# Modular and GraphRAG：Evolving RAG Architectures

# Acknowledgements


## Technical Writer

Kwang-Yong Jung
Minji Kang
Jaeho Kim
Jaemin Hong
Harheem Kim

## Contributor & Reviewer

Teddy Lee

## Curators and Editors

Aera Shin

# Contents



# 1. Introduction

## 1.1 Background of Retrieval-Augmented Generation (RAG)

Large Language Models (LLMs) have shown remarkable advancements in handling complex language tasks. However, they still face limitations such as outdated knowledge and hallucinations. To overcome these challenges, Retrieval-Augmented Generation (RAG) was introduced in 2020 by researchers at Facebook AI Research (now

Meta AI), allowing LLMs to access external knowledge bases dynamically. This enables improved factual accuracy and context-specific responses by grounding model outputs in retrieved information.

RAG has become widely adopted in various applications such as customer support, recommendation systems, enterprise knowledge retrieval, and real-time information processing. Its fundamental process consists of:

1. **Indexing** : Converting documents into searchable vectors by preprocessing data and generating embeddings.
2. **Retrieval**: Fetching relevant documents based on a user query using similarity search techniques.
3. **Generation**: Using the retrieved data to augment the prompt and enhance LLM responses with factual, up-to-date information.

This approach effectively addresses the limitations of traditional LLMs by reducing hallucinations and providing access to information beyond the model's training cutoff date.

## 1.2 Limitations of Traditional RAG and the Need for Advancement

While Retrieval-Augmented Generation (RAG) has significantly improved LLM capabilities by enabling external knowledge retrieval, traditional approaches face multiple challenges. These limitations can be categorized into Naive RAG and Advanced RAG constraints.

### Limitations of Naive RAG

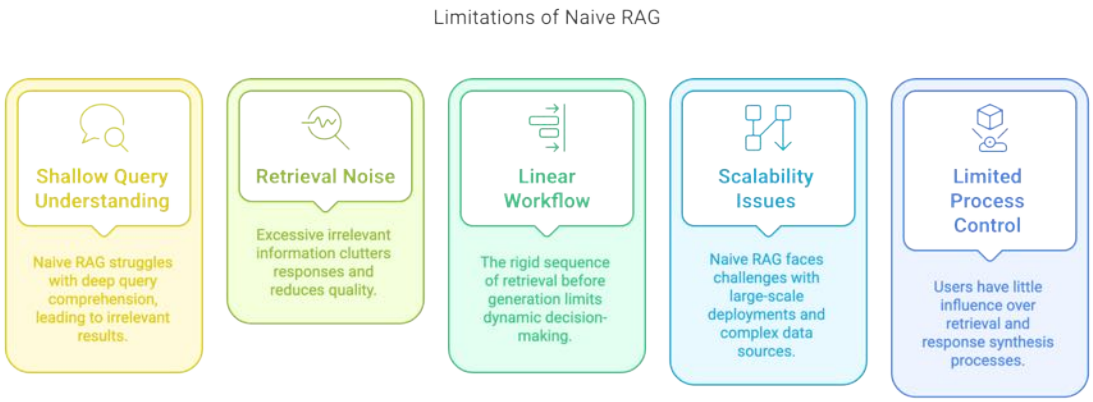
- **Shallow Query Understanding**: Naive RAG relies heavily on keyword or vector-based similarity matching, which often results in a lack of deep query comprehension. It struggles with ambiguous, multi-faceted, or context-dependent queries, leading to irrelevant or incomplete retrieval results. While effective for straightforward "synthesis queries," naive RAG systems perform poorly on complex multi-hop queries or those requiring deeper reasoning.

- **Retrieval Noise**: Since Naive RAG selects documents primarily based on simple similarity measures, it frequently retrieves excessive, irrelevant, or contradictory information. This retrieval noise—categorized as irrelevant, redundant, or counterfactual—can clutter responses, increase hallucinations, and cause context window overflow in LLMs, leading to reduced response quality.

- **Linear Workflow**: The sequential nature of Naive RAG mandates that retrieval always precedes generation, leaving no room for dynamic decision-making or iteration. If incorrect documents are retrieved, the system lacks self-correction mechanisms before response generation, causing error propagation. More advanced systems address this through iterative retrieval or agentic approaches that can decompose and refine queries.

- **Scalability Issues**: As data sources grow in volume and complexity, Naive RAG's fixed retrieval-then-generation pipeline becomes computationally expensive and inefficient, leading to increased response times and processing costs. While the architecture itself can scale with appropriate infrastructure, the quality of results often degrades with larger datasets without additional optimization techniques. Additionally, integrating structured data sources such as relational databases or knowledge graphs poses significant challenges due to the need for specialized retrieval mechanisms.

- **Limited Process Control**: Users have minimal influence over retrieval, ranking, and response synthesis processes. Naive RAG operates as a black box, making it difficult to fine-tune retrieval strategies or dynamically optimize responses based on context or user feedback. Advanced approaches address this through configurable pipelines, explicit retrieval filtering, and adaptive ranking mechanisms.

