Returns:
        Dictionary containing verification results and any identified issues
    """
    if llm is None:
        llm = ChatGoogleGenerativeAI(model="gemini-2.0-flash", temperature=0.7, google_api_key="AIzaSyAMAYxkjP49(

    # Create context from retrieved documents
    context = "\n\n".join([doc.page_content for doc in retrieved_docs])

    # Define verification prompt - fixed to avoid JSON formatting issues in the template
    verification_prompt = ChatPromptTemplate.from_template("""
    As a fact-checking assistant, verify whether the following answer is fully supported
    by the provided context. Identify any statements that are not supported or contradict the context.

    Context:
    {context}

    Answer to verify:
    {answer}

    Perform a detailed analysis with the following structure:
    1. List any factual claims in the answer
    2. For each claim, indicate whether it is:
       - Fully supported (provide the supporting text from context)
       - Partially supported (explain what parts lack support)
       - Contradicted (identify the contradiction)
       - Not mentioned in context
    3. Overall assessment: Is the answer fully grounded in the context?

    Return your analysis in JSON format with the following structure:
    {{
      "claims": [
        {{
          "claim": "The factual claim",
          "status": "fully_supported|partially_supported|contradicted|not_mentioned",
          "evidence": "Supporting or contradicting text from context",
          "explanation": "Your explanation"
        }}
      ],
      "fully_grounded": true|false,
      "issues_identified": ["List any specific issues"]
    }}
```

```
    """)

    # Create verification chain using LCEL
    verification_chain = (
        verification_prompt
        | llm
        | StrOutputParser()
    )

    # Run verification
    result = verification_chain.invoke({
        "context": context,
        "answer": generated_answer
    })

    return result

# Example usage
retrieved_docs = [
    Document(page_content="The transformer architecture was introduced in the paper 'Attention Is All You Need' b
    Document(page_content="BERT is a transformer-based model developed by Google that uses masked language modeli
]

generated_answer = "The transformer architecture was introduced by OpenAI in 2018 and uses recurrent neural netwo

verification_result = verify_response_accuracy(retrieved_docs, generated_answer)
print(verification_result)
```

```json
{
  "claims": [
    {
      "claim": "The transformer architecture was introduced by OpenAI in 2018",
      "status": "contradicted",
      "evidence": "The transformer architecture was introduced in the paper 'Attention Is All You Need' by Vaswan
i et al. in 2017.",
      "explanation": "The context states the transformer architecture was introduced by Vaswani et al. in 2017, w
hich contradicts the claim that it was introduced by OpenAI in 2018."
    },
    {
      "claim": "The transformer architecture uses recurrent neural networks",
      "status": "contradicted",
      "evidence": "It relies on self-attention mechanisms instead of recurrent or convolutional neural network
s.",
      "explanation": "The context explicitly states that the transformer architecture relies on self-attention me
chanisms *instead* of recurrent neural networks. Therefore, the claim is contradicted."
    },
    {
      "claim": "BERT is a transformer model developed by Google.",
      "status": "fully_supported",
      "evidence": "BERT is a transformer-based model developed by Google",
      "explanation": "The context directly supports this claim."
    }
  ],
  "fully_grounded": false,
  "issues_identified": [
    "Incorrect originator of the transformer architecture.",
    "Incorrect architecture used by transformers (recurrent neural networks instead of self-attention)."
  ]
}
```