Comprehensive Manual on Retrieval-Augmented Generation and LLM Agents

Author: Maryam Abdolali

Institution: KNTU

Course: Large Language Models

Term: Fall 2025

Abstract

This comprehensive manual provides an in-depth exploration of Retrieval-Augmented Generation (RAG) and LLM Agent architectures. Beginning with the fundamental limitations of large language models—including hallucination, static knowledge boundaries, and context window constraints—we systematically develop the theoretical and practical foundations of RAG systems. The document covers classical and neural retrieval methods, advanced RAG techniques, tool integration strategies, and sophisticated agentic workflows. Each concept is presented with formal definitions, mathematical formulations, practical examples, and implementation considerations,

providing students with both theoretical understanding and practical insights for building robust, knowledge-grounded AI systems.

Contents

Part I: Fundamental Limitations of Large Language Models

1 Introduction: The Promise and Peril of Factoid Question Answering

Large Language Models (LLMs) have demonstrated remarkable capabilities in processing and generating human-like text. A particularly impressive achievement is their ability to answer **factoid questions**—questions that seek specific, factual information such as dates, locations, names, or numerical data. Consider the following illustrative questions:

Example

Example Factoid Questions:

- *Where is the Louvre Museum located?*
- *Where does the energy in a nuclear explosion come from?*
- *Who discovered penicillin?*
- *What is the capital of Azerbaijan?*

The apparent simplicity of querying an LLM with such questions belies a complex underlying mechanism. When an LLM correctly answers "Paris" to the first question, it is retrieving this information from its **parametric memory**—the knowledge encoded within the billions of parameters during pre-training. Research suggests that factual knowledge in transformer models is primarily stored in the feedforward layers, particularly in the middle layers of the network architecture.

Definition

Parametric Memory: The knowledge that is encoded within the weights and parameters of a neural network during the training process. For LLMs, this includes factual information, linguistic patterns, reasoning heuristics, and world knowledge acquired from the training corpus.

The capacity of LLMs to store and retrieve such information is non-trivial. However, this approach to knowledge representation leads to several critical limitations that form the motivation for Retrieval-Augmented Generation.

2 Systematic Analysis of LLM Limitations

2.1 Hallucination: The Reliability Crisis

Critical Warning

Critical Issue: Hallucination
A hallucination occurs when a language model generates content that appears plausible but is not grounded in factual reality or its provided context. This represents a fundamental reliability challenge for LLM deployment in critical applications.

The phenomenon of hallucination is not merely occasional error but a systematic issue with multiple contributing factors:

1. **Recall Failure:** Even when correct information exists within the parametric memory, the model may fail to retrieve it due to competing probabilities or attention mechanism limitations.

2. **Statistical Artifact:** LLMs generate text based on statistical patterns rather than truth verification. If certain incorrect associations were frequent in training data, they may be generated with high confidence.

3. **Calibration Mismatch:** LLMs are poorly calibrated—their confidence scores do not reliably correlate with accuracy. A model may assert incorrect information with 99% confidence.

Example

Empirical Evidence of Hallucination: Dahl et al. (2024) conducted a comprehensive study on legal domain questions and found hallucination rates between 69% and 88% across different model sizes. Even state-of-the-art models produced legally incorrect citations, misattributed precedents, and fabricated case details while maintaining high confidence scores.

The mathematical formulation of this problem can be expressed as follows: Given a question q and the true answer a^*, the LLM generates a response a_{LLM} with confidence score c. The hallucination problem manifests as:

$$P(a_{\text{LLM}} \neq a^* \mid c > \tau) \gg 0$$

where τ is a high confidence threshold (e.g., 0.9). This violates the principle of **well-calibrated** systems where high confidence should indicate high accuracy.

2.2 Knowledge Boundary Limitations

Beyond hallucination, LLMs face three additional fundamental constraints:

Definition

Static Knowledge Problem: LLMs are trained on fixed datasets with specific cutoff dates. Their knowledge cannot be updated without expensive retraining, making them unable to answer questions about recent events or developments.

Definition

Proprietary Data Exclusion: LLMs cannot access or reason about private, confidential, or proprietary information that was not included in their training corpus. This includes personal emails, corporate documents, medical records, and legal discovery materials.

Definition

Context Window Constraints: Transformer architectures have practical limits on the number of tokens they can process in a single forward pass. While context windows have expanded (e.g., 400K tokens in GPT-5.1), they remain finite and economically constrained by computational costs.

