L4-M4: Advanced RAG - Agentic Retrieval & Synthesis

1. Introduction to Advanced Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) has emerged as a cornerstone technique for grounding Large Language Models (LLMs) in external, up-to-date, or proprietary knowledge. While basic RAG pipelines offer a significant improvement over un-augmented LLMs, they often struggle with complex, multi-faceted queries that require deep reasoning, iterative information gathering, and sophisticated synthesis.

Advanced RAG systems move beyond the simple "query-retrieve-generate" pattern by incorporating a suite of intelligent mechanisms, including query transformation, iterative refinement, and agentic planning. This module explores these advanced techniques, focusing on the architecture and implementation of **Agentic RAG**, which leverages the LLM's reasoning capabilities to manage the entire retrieval and synthesis process.

1.1. Limitations of Basic RAG

The fundamental limitations of a naive RAG implementation stem from its linear, single-pass nature:

Limitation	Description	Advanced RAG Solution
Contextual Mismatch	A short, ambiguous query may not align semantically with the dense embeddings of long, detailed documents, leading to poor retrieval.	**Query Transformation** (e.g., HyDE) to generate a richer, more representative search vector.
Single-Step Retrieval	Complex questions requiring multiple facts or steps of reasoning are often poorly served by a single retrieval call.	**Iterative Retrieval** and **Agentic Planning** to decompose the query and execute sequential searches.
Single-Source Bias	Reliance on a single index or data source limits the scope of information and can lead to incomplete or biased answers.	**Multi-Source Search** and **Dynamic Tool Selection** to query diverse knowledge bases.
Lack of Reflection	The system cannot self-correct or refine its search strategy based on the quality of initial retrieval results.	**Reflection and Critique Agents** to evaluate retrieved context and refine the next steps.

2. Agentic Retrieval and Synthesis

Agentic RAG represents a paradigm shift, transforming the RAG pipeline from a static function into a dynamic, decision-making workflow. It introduces an orchestrating **LLM Agent** that can plan, execute, and refine a multi-step process to answer a complex query.

2.1. The Agentic RAG Architecture

The core of Agentic RAG is the decomposition of a complex task into manageable sub-tasks, each handled by specialized agents or tools. A typical workflow involves the following stages [2]:

1. **Planning and Decomposition:** The main LLM Agent receives the user query and breaks it down into a sequence of sub-queries and actions. It determines which tools (e.g., vector store, keyword search, web search) are necessary for each step.
2. **Adaptive Retrieval:** For each sub-query, a specialized **Retrieval Agent** selects the optimal retrieval strategy, potentially using different indexes or query transformation techniques.

3. **Refinement and Critique:** A **Critique Agent** evaluates the retrieved documents for relevance, completeness, and factual consistency. If the context is insufficient, the process is sent back to the Planning Agent for **Iterative Query Refinement**.
4. **Synthesis:** Once all necessary information is gathered, a final **Synthesis Agent** (often the main LLM) combines the retrieved context and intermediate findings to formulate the final, comprehensive answer.

This architecture allows the system to exhibit **deep thinking** by simulating a human researcher's workflow: planning, searching, evaluating, and synthesizing.

3. Query Transformation Techniques

Query transformation is a critical pre-retrieval step designed to bridge the semantic gap between a concise user query and the rich documents in the knowledge base.

3.1. Hypothetical Document Embeddings (HyDE)

Hypothetical Document Embeddings (HyDE) is a powerful technique for improving dense retrieval, particularly when the user query is short or lacks sufficient context [1].

3.1.1. Mechanism of HyDE

The HyDE method operates on the principle that the embedding space of a *hypothetical answer* to a query is closer to the embedding space of the *actual relevant documents* than the embedding of the query itself [1].