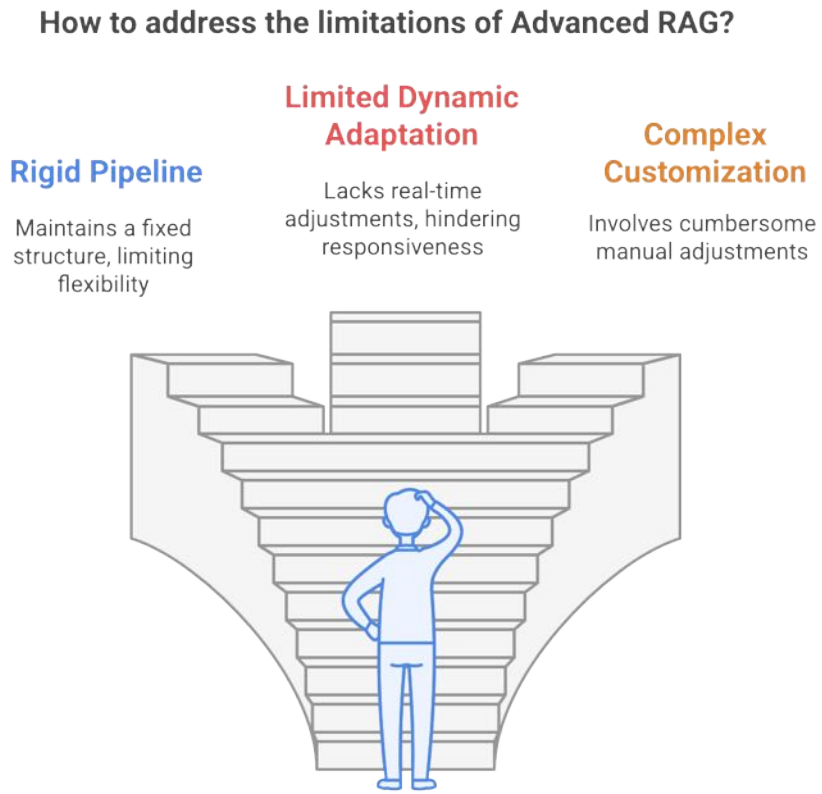



## Limitations of Advanced RAG

To address some of Naive RAG's limitations, Advanced RAG incorporates improvements such as query rewriting, re-ranking, and sophisticated retrieval techniques. However, despite these enhancements, certain challenges remain:

- **Implementation Complexity**: While Advanced RAG systems offer greater flexibility through modular components, this increased sophistication comes with higher implementation and maintenance complexity. Organizations must invest significant resources in building, tuning, and maintaining these more complex systems.
- **Computational Overhead**: The additional processing steps in Advanced RAG—such as query decomposition, multiple retrieval rounds, and re-ranking—introduce computational overhead that can impact response times and resource utilization, particularly in high-volume applications.
- **Technical Expertise Requirements**: Advanced RAG systems demand deeper technical expertise across multiple domains, including vector databases, embedding techniques, LLM fine-tuning, and pipeline orchestration. This requirement creates barriers to adoption for organizations with limited AI/ML capabilities.
- **Integration Challenges**: While Advanced RAG offers improved capabilities for handling diverse data sources, integrating heterogeneous information systems (such as combining unstructured document retrieval with structured database queries) remains challenging and often requires custom connectors and specialized knowledge.
- **Evaluation Complexity**: As RAG systems become more sophisticated, evaluating their performance becomes increasingly complex. Traditional metrics may not fully capture improvements in nuanced areas like factuality, relevance, and coherence, making system optimization more difficult.

Recent developments in the RAG ecosystem have focused on further enhancing modularity, creating standardized interfaces between components, improving evaluation frameworks, and developing more accessible tooling to address these remaining limitations. Rather than representing a distinct category, these evolved approaches extend the Advanced RAG paradigm with greater emphasis on composability, observability, and systematic evaluation.