Note

Economic Consideration: LLM pricing typically follows per-token billing models. As of Fall 2025, GPT-5.1 costs approximately \$1.25 per million input tokens and \$10 per million output tokens. Processing large documents through such models can become prohibitively expensive for many applications.

3 The Retrieval-Augmented Generation Solution

The limitations described above converge to a clear architectural need: we require systems that can **dynamically access** external knowledge sources while maintaining the generative capabilities of LLMs. This leads us to the central concept of this manual.

Definition

Retrieval-Augmented Generation (RAG): A hybrid architecture that combines information retrieval systems with generative language models. RAG systems first retrieve relevant documents from external knowledge bases, then condition the LLM generation on these retrieved documents to produce accurate, verifiable responses.

The fundamental RAG pipeline consists of three sequential stages:

1. **Retrieval:** Given a user query q, retrieve the most relevant documents $D = \{d_1, d_2, ..., d_k\}$ from a knowledge base K.

2. **Augmentation:** Construct an augmented prompt p' that combines the original query with retrieved documents: $p' = f(q, D)$.

3. **Generation:** Generate the final response using the LLM conditioned on the augmented prompt: $r = \text{LLM}(p')$.

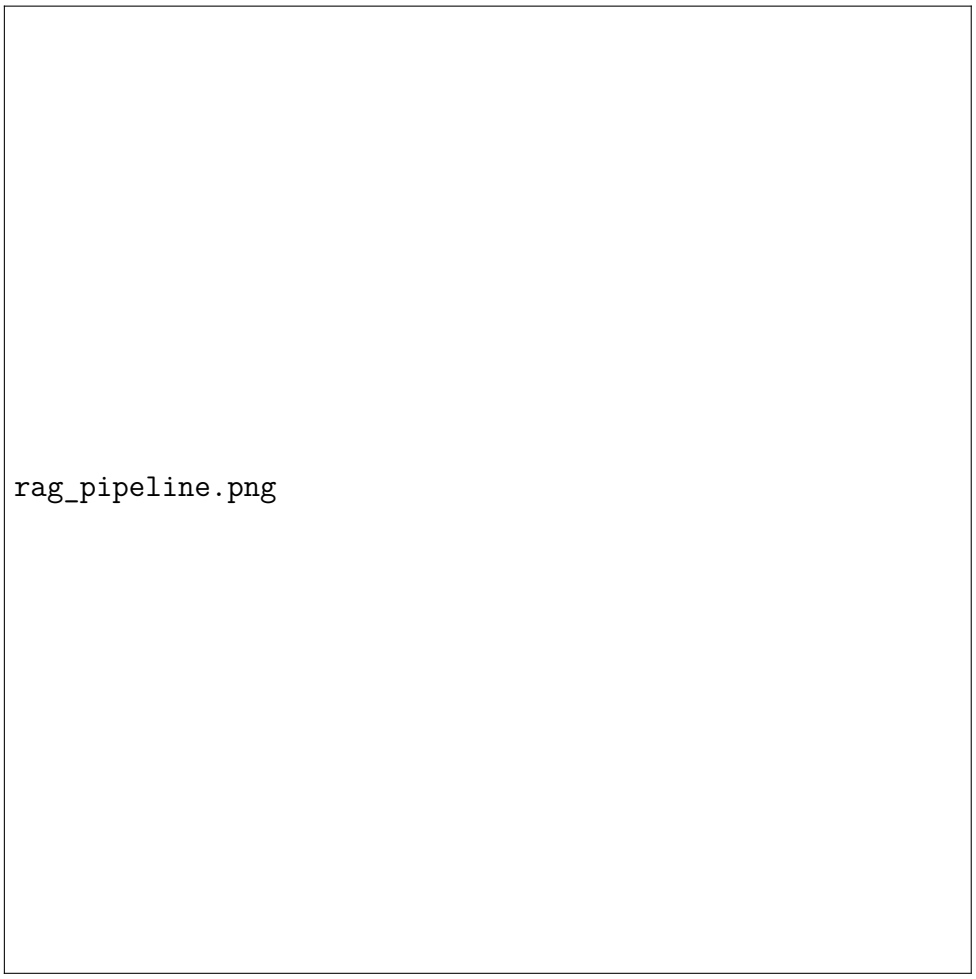

Figure 1: The three-stage RAG pipeline: Retrieve, Augment, Generate

The power of RAG lies in its ability to separate **knowledge storage** from **knowledge utilization**. The knowledge base can be updated independently of the LLM, proprietary documents can be included without model retraining, and responses can be grounded in specific sources.