The process is as follows:

1. **Hypothetical Generation:** The user query (Q) is passed to an LLM, which is instructed to generate a plausible, but potentially incorrect, **hypothetical document** (D_{hyp}) that answers the question. ```math

   D_{hyp} = \text{LLM}(\text{"Write a detailed answer to the question: } Q \text{"})

```
2.  **Embedding:** The hypothetical document ($`D_{hyp}`$) is then converted into a vector embedding ($`E_{hyp}`$) using the same embedding model used for the document corpus.
    ```math

    E_{hyp} = \text{EmbeddingModel}(D_{hyp})
```
```

1. **Retrieval:** The embedding $ E_{hyp} $ is used to perform the vector search against the document index. Since $ E_{hyp} $ is a rich, long-form vector, it is more likely to capture the semantic essence of the required documents than the short query embedding $ E_Q $.
2. **Final Generation:** The retrieved, actual documents are passed to the LLM for final, factual answer generation, ignoring the initial hypothetical document [3].

Table 3.1: Comparison of Standard RAG and HyDE Retrieval

Feature	Standard RAG	HyDE-Augmented RAG
Search Vector	Embedding of the user query ($ E_Q $).	Embedding of a hypothetical document ($ E_{hyp} $).
Vector Richness	Low (short query).	High (long, generated text).
Semantic Alignment	Relies on query-document similarity.	Relies on document-document similarity, which is often higher.
Computational Cost	Low (one embedding call).	Higher (one LLM generation call + one embedding call).

3.2. Iterative Query Refinement

Iterative Query Refinement is a technique where the LLM is used to systematically modify, expand, or decompose the original query based on intermediate results. This is particularly useful in multi-source or multi-step retrieval scenarios.

Step-by-Step Iterative Refinement:

1. **Initial Retrieval:** Execute a search with the original query ($ Q_0 $).
2. **Analysis:** The LLM analyzes the top-$ k $ retrieved documents ($ R_0 $) and the original query $ Q_0 $.
3. **Refinement:** The LLM generates a new, more precise query ($ Q_1 $) based on the missing information or ambiguity identified in $ R_0 $. For example, if $ Q_0 $ was "What is the capital of the largest country in South America?", and $ R_0 $ only confirms the largest country is Brazil, $ Q_1 $ might become "What is the capital of Brazil?".
4. **Re-Retrieval:** Execute a new search with $ Q_1 $ to get $ R_1 $.

5. **Iteration:** This process repeats until the Critique Agent determines the retrieved context is sufficient for a final answer.

4. Multi-Source Search and Dynamic Tool Selection

In advanced RAG, the knowledge base is rarely a single, monolithic vector store. Instead, it is a collection of diverse sources, each optimized for different data types or retrieval methods.

4.1. The Need for Multi-Source Search

Complex, real-world queries often require information from different modalities or systems:

- **Structured Data:** Databases, APIs, or spreadsheets (best searched via SQL or function calls).
- **Unstructured Text:** Internal documents, PDFs, or knowledge bases (best searched via vector or keyword search).
- **Real-time Information:** Current events, stock prices, or weather (best searched via web search or dedicated APIs).

Multi-Source Search is the strategy of querying these diverse sources within a single RAG pipeline.

4.2. Dynamic Tool Selection

The Agentic RAG system employs **Dynamic Tool Selection** (also known as **Query Routing** or **Tool-Use**) to determine the most appropriate source and method for a given sub-query.

Process:

1. **Query Analysis:** The Planning Agent analyzes the sub-query and its intent (e.g., factual question, calculation, definition).
2. **Tool Mapping:** The agent maps the query intent to a list of available tools (e.g., `vector_db_tool`, `keyword_search_tool`, `web_search_tool`, `sql_query_tool`).

3. **Tool Execution:** The selected tool is executed, and its output (retrieved context) is returned to the agent.

This approach is significantly more effective than brute-force searching all sources, as it leverages the LLM's reasoning to select the most efficient and relevant search path.

Example: Dynamic Tool Selection

User Query	Agent's Decomposition & Tool Selection
"Compare the Q3 2024 revenue of Tesla to the key findings from their latest press release on the Cybertruck."	**Step 1 (Structured):** Query: "Tesla Q3 2024 revenue." Tool: `Financial_API_Tool` or `SQL_DB_Tool`. **Step 2 (Unstructured):** Query: "Key findings latest Cybertruck press release." Tool: `Vector_DB_Tool` (on internal documents).
"What is the definition of Agentic RAG, and how does it differ from a multi-query RAG approach?"	**Step 1 (Definition):** Query: "Definition of Agentic RAG." Tool: `Vector_DB_Tool` (on technical documentation). **Step 2 (Comparison):** Query: "Differences between Agentic RAG and multi-query RAG." Tool: `Web_Search_Tool` or `Vector_DB_Tool`.

5. Practical Application: Building a Deep-Thinking RAG Pipeline

Building an advanced RAG system requires orchestrating the components discussed above. We can conceptualize this using a multi-agent framework.

5.1. Multi-Agent Framework for Deep RAG

A robust Agentic RAG system can be structured with three primary agents:

1. **Router/Planner Agent:** The entry point. It analyzes the initial query, decides on the overall strategy (decomposition, tool selection), and manages the iterative loop.
2. **Retrieval Agent:** Executes the search. It handles query transformation (HyDE), multi-source routing, and initial filtering/re-ranking of results.

3. **Critique/Refiner Agent:** The quality control. It evaluates the retrieved context against the original query and provides feedback to the Router Agent, triggering refinement or synthesis.

5.2. Step-by-Step: Implementing HyDE for Retrieval

The following steps illustrate how to integrate HyDE into a standard RAG pipeline using a conceptual Python-like pseudocode.