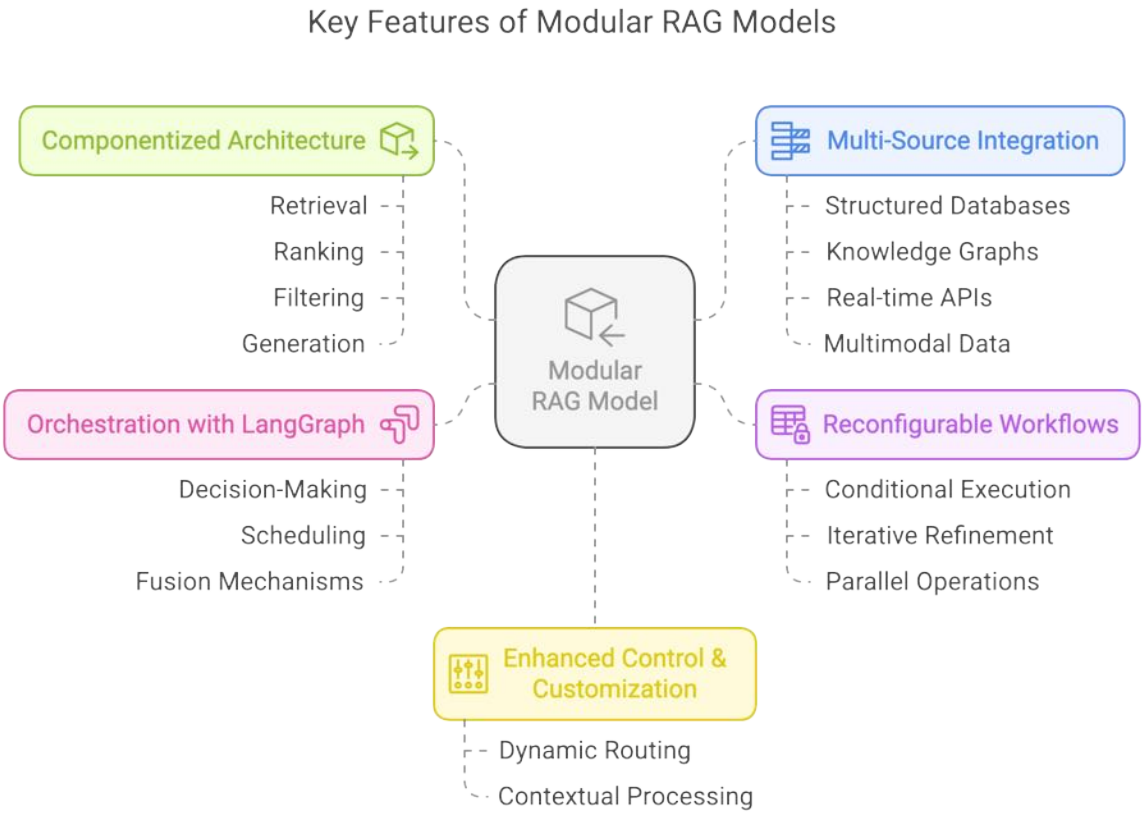

## 1.3 Introduction to Modular RAG Architectures

Modular approaches to Retrieval-Augmented Generation represent an evolution in RAG architecture, designed to address the limitations of simpler implementations by introducing a highly flexible and adaptable framework. Instead of following a fixed, linear retrieval-to-generation pipeline, modular RAG architectures decompose the system into independent, interchangeable components, enabling greater control, scalability, and efficiency in knowledge retrieval and synthesis.

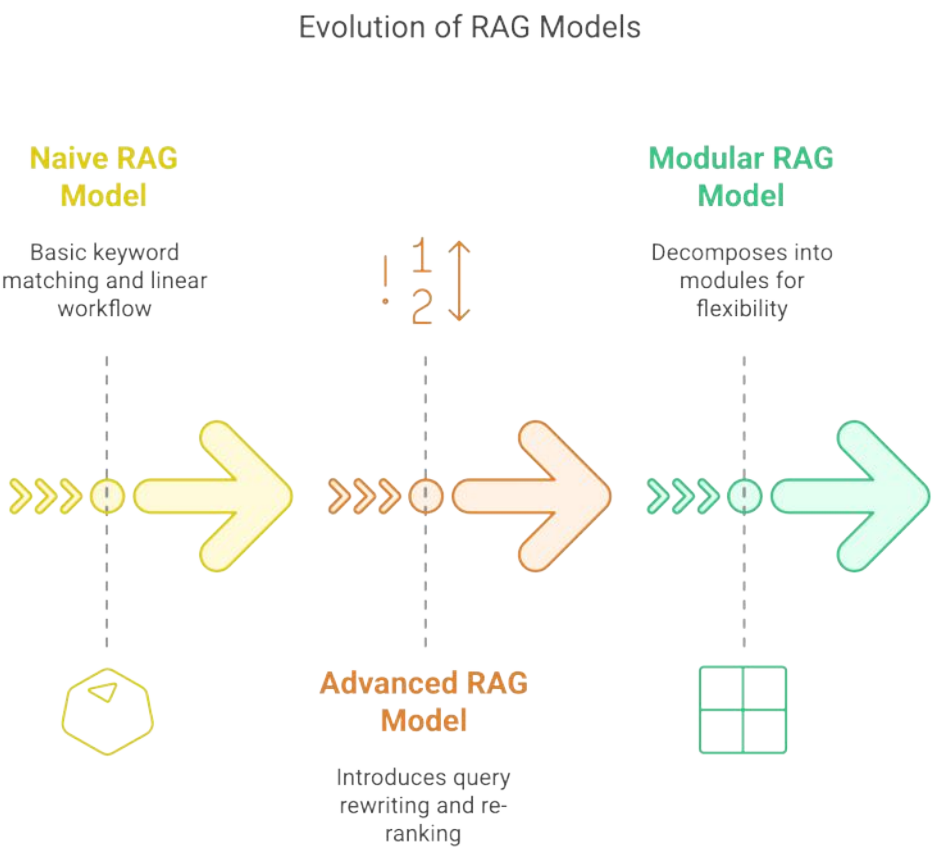
### Key Characteristics of Modular RAG Architectures

- **Componentized Architecture:** Each fundamental process in RAG—such as query analysis, retrieval, ranking, filtering, and generation—is implemented as an independent module. This design allows for easy upgrades, replacements, and optimizations without disrupting the overall system. By enabling independent management of components, modular architectures enhance system maintainability and foster adaptability to domain-specific needs.
- **Reconfigurable Workflows**: Unlike basic RAG implementations, which follow a rigid retrieve-then-generate approach, modular architectures introduce dynamic workflows that adapt based on query complexity and intent. This flexibility allows for conditional execution paths, iterative refinement processes, and parallel retrieval-ranking-generation operations, improving response quality and contextual accuracy.

- **Enhanced Control & Customization**: Developers and system architects gain fine-grained control over retrieval strategies, ranking algorithms, and response synthesis processes. By introducing decision-making mechanisms within the workflow, modular RAG supports dynamic routing, where queries are processed based on their context, source reliability, and domain-specific constraints.
- **Multi-Source Integration**: Basic RAG implementations primarily focus on unstructured text retrieval. Modular architectures extend this capability by seamlessly integrating structured databases, knowledge graphs, real-time APIs, and multimodal data sources. This multi-source approach enriches the retrieved content, leading to more accurate and contextually relevant outputs.
- **Orchestration Frameworks**: Frameworks like LangGraph, LlamaIndex, and DSPy facilitate the implementation of modular RAG by providing tools for orchestrating complex workflows. These frameworks incorporate decision-making, scheduling, and fusion mechanisms to optimize retrieval and generation processes. The orchestration layer ensures that retrieval and ranking components interact efficiently, facilitating dynamic query expansion, re-ranking, and multi-turn dialogue support.