Part II: Implementing Retrieval-Augmented Generation Systems

4 Knowledge Base Construction

The foundation of any RAG system is its knowledge base. Construction involves several critical design decisions that significantly impact retrieval performance.

4.1 Document Chunking Strategies

Since transformer models have finite context windows and retrieval operates on discrete units, documents must be divided into appropriately sized chunks. The optimal chunking strategy depends on the document structure and expected query types.

Algorithm 1 Adaptive Document Chunking Algorithm

procedure ChunkDocument(D, max_size, $overlap$)
    $chunks \leftarrow \emptyset$
    $current\_chunk \leftarrow \emptyset$
    $current\_size \leftarrow 0$
    for each sentence s in D **do**
        if $current\_size + |s| > max\_size$ **then**
            Add $current\_chunk$ to $chunks$
            $current\_chunk \leftarrow$ last $overlap$ sentences of $current\_chunk$
            $current\_size \leftarrow |current\_chunk|$
        end if
        Add s to $current\_chunk$
        $current\_size \leftarrow current\_size + |s|$
    end for
    if $current\_chunk \neq \emptyset$ **then**
        Add $current\_chunk$ to $chunks$
    end if
    return $chunks$
end procedure

Example

Chunking Strategy Comparison:

Strategy	Advantages	Limitations
Sentence-level	Maximum granularity, precise retrieval	Loss of context, too fine-grained for some queries
Paragraph-level	Preserves logical structure	Variable size, may be too large or small
Fixed-size windows with overlap	Consistent processing, context preservation	May break logical units
Semantic segmentation (learned)	Adapts to content structure	Requires training, computationally intensive

4.2 Metadata Enhancement

To improve retrieval accuracy, chunks can be enriched with metadata:

$$\text{EnhancedChunk} = \langle \text{Content}, \text{DocumentTitle}, \text{SectionHeader}, \text{PositionIndex}, \text{EntityTags} \rangle$$

This metadata assists both in retrieval ranking and in providing context during generation.

5 Retrieval Methods: From Keywords to Embeddings

Retrieval in RAG systems involves finding the most relevant documents from a potentially massive knowledge base. We explore both classical and neural approaches.

5.1 Classical Information Retrieval: TF-IDF and BM25

5.1.1 TF-IDF Formulation

The Term Frequency-Inverse Document Frequency (TF-IDF) framework measures word importance through two components:

Definition

Term Frequency (TF): Measures how frequently a term appears in a document, normalized by document length. Common formulations include:

$$\text{tf}_{t,d} = 1 + \log_{10}(\text{count}(t,d)) \quad \text{or} \quad \text{tf}_{t,d} = \frac{\text{count}(t,d)}{\sum_{t' \in d} \text{count}(t',d)}$$

Definition

Inverse Document Frequency (IDF): Measures how rare a term is across the entire corpus:

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

where N is the total number of documents and df_t is the number of documents containing term t.

The TF-IDF weight combines these measures:

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

For document scoring with respect to a query q, we compute cosine similarity between the query vector $\vec{q}$ and document vector $\vec{d}$:

$$\text{score}(q,d) = \cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{\|\vec{q}\|\|\vec{d}\|} = \frac{\sum_{t\in q} \text{tf-idf}_{t,q} \times \text{tf-idf}_{t,d}}{\sqrt{\sum_{t\in q} \text{tf-idf}_{t,q}^2} \times \sqrt{\sum_{t\in d} \text{tf-idf}_{t,d}^2}}$$

Example

TF-IDF Calculation Example:
Consider a corpus with 4 documents and query "sweet love":

Document 1: "Sweet sweet nurse! Love?"
Document 2: "Sweet sorrow"
Document 3: "How sweet is love?"
Document 4: "Nurse!"

For term "sweet" in Document 1: $\text{tf} = 1 + \log_{10}(2) \approx 1.301$, $\text{idf} = \log_{10}(4/3) \approx 0.125$, so tf-idf ≈ 0.163.
The full calculation shows Document 1 has the highest similarity (0.747) to the query.