```
# Conceptual Implementation of HyDE for Retrieval

def hyde_retrieval(user_query, llm_model, embedding_model, vector_store):
    """
    Performs retrieval using the Hypothetical Document Embeddings (HyDE)
technique.
    """

    # 1. Hypothetical Document Generation
    hyde_prompt = f"""
    You are an expert document writer. Based on the user's question,
    write a detailed, plausible, and complete answer. This answer does not
    need to be factually correct, but it must be semantically rich and
    representative of the document content that would contain the true answer.
    User Question: {user_query}
    """

    print("Generating hypothetical document...")
    hypothetical_document = llm_model.generate(hyde_prompt)

    # 2. Embedding the Hypothetical Document
    print("Embedding hypothetical document...")
    hypothetical_embedding = embedding_model.embed(hypothetical_document)

    # 3. Retrieval using the Hypothetical Embedding
    print("Searching vector store with hypothetical embedding...")
    retrieved_documents = vector_store.search(
        query_vector=hypothetical_embedding,
        k=5
    )

    return retrieved_documents

# Example Usage
# documents = hyde_retrieval("What were the main causes of the 2008 financial
crisis?", llm, embedder, index)
# final_answer = llm.generate_answer(user_query, documents)
```

6. Conclusion and Key Takeaways

Advanced RAG is not simply about using a better vector store or a larger LLM; it is about introducing **reasoning and planning** into the retrieval process. By adopting an

Agentic architecture, RAG systems can move from simple pattern matching to complex problem-solving.

Key Takeaways

- **Agentic RAG** employs an orchestrating LLM to plan, execute, and refine multi-step retrieval and synthesis workflows, overcoming the limitations of linear RAG.
- **Query Transformation** techniques, such as **HyDE**, improve retrieval accuracy by generating a semantically richer search vector (the hypothetical document embedding) that better aligns with the document corpus.
- **Iterative Query Refinement** allows the system to self-correct by using intermediate retrieval results to generate more precise follow-up queries.
- **Multi-Source Search** and **Dynamic Tool Selection** enable the agent to query diverse knowledge bases (vector stores, databases, web search) based on the specific requirements of each sub-query.

Mastery of these concepts is essential for building robust, high-performance RAG applications capable of handling the most challenging, real-world information retrieval tasks.

References

[1] Gao, L., Ma, X., Lin, J., & Dai, Z. (2022). **Precise Zero-Shot Dense Retrieval without Relevance Labels**. *arXiv preprint arXiv:2212.10496*. https://arxiv.org/abs/2212.10496

[2] Khan, F. (2025). **Building an Agentic Deep-Thinking RAG Pipeline to Solve Complex Queries**. *Level Up Coding*. https://levelup.gitconnected.com/building-an-agentic-deep-thinking-rag-pipeline-to-solve-complex-queries-af69c5e044db

[3] Eversberg, L. (2024). **How to Use HyDE for Better LLM RAG Retrieval**. *TDS Archive*. https://medium.com/data-science/how-to-use-hyde-for-better-llm-rag-retrieval-a0aa5d0e23e8