Modular architectures for RAG represent an important advancement in retrieval-augmented generation systems. By emphasizing component independence, workflow flexibility, and system adaptability, these approaches help address common challenges in RAG implementations—such as retrieval noise, workflow rigidity, and limited scalability. This architectural philosophy aligns with broader software engineering principles of modularity and provides a solid foundation for continuously incorporating advances in retrieval techniques, ranking algorithms, and generation strategies. As the field evolves, modular approaches to RAG enable organizations to build more maintainable, extensible, and effective knowledge systems that can adapt to changing information needs and technological capabilities.




# 2. Key Concepts of RAG

Here's a breakdown of the key RAG concepts, categorized into Naïve RAG, Advanced RAG, and Modular RAG

## 2.1. Naive RAG

### Core Components

Naïve RAG consists of three core components: indexing, retrieval, and generation.

### How it works

Documents are divided into small chunks, converted into vector representations using an embedding model, and stored in a vector database.
A user query is transformed into a vector using the same embedding model. The system then retrieves the top k most similar document chunks from the vector database.
The retrieved document chunks and the user query are inputted together into an LLM to generate the final answer.

### Limitations

- **Vector Similarity Limitations:** Relies primarily on vector similarity between the query and document chunks, which may not capture complex semantic relationships or nuances in the query intent.
- **Retrieval Redundancy and Noise**: Directly feeding all retrieved chunks into LLMs can introduce irrelevant information or conflicting context, increasing the risk of generating erroneous responses.
- **Difficulty Handling Complex Queries**: Performance degrades when faced with multi-part queries, implicit questions, or when the knowledge base contains diverse or specialized content.

### Features

A straightforward retrieval method based on similarity calculations, making it effective for simple question-answering but challenging to apply to complex scenarios.

## 2.2. Advanced RAG

### Core Improvements

Advanced RAG focuses on optimizing the entire RAG pipeline, including pre-retrieval, retrieval, and post-retrieval stages, to enhance both retrieval accuracy and generation quality.

### Key Strategies

#### Pre-retrieval Processing

- **Query Rewriting**: Makes queries clearer and more specific, thereby increasing the accuracy of retrieval.
- **Query Expansion**: Expands a single query into multiple queries to enrich the query content and address any lack of specific nuances.
- **Query Transformation**: Retrieves and generates based on a transformed query instead of the user's original query.

#### Post-retrieval Processing

- **Reranking:** Enhances the visibility of more crucial document chunks by reordering them based on relevance to the query using cross-encoders or other specialized models.
- **Compression**: Reduces the retrieved content through summarization, selective extraction, or redundancy removal to help LLMs focus on key information and manage context window limitations.
- **Selection**: Directly filters out irrelevant chunks using relevance scoring or metadata filtering, improving the reasoning efficiency of the LLM.

### Features

While significantly enhancing retrieval accuracy and generation quality through sophisticated processing throughout the pipeline, Advanced RAG still faces limitations including handling multi-hop reasoning, managing contradictory information, and dynamically adjusting to varying query complexities in real-world applications.

## 2.3. Modular RAG

### Core Concept

Modular RAG decomposes complex RAG systems into independent modules and specialized operators, providing a highly reconfigurable framework. This design moves beyond the traditional linear architecture and integrates routing, scheduling, and fusion mechanisms.

### Three-Tier Structure

- Module Type (First Level): Represents the critical stages of RAG as independent functional categories, including indexing, retrieval, generation, and a central orchestration component.
- Module (Second Level): Consists of specific functional implementations within each module type, further refining and optimizing functions.
- Operator (Third Level): The basic unit of operation that implements specific algorithmic functions within a module, representing the most granular level of the system.

#### Modules and Operators

- Indexing Module: Handles document splitting, chunk optimization, and structured indexing.
- Pre-retrieval Module: Performs query expansion, query transformation, and query construction.
- Retrieval Module: Handles sparse retrieval, dense retrieval, hybrid retrieval, and retriever fine-tuning.
- Post-retrieval Module: Processes retrieved chunks through reranking, compression, and selection.
- Generation Module: Generates answers using LLMs, including generator fine-tuning and verification.
- Orchestration Module: Controls the entire RAG process through routing, scheduling, and fusion, serving as the central coordination mechanism for implementing various flow patterns.