5.1.2 BM25: Advanced Probabilistic Retrieval

BM25 improves upon TF-IDF with saturation functions and length normalization:

Definition

BM25 Scoring Function:

$$\text{BM25}(q, d) = \sum_{t \in q} \text{IDF}(t) \times \frac{\text{TF}(t, d) \times (k_1 + 1)}{\text{TF}(t, d) + k_1 \times \left(1 - b + b \times \frac{|d|}{|d_{\text{avg}}|}\right)}$$

where:

- $\text{IDF}(t) = \log\left(\frac{N - \text{df}_t + 0.5}{\text{df}_t + 0.5}\right)$ (Robertson-Spark Jones IDF)
- k_1 controls term frequency saturation (typically 1.2-2.0)
- b controls length normalization (0 = no normalization, 1 = full normalization)
- $|d|$ is document length in words
- $|d_{\text{avg}}|$ is average document length in corpus

The saturation function $\frac{\text{TF} \times (k_1+1)}{\text{TF}+k_1}$ ensures that additional occurrences of a term contribute diminishing returns. This reflects the intuition that the first few occurrences are most important for relevance.

5.2 Neural Retrieval with Dense Embeddings

Classical retrieval methods suffer from the **vocabulary mismatch problem**: they require exact word overlap between queries and documents. Neural methods overcome this by mapping text to dense vector spaces where semantic similarity can be measured.

5.2.1 Bi-Encoder Architecture

Definition

Bi-Encoder (Dual Encoder): Uses separate neural networks to encode queries and documents into dense vectors. Relevance is computed as the dot product or cosine similarity between these vectors:

$$\text{score}(q, d) = \text{sim}(E_Q(q), E_D(d))$$

where E_Q and E_D are query and document encoders, typically based on BERT or similar architectures.

The bi-encoder is efficient for large-scale retrieval since document embeddings can be precomputed and stored in a vector database, with retrieval performed via approximate nearest neighbor search.

5.2.2 Cross-Encoder Architecture

Definition

Cross-Encoder: Jointly processes query-document pairs through a single transformer, producing a relevance score directly:

$$\text{score}(q, d) = \text{Linear}(\text{BERT}([\text{CLS}]q\ [\text{SEP}]d\ [\text{SEP}])_{[\text{CLS}]})$$

Cross-encoders achieve higher accuracy by modeling fine-grained interactions between query and document tokens. However, they are computationally expensive since they require processing each candidate document with the query.

5.3 Hybrid and Multi-Stage Retrieval

Practical systems often combine multiple retrieval methods:

Algorithm 2 Multi-Stage Retrieval Pipeline

procedure RETRIEVE(q, K, k_1, k_2)
  $C_{\text{keyword}} \leftarrow \text{BM25Retrieve}(q, K, k_1)$ ▷ First stage: high recall
  $C_{\text{dense}} \leftarrow \text{DenseRetrieve}(q, K, k_1)$ ▷ Parallel dense retrieval
  $C_{\text{union}} \leftarrow \text{Deduplicate}(C_{\text{keyword}} \cup C_{\text{dense}})$
  $C_{\text{reranked}} \leftarrow \text{CrossEncoderRerank}(q, C_{\text{union}}, k_2)$ ▷ Second stage: high precision
  return C_{reranked}
end procedure

6 Reranking for Precision Enhancement

Reranking refines initial retrieval results using more sophisticated (but computationally expensive) models. Common approaches include:

1. **Cross-Encoder Rerankers:** Fine-tuned BERT models that score query-document pairs

2. **LLM-based Rerankers:** Prompt small LLMs to assess relevance:

   System Prompt: "Given the query '[QUERY]' and the document '[DOCUMENT]', rate the relevance from 1-10 and explain why."

3. **Ensemble Methods:** Combine multiple reranker scores with learned weights

7 Advanced RAG Techniques

7.1 Query Transformation and Expansion

Simple queries often fail to retrieve relevant documents. Query transformation techniques address this:

Definition

Query Rewriting: Use an LLM to reformulate ambiguous or verbose queries into optimal retrieval queries.

Original: "We have an essay due tomorrow. We have to write about some animal. I love penguins. I could write about them. But I could also write about dolphins. Are they animals? Maybe. Let's do dolphins. Where do they live for example?"

Rewritten: "Where do dolphins live?"

Definition

HyDE (Hypothetical Document Embeddings): Generate a hypothetical ideal answer, then retrieve documents similar to this hypothetical answer rather than the original query.