#### RAG Flow Patterns

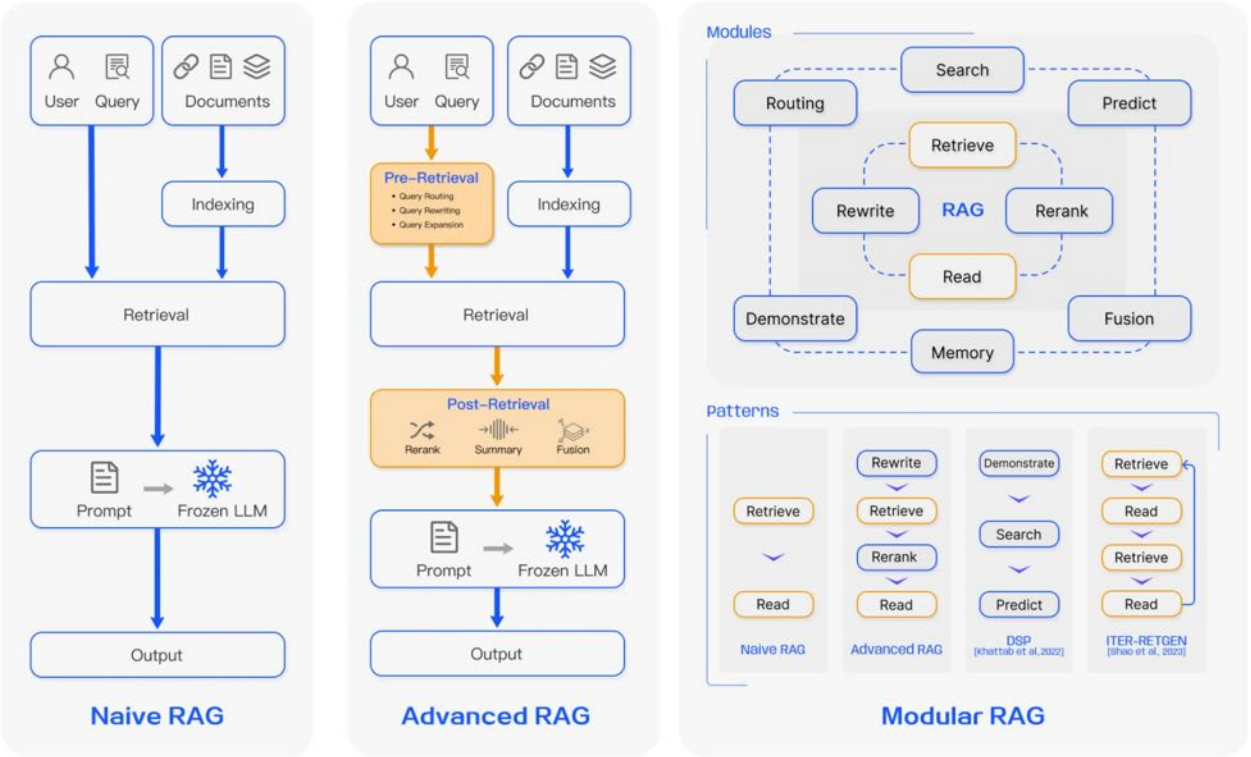
- **Linear Pattern**: Modules are processed sequentially in a fixed order, representing both Naive and Advanced RAG paradigms depending on included modules.
- **Conditional Pattern:** Selects different RAG pipelines based on various conditions such as query type or document characteristics.

- **Branching Pattern:** Uses multiple parallel running branches to increase the diversity of generated results and explore different retrieval or generation strategies simultaneously.
- **Loop Pattern:** Involves interdependent retrieval and generation steps, including:
  - Iterative retrieval: Multiple rounds of retrieval to refine results
  - Recursive retrieval: Breaking down complex queries into sub-queries and building hierarchical responses
  - Adaptive retrieval: Dynamically adjusting retrieval strategies based on intermediate results and feedback
- **Tuning:** Fine-tunes the retriever, generator, or both to optimize system performance for specific domains or tasks.

## Features

- **High Flexibility and Scalability:** Users can flexibly combine different modules and operators according to the requirements of data sources and task scenarios.
- **Enhanced Maintainability and Comprehensibility**: Independent module design facilitates system maintenance and debugging.
- **Compatibility with New Methods**: The modular structure enables easy integration of new functional modules and workflows.
- **Evolution of RAG Systems**: Builds upon the development of previous RAG paradigms, with Advanced and Naïve RAG being special cases of Modular RAG.

In summary, Modular RAG provides a robust framework for effectively managing the complexity of RAG systems, allowing for the creation of customized systems tailored to diverse needs. This modular approach presents a solid theoretical foundation and a practical roadmap for the continuous evolution and practical deployment of RAG technology.




## 2.4 Comparison of Major RAG Methodologies

Each RAG methodology has its own advantages and limitations depending on the specific context and purpose. Therefore, selecting and applying the appropriate methodology is important. Naive RAG, with its simple three-step process (indexing, retrieval, generation), can be quickly implemented but suffers from issues such as retrieval noise and limited accuracy. Advanced RAG incorporates sophisticated techniques including pre-retrieval optimization, enhanced retrieval methods (like dense retrieval and re-ranking), and generation refinement to improve accuracy and relevance. Modular RAG, in contrast, breaks the RAG process into independent, interchangeable components, enabling greater flexibility and scalability through customizable retrieval and generation modules, making it highly adaptable and optimizable for various domains.

| Category | Naive RAG | Advanced RAG | Modular RAG |
|---|---|---|---|
| Concept | ● Basic retrieval and generation approach | ● Optimized retrieval and post-processing applied | ● Modular retrieval and generation structure |

| | | | |
|---|---|---|---|
| **Search Method** | ● Simple vector search | ● Hybrid Search (keyword + semantic) | ● Multiple data sources and adaptive search |
| **Response Generation** | ● Uses retrieved documents as is | ● Generation after filtering and summarization | ● Optimized modular-based response generation |
| **Accuracy** | ● Heavily depends on retrieval quality | ● Improved accuracy through retrieval post-processing | ● Precise filtering and utilization of multiple data sources |
| **Flexibility & Scalability** | ● Fixed retrieval-generation structure | ● Limited adjustability | ● Dynamic workflow and scalable modular structure |
| **Applicable Domains** | ● Simple FAQs, document search | ● Suitable for legal, finance, and healthcare domains | ● Real-time data utilization and customizable search |
| **Advantages** | ● Simple and easy-to-understand basic approach | ● Employs advanced retrieval techniques (e.g., indexing, pre-retrieval, post-retrieval) to overcome some limitations of Naive RAG | ● Modular architecture enables independent management<br>● Bidirectional process allows for conditional handling and user intervention, offering high flexibility and scalability |
| **Limitations** | ● Low query comprehension due to reliance on similarity calculations<br>● Retrieval noise introduces unnecessary information<br>● Due to the fixed linear workflow, previous results cannot be modified | ● Still suffers from the structural limitation of a one-directional process, similar to Naive RAG | ● Complex initial design<br>● Maintenance burden |

This table can help compare the characteristics of each RAG methodology and assist in selecting the optimal approach that meets the project's objectives and requirements. Modular RAG provides high flexibility and scalability, demonstrating robust performance in fields that require complex domain knowledge.