7.2 Multi-Query and Multi-Hop RAG

Complex questions require multiple retrieval steps:

Example

Multi-Hop RAG Example:

Query: "Who are the largest car manufacturers in 2023? Do they each make EVs or not?"

Step 1: Retrieve documents about "largest car manufacturers 2023"

Result: Toyota, Volkswagen, Hyundai

Step 2: For each manufacturer, retrieve documents about their EV production:

- "Toyota Motor Corporation electric vehicles"
- "Volkswagen AG electric vehicles"
- "Hyundai Motor Company electric vehicles"

Final Answer: Synthesize information from all retrieved documents.

7.3 Query Routing and Source Selection

Different queries may require different knowledge sources. Routing mechanisms direct queries to appropriate databases:

$$\text{Source}(q) = \begin{cases} \text{HR Database} & \text{if Classify}(q) = \text{"HR Question"} \\ \text{CRM Database} & \text{if Classify}(q) = \text{"Customer Query"} \\ \text{Legal Database} & \text{if Classify}(q) = \text{"Legal Question"} \\ \text{General Knowledge Base} & \text{otherwise} \end{cases}$$

Part III: LLM Agents and Tool Integration

8 From RAG to Agentic Systems

While RAG enhances LLMs with retrieval capabilities, **LLM Agents** extend this paradigm to include tool usage, planning, and interaction with external systems. An agent can be formally defined as:

Definition

LLM Agent: A system that perceives its environment through sensors (text input, API responses), reasons about goals and actions, and acts upon the environment through actuators (tool calls, API requests) to achieve objectives.

The agent architecture extends the basic LLM with three key components:

1. **Tools:** Functions or APIs that the agent can invoke
2. **Memory:** Short-term (context) and long-term (vector database) storage
3. **Planning:** Reasoning about action sequences to achieve goals

9 Tool Integration and Function Calling

9.1 Formalizing Tool Usage

Tools extend an agent's capabilities beyond text generation. Each tool is defined with:

$$\text{Tool} = \langle \text{Name}, \text{Description}, \text{Parameters}, \text{ReturnType}, \text{Implementation} \rangle$$

Example

Tool Definition Example:

```
{
  "name": "find_teddy_bear",
  "description": "Finds the nearest teddy bear
                  to given GPS coordinates",
  "parameters": {
    "type": "object",
    "properties": {
      "latitude": {"type": "number"},
      "longitude": {"type": "number"}
    },
    "required": ["latitude", "longitude"]
  },
  "returns": {
    "type": "object",
    "properties": {
      "name": {"type": "string"},
      "distance_meters": {"type": "number"},
      "model": {"type": "string"},
      "message": {"type": "string"}
    }
  }
}
```

9.2 Tool Calling Protocol

The tool calling process follows a structured conversation pattern:

Example

Tool Calling Dialogue:

Role	Content
User	What's the weather in Paris?
LLM	{"name": "get_weather", "arguments": {"city": "Paris"}}
System	{"temperature": 14, "conditions": "partly cloudy"}
LLM	It's currently 14°C in Paris with partly cloudy conditions.

Algorithm 3 Tool Calling Protocol

Initialization: Provide LLM with tool definitions
Step 1: LLM receives user query, decides if tool needed
if tool needed **then**
    LLM generates structured tool call: {"name": tool_name, "arguments": args}
    Step 2: System executes tool with provided arguments
    Step 3: System returns tool result to LLM
    Step 4: LLM incorporates result into response
else
    LLM generates direct response
end if

10 Agentic Reasoning Patterns

10.1 Chain-of-Thought (CoT) Reasoning

CoT prompting improves complex reasoning by decomposing problems:

Definition

Zero-Shot CoT: Append "Let's think step by step" to prompts to elicit reasoning traces without examples.

Few-Shot CoT: Provide examples of reasoning chains to establish the desired reasoning pattern.

Example

Few-Shot CoT Example:

Example 1: Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Example 2: Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: There are 16 balls total. Half are golf balls: $16/2 = 8$ golf balls. Half of golf balls are blue: $8/2 = 4$ blue golf balls. The answer is 4.

10.2 Self-Consistency

Improves CoT by sampling multiple reasoning paths and selecting the most consistent answer:

$$\text{Final Answer} = \arg\max_a \sum_{i=1}^{k} \mathbb{I}(a_i = a)$$

where $\{a_1, ..., a_k\}$ are answers from k sampled reasoning chains.

10.3 Tree of Thoughts (ToT)

Extends linear reasoning to tree structures, enabling exploration of multiple reasoning paths:

Algorithm 4 Tree of Thoughts Algorithm

```
procedure TreeOfThoughts(problem, breadth, depth)
    root ← CreateNode(problem)
    frontier ← [root]
    for i = 1 to depth do
        new_frontier ← ∅
        for each node n in frontier do
            thoughts ← GenerateThoughts(n, breadth)
            for each thought t in thoughts do
                child ← CreateNode(t, parent = n)
                child.value ← Evaluate(t)
                Add child to new_frontier
            end for
        end for
        frontier ← SelectTopK(new_frontier, breadth)
    end for
    return BestPath(root)
end procedure
```

11 ReAct: Reasoning and Acting Interleaved

The ReAct framework alternates between reasoning and action steps:

Definition

ReAct Cycle: At each step t, the agent:

1. **Thinks:** Generates reasoning r_t about what to do next
2. **Acts:** Executes action a_t (tool call or final answer)
3. **Observes:** Receives observation o_t from environment

The process continues until a termination condition is met.

Example

ReAct Trace Example:

Step	Content
Thought 1	Need today's date and the Louvre's official hours. Museum hours can change, so I should search for current information.
Action 1	web.search_query("Louvre Museum opening hours today")
Observation 1	Official Louvre website appears in results. Knowledge panel shows standard hours: 9:00-18:00, closed Tuesdays.
Thought 2	Need to verify if today is Tuesday and check for any special closures or extended hours.
Action 2	time.now(timezone="Europe/Paris")
Observation 2	Today is Friday, December 19, 2025, 10:30 AM.
Thought 3	Friday is not Tuesday. Should check for Friday-specific hours or late openings.
Action 3	web.find("Friday hours Louvre")
Observation 3	Louvre has late opening until 21:45 on Fridays.
Final Answer	Today (Friday, December 19, 2025), the Louvre is open from 9:00 to 21:45. It is closed on Tuesdays.

12 Reflection and Self-Critique

Reflection enables agents to improve their outputs through iterative refinement:

Algorithm 5 Reflection Pattern

```
procedure REFLECT(query, initial_response, max_iterations)
    response ← initial_response
    for i = 1 to max_iterations do
        critique ← LLM("Critique this response: ", response)
        if critique indicates no improvements needed then
            break
        end if
        response ← LLM("Improve based on critique: ", response, critique)
    end for
    return response
end procedure
```

13 Multi-Agent Systems

Complex tasks can be distributed among specialized agents:

Definition

Multi-Agent System: A collection of specialized agents that collaborate to solve problems. Common patterns include:

- **Supervisor-Agents:** Hierarchical structure with a supervisor delegating to specialized workers
- **Collaborative Networks:** Peer-to-peer collaboration among equals
- **Competitive Systems:** Multiple agents propose solutions, with selection or voting

Example

Multi-Agent Research Assistant:

- **Query Analyst:** Understands and decomposes complex queries
- **Retrieval Specialist:** Expert in finding relevant information
- **Analytical Agent:** Performs calculations and logical reasoning
- **Synthesis Agent:** Combines information into coherent responses
- **Quality Assurance:** Reviews and critiques outputs

14 Implementation Considerations

14.1 Memory Management

Agents require both short-term and long-term memory:

Definition

Short-term Memory: Maintained within the LLM's context window through conversation history and recent observations.

Long-term Memory: Stored externally in vector databases or structured storage, with retrieval mechanisms for relevant recall.

14.2 Error Handling and Recovery

Robust agents require mechanisms to handle failures:

1. **Tool Execution Errors:** Retry with modified parameters, fallback to alternative

tools

2. **Retrieval Failures:** Query reformulation, broadening search scope
3. **Reasoning Errors:** Self-critique, alternative reasoning paths
4. **Contradiction Resolution:** Evidence weighing, source verification

14.3 Economic Optimization

Agentic systems must balance performance with cost:

$$\text{Total Cost} = \sum_{\text{tokens}} \text{Input Cost} + \sum_{\text{tokens}} \text{Output Cost} + \sum_{\text{tools}} \text{API Costs} + \text{Computational Overhead}$$

Strategies include:

- Caching frequent retrievals and computations item Using smaller models for simpler subtasks
- Early termination when confidence is high
- Batch processing of similar operations

Conclusion and Future Directions

The integration of retrieval capabilities with large language models through RAG architectures represents a fundamental advancement in AI system design. By separating knowledge storage from knowledge utilization, these systems address critical limitations of standalone LLMs while preserving their remarkable generative capabilities.