# 3. Key Components of Modular RAG

The hierarchical components and their roles in Modular RAG are as follows

| Agent | Workflow |
|---|---|
| Module | Responsible for key processes (e.g., retrieval, augmentation, generation) |
| Sub-module | Handles detailed task steps within a module |
| Operator | Performs fundamental tasks such as data processing and transformation |

We'll examine each component step by step.

## 3.1 AI Agent Architecture

The core of Modular RAG is its agent-based architecture. Unlike general AI agent architecture, the agent in Modular RAG is designed to coordinate multiple modules and execute specific tasks. Accordingly, I will distinguish between a general AI Agent and a Modular RAG Agent and explain them separately.

### Agent-Based Architecture

An AI agent is a system that perceives, interprets, and acts within an environment to achieve a specific goal. The AI agent architecture is a structural design that defines how an AI system perceives, analyzes, and acts within its environment. It includes key components that enable the agent to operate autonomously and accomplish its objectives, incorporating various design patterns and structural elements.

### AI Agent Architecture in Modular RAG

A Modular RAG based AI system is designed with a modular structure that separates retrieval and generation functions. It builds flexible and extensible AI by combining multiple independent modules. The core components of this system are agents and modules.

The agent plays a key role in orchestrating the overall workflow, managing the system's flow and strategic control.

Unlike a standard LLM, which primarily relies on pre-trained knowledge and in-context learning, an agent dynamically interacts with external tools (e.g., web search, APIs, structured databases) to augment its capabilities. It autonomously decides which tools to invoke based on the given task, enabling more accurate and context-aware responses. Additionally, the agent employs advanced routing mechanisms to analyze user queries and direct them to the most relevant data sources or processing paths. For example, it can determine whether a query requires retrieving information from a specific database, conducting a web search, or leveraging domain-specific knowledge.

The agent also utilizes reasoning and routing frameworks, such as the ReAct framework (Reasoning and Acting), to iteratively refine its outputs by combining retrieval, decision-making, and generation steps. This allows it to break

down complex problems into manageable sub-tasks, access real-time or domain-specific knowledge, and adapt dynamically to diverse user needs. By integrating routing and tool usage, the agent expands beyond the limitations of traditional LLMs, ensuring efficient query handling and precise responses tailored to the context. In advanced implementations known as Agentic RAG, the system can incorporate specialized agents handling distinct tasks, support multi-step reasoning, and employ adaptive query handling to further enhance its performance on complex tasks.

Unlike traditional single-model approaches, the Modular RAG architecture is designed as a three-tier system with modules, sub-modules, and operators that retrieve, process, and generate information through collaboration between multiple specialized components orchestrated by an agent. This architecture supports various workflow patterns including linear, conditional, branching, and looping for flexible task execution. Here, modules function as independent components responsible for specific tasks, such as retrieval, query expansion, and routing. These modules work in coordination to analyze user queries, determine the optimal processing path, and invoke the appropriate tools, enabling more precise and context-aware responses. The agent orchestrates these modules, ensuring they operate efficiently and cohesively to achieve its objectives.

This structure enables the system to provide more refined and optimized responses to user queries.

## Definition of AI Agent in Modular RAG

An AI agent in Modular RAG is an artificial intelligence component designed to autonomously interact with various modules and tools to achieve specific goals. It orchestrates the overall workflow, makes strategic decisions, and integrates different components (e.g., LLMs, retrieval modules, prompts, and external tools) to generate optimal results as an autonomous and goal-oriented agent.

## Role of AI Agent in Modular RAG

The agent acts as the central controller that orchestrates the entire system and makes strategic decisions to dynamically optimize execution paths. In other words, it analyzes user requests and internal conditions to invoke the appropriate modules, adjust the execution order, and, if necessary, modify the workflow during execution.

An AI agent is an independent system designed to autonomously interact with its environment and achieve specific goals. It can perceive data, analyze it, make decisions, and take actions.

Tools can be directly invoked by the agent or operate independently within specific modules. For example, a tool responsible for making API calls can be executed directly within a module, while in some cases, the agent itself may invoke tools to perform specific functions.

## Characteristics of AI Agent in Modular RAG

Dynamic Workflow Management: Selects and connects necessary modules based on the situation to create a flexible workflow.

- **Decision Making:** Employs advanced reasoning algorithms to determine optimal task sequencing, execution paths, and resource allocation when solving complex problems. This involves breaking down queries into sub-tasks, formulating execution strategies based on available modules and tools, and continuously adapting these strategies based on intermediate results and changing conditions.
- **Module Interface Management:** Coordinates data exchange, execution order, and dependencies between modules and submodules to maintain system consistency. Each agent is responsible for a specific role (e.g., retrieval, generation, data processing) and operates independently while maintaining communication channels with other agents. In a multi-agent system, agents are connected to different modules to distribute tasks and utilize coordination protocols to ensure aligned strategies. Through structured message passing, shared knowledge repositories, and established collaboration frameworks, complex tasks can be efficiently executed with clear synchronization mechanisms that prevent conflicts and redundancies.
- **Flexibility & Scalability:** To add new functionalities, simply introduce a new agent that performs the required task.
- **Parallel Processing:** Multiple specialized agents can work simultaneously on different aspects of a complex query, significantly improving overall system throughput and response times. This concurrent operation enables efficient handling of multi-step reasoning tasks, simultaneous querying of diverse data sources, and the ability to process different modalities of information in parallel, making the system particularly effective for time-sensitive or computationally intensive applications.

## AI Agent Interactions

The agent orchestrates the entire workflow and executes tasks using LLMs, tools, and prompts.
Tools can be used as independent resources for specific functions (e.g., retrieval, computation) and, in some cases, can be treated as a standalone module. In modern Modular RAG architectures, tools are typically integrated through standardized interfaces that allow for consistent invocation patterns, input/output handling, and error management. The agent-tool relationship follows established orchestration patterns where tools may be invoked synchronously or asynchronously depending on the task requirements and system design.

Within tools, detailed tasks (e.g., API calls or data filtering) can be defined as submodules.
In some systems, LLMs may function as an independent module, and certain tools can be executed directly without an agent.

## Example

- **Search Agent**: Retrieves information from external databases (e.g., Tavily Search, web scraper).
- **Document Writing Agent:** Composes documents based on the retrieved data.
- **Graph Creation Agent:** Generates graphs as needed based on the retrieved data (e.g., using Python REPL).
- **Supervisor Agent:** Serves as a higher-order coordination entity that orchestrates multiple specialized agents through hierarchical control structures. It manages task delegation, resolves conflicts, monitors performance, and maintains alignment with overall objectives. The supervisor implements governance protocols that balance agent autonomy with system-level constraints and can dynamically adjust the allocation of tasks based on real-time performance metrics and changing priorities.

## 3.2 Module

### Module Definition

A module is a core building block of the Modular RAG system, independently performing key system functions or tasks (e.g., information retrieval, summarization, text generation). Each module is designed to execute specific tasks autonomously and can operate separately from other modules.

### Role of Modules

Modules directly execute specific functions based on user requests. They are independent components responsible for performing specialized tasks. Each module operates with a distinct function and collaborates under the agent's direction to achieve the final goal. Inter-module communication is facilitated through standardized message formats, shared context objects, and well-defined APIs that enable seamless data exchange. These communication protocols support various interaction patterns including request-response, publish-subscribe, and streaming, allowing for efficient data flow while maintaining loose coupling between modules.

For example, a retrieval module searches for relevant information from external data sources, while a generation module creates the final response. Modules can also handle tasks such as data preprocessing and text generation independently.

### Characteristics of Modules

- **Independence & Modularity:** Modules are designed as independent units, allowing them to function autonomously with minimal dependencies.
- **Specialization:** Each module focuses on a specific task or function, enabling optimization and expertise in its domain.
- **Reusability & Flexibility:** Modules can be reused within the system and adapted through easy modifications, replacements, or extensions. They can also be applied to other projects when needed.
- **Scalability**: The system can evolve by adding new modules or enhancing existing ones to accommodate new features or services.
- **Composability:** Multiple modules can be integrated through well-defined interfaces and composition patterns to handle complex tasks or introduce new functionalities. This integration follows established architectural patterns including pipeline arrangements, branching workflows, feedback loops, and ensemble methods. Each composition pattern offers distinct advantages for different use cases, with modern systems often employing hybrid approaches that combine multiple patterns to achieve optimal performance and flexibility.

## 3.3 Sub-Module

### Definition of Submodules

A submodule is a unit that decomposes specific functions of a parent module for execution. It operates through standardized interfaces with clearly defined input/output contracts, enabling it to be added, combined, or replaced as needed. In sophisticated RAG architectures, certain submodules may be shared across multiple modules or operate independently, facilitated by interface consistency and adherence to modular design principles that ensure compatibility across the system.

## Role of Submodules

Within a module, complex tasks are divided into multiple sub-steps through established composition patterns, clarifying the role of each step. These patterns include sequential pipelines, parallel execution frameworks, conditional branching structures, and recursive processing models. Dividing complex tasks into smaller steps not only improves efficiency but also enables specialized optimization of each component, better error isolation, and more granular performance monitoring. The orchestration of these submodules follows design principles that balance coupling, cohesion, and information flow.

To enhance the performance and flexibility of the parent module, submodules focus on implementing detailed functions.

Each submodule is responsible for a specific function within the parent module, allowing for targeted improvements or modifications when issues arise. This granular structure enables independent testing, validation, and performance measurement of each component, which is critical for robust system development. Submodules can be evaluated in isolation with standardized test suites, facilitating continuous improvement through targeted refinements that don't risk destabilizing the entire system. In other words, submodules help structure complex tasks into distinct steps within a parent module, ensuring clear roles and responsibilities while enabling focused troubleshooting and refinement when needed.

## Characteristics of Submodules

- **Decomposition**: Divide complex functions of the parent module into smaller, well-defined steps for better management.
- **Independence:** Each submodule operates independently without interference, minimizing its impact on other components.
- **Maintainability:** When issues arise, only the affected submodule needs to be modified, making system maintenance easier.
- **Reusability:** Well-defined submodules can be easily reused in other modules or projects.
- **Scalability:** When new features or modifications are needed, specific submodules can be expanded or replaced while maintaining interface contracts, ensuring overall system scalability. This modular evolution is supported by version management protocols that maintain backward compatibility or provide clear migration paths. Modern RAG architectures implement submodule registry systems that track dependencies, compatibility constraints, and version histories to ensure coherent system evolution across multiple development cycles.

## 3.4 Operator

### Definition of Operators

An operator is the smallest atomic execution unit within a module, responsible for performing a specific well-defined function. Each operator functions independently, exposing standardized interfaces with clearly defined inputs and outputs, ensuring modularity and reusability. Operators follow uniform design patterns that include error handling protocols, performance monitoring hooks, resource management capabilities, and versioning metadata, enabling consistent behavior across the system architecture. In the provided content, the Operators within each module perform specific tasks (e.g., chunk optimization, query expansion, retriever selection) and are independently defined.