The evolution from basic RAG to sophisticated agentic systems with tool integration, planning, and multi-agent collaboration creates a pathway toward truly autonomous AI assistants. However, significant challenges remain:

1. **Evaluation Complexity:** Measuring the accuracy, reliability, and efficiency of RAG and agentic systems requires sophisticated benchmarks beyond traditional NLP metrics.

2. **Scalability:** As knowledge bases grow to billions of documents, retrieval efficiency becomes paramount while maintaining precision.

3. **Verification and Trust:** Ensuring that generated responses are properly grounded in retrieved sources and that tool usage follows intended constraints.

4. **Multimodal Integration:** Extending RAG to incorporate images, audio, video, and structured data sources.

5. **Adaptive Learning:** Systems that can learn from interactions to improve retrieval, tool usage, and reasoning over time.

The field continues to evolve rapidly, with innovations in retrieval models, agent architectures, and evaluation methodologies. Students mastering these concepts today will be at the forefront of designing the next generation of intelligent systems that can truly understand, reason about, and interact with the world's knowledge.

Appendix: Mathematical Reference

Retrieval Scoring Functions

$$\text{TF-IDF:} \quad \text{score}(q,d) = \frac{\sum_{t \in q} \text{tf-idf}_{t,q} \cdot \text{tf-idf}_{t,d}}{\|\vec{q}\|\|\vec{d}\|}$$

$$\text{BM25:} \quad \text{score}(q,d) = \sum_{t \in q} \text{IDF}(t) \cdot \frac{\text{TF}(t,d) \cdot (k_1+1)}{\text{TF}(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{|d_{\text{avg}}|}\right)}$$

$$\text{Cosine Similarity:} \quad \text{sim}(\vec{u},\vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\|\|\vec{v}\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2}\sqrt{\sum_{i=1}^{n} v_i^2}}$$

Agent Decision Theory

The optimal action for an agent can be formulated as:

$$a^* = \underset{a \in \mathcal{A}}{\text{argmax}} \left[\mathbb{E}_{o \sim P(o|a,s)} \left[R(s,a,o) + \gamma V(s') \right] \right]$$

where:

- $\mathcal{A}$ is the set of available actions (including tool calls)
- s is the current state (conversation history, retrieved documents)
- o is the observation resulting from action a
- R is the reward function (quality of response, user satisfaction)
- V is the value function of future states
- γ is the discount factor

Information Theory of Retrieval

The optimal retrieval can be viewed as maximizing mutual information:

$$D^* = \underset{D \subseteq K}{\text{argmax}} I(A; D \mid Q)$$

where:

- Q is the query
- D is the retrieved documents
- A is the answer
- $I(X; Y \mid Z)$ is conditional mutual information

This formulation captures the intuition that ideal retrieval provides documents that maximally reduce uncertainty about the correct answer.

References

[1] Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33.

[2] Guu, K., et al. (2020). REALM: Retrieval-Augmented Language Model Pre-Training. *Proceedings of the 37th International Conference on Machine Learning*.

[3] Izacard, G., et al. (2021). Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*.

[4] Yao, S., et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *International Conference on Learning Representations*.

[5] Wei, J., et al. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems*, 35.

[6] Wang, X., et al. (2023). Self-Consistency Improves Chain of Thought Reasoning in Language Models. *International Conference on Learning Representations*.

[7] Yao, S., et al. (2024). Tree of Thoughts: Deliberate Problem Solving with Large Language Models. *Advances in Neural Information Processing Systems*, 36.

[8] Schlag, I., et al. (2021). Improving Retrieval for RAG with Hypothetical Document Embeddings (HyDE). *arXiv preprint arXiv:2112.07577*.

[9] Ram, O., et al. (2023). In-Context Retrieval-Augmented Language Models. *Transactions of the Association for Computational Linguistics*, 11.

[10] Gao, L., et al. (2023). Retrieval-Augmented Generation: A Survey. *arXiv preprint arXiv:2310.10944*.

[11] Borgeaud, S., et al. (2022). Improving Language Models by Retrieving from Trillions of Tokens. *International Conference on Machine Learning*.

[12] Shuster, K., et al. (2022). Retrieval Augmentation Reduces Hallucination in Conversation. *Findings of the Association for Computational Linguistics: EMNLP 2022*.