### Relationship Between Modules and Operators：

Modular RAG is based on a hierarchical structure of modules and operators, where each module consists of multiple operators organized in defined processing sequences. Operators can be composed using various patterns including sequential chains, parallel execution groups, conditional branches, and feedback loops. This composition flexibility allows for complex data transformation pipelines where the output of one operator becomes the input for subsequent operators, enabling sophisticated processing workflows while maintaining clean architectural boundaries. Each module (e.g., Indexing, Pre-Retrieval, Retrieval, Post-Retriever, Generation, Orchestration) contains its own set of operators, designed to perform specific functions through standardized implementation patterns. Modern RAG frameworks provide developer-friendly APIs for creating custom operators that conform to the system's interface requirements. These extensibility mechanisms allow for the integration of domain-specific operators, proprietary algorithms, or third-party services while maintaining architectural consistency. Operators can be implemented using various technologies, from simple functions to containerized microservices, depending on complexity and resource requirements.

### Role of Operators

Operators are responsible for a diverse range of specialized tasks, functioning as the actual execution units within a module. They can be categorized into several functional types：data processing operators (e.g., chunk optimization, normalization, filtering), transformation operators (e.g., query expansion, document translation, format conversion), analysis operators (e.g., reranking, relevance scoring, sentiment analysis), integration operators (e.g., API connectors, database adapters), and utility operators (e.g., caching, logging, rate limiting). This taxonomy enables systematic design and selection of appropriate operators for specific RAG pipeline requirements.

### Examples of Operators by Module

Here, the modules are considered as Indexing, Pre-Retrieval, Retrieval, Post-Retrieval, Generation, and Orchestration.

### 1. Indexing:

- Module: Responsible for chunking and optimizing documents.
- Operator:
  - Chunk Optimization: Adjusting chunk size, overlap, chunk boundary determination (semantic vs. syntactic), and enriching with structured metadata.
  - Structural Organization: Hierarchical indexing, knowledge graph (KG)-based indexing, parent-child relationships, and recursive retrieval structures.
  - Vector Index Configuration: Selection of vector dimensions, quantization parameters, ANN algorithm selection (HNSW, IVF, etc.), and sharding strategies for large-scale deployments.
  - Multi-modal Indexing: Processing and indexing of text, images, and other data types within unified retrieval frameworks.

Operators related to chunking and structuring focus on data processing tasks.

### 2. Pre-Retrieval:

- Module: The process of improving the original query to optimize search performance.
- Operator:
  - Query Expansion: Multi-Query, Sub-Query.
  - Query Transformation: Rewrite, HyDE, Step-back Prompting.
  - Query Construction: Text-to-SQL, Text-to-Cypher.

Operators that transform and expand queries fall under data transformation and analysis.

### 3. Retrieval:

- Module: Retrieves relevant documents based on the user query.
- Operator:
  - Retriever Selection: Sparse (BM25, SPLADE), Dense (embedding-based), Hybrid (ensemble methods combining multiple retrieval signals), and Graph-based retrievers.
  - Retriever Fine-tuning: Supervised Fine-Tuning (SFT), Parameter-Efficient Fine-Tuning (PEFT), Adapters, and Contrastive Learning approaches.
  - Multi-Vector Retrieval: Parent-child chunking strategies, document-passage dual encodings, and hierarchical retrievers that capture multiple representation levels.
  - Cross-Encoder Integration: Combination of bi-encoders for initial retrieval with cross-encoders for precision refinement.

Operators related to retriever selection and fine-tuning play a key role in improving retrieval efficiency.

### 4. Post-Retrieval:

- Module: Processes retrieved documents to optimize them for LLM input.
- Operator:
  - Rerank: Rule-based Rerank, Cross-encoder Reranking, Reciprocal Rank Fusion (RRF), FLARE (Forward-Looking Active REtrieval), and Relevance-guided reranking.

- Compression: Selective Context, LLM-Critique, Recursive Summarization, MapReduce paradigms for large context windows, and Attention-guided pruning.
- Context Augmentation: Entity linking, Knowledge Graph enrichment, Citation retrieval, and Contextual metadata integration.
- Diversity Optimization: Maximum Marginal Relevance (MMR), clustering-based selection, and coverage optimization algorithms.

Operators in post-processing play a role in improving efficiency by sorting and compressing data.

### 5. Generation:

- Module: Generates responses based on retrieved information.
- Operator:
  - Generator Fine-tuning: Instruct-Tuning, Reinforcement Learning, Dual Fine-tuning.
  - Verification: Knowledge-based Verification, Model-based Verification.

Operators contribute to data analysis and optimization by fine-tuning and verifying generated content to enhance output quality.

### 6. Orchestration:

- Module: Controls the overall flow of the RAG system and performs dynamic decision-making.
- Operator:
  - Routing: Hybrid Routing.
  - Scheduling: Rule-based Judge, LLM-based Judge, Knowledge-guided Scheduling.
  - Fusion: LLM Fusion, Weighted Ensemble, RRF (Reciprocal Rank Fusion).

Operators related to routing and scheduling support dynamic decision-making.

We propose a framework that decomposes the RAG system into independent modules and specialized operators, enabling flexible reconfiguration. This approach goes beyond traditional linear architecture by incorporating advanced mechanisms such as routing, scheduling, and fusion. The framework supports multiple architectural patterns:

- **Linear Pattern:** Sequential processing through predefined module stages
- **Conditional Pattern:** Dynamic path selection based on query characteristics or intermediate results
- **Branching Pattern:** Parallel execution of multiple retrieval or generation strategies with subsequent result fusion
- **Iterative Pattern:** Recursive refinement through feedback loops between generation and retrieval stages
- **Self-reflective Pattern:** Integration of self-evaluation and correction mechanisms within the processing pipeline

Each pattern offers distinct advantages for different use cases, with implementation details covering inter-module communication, state management, and failure handling strategies.

## 3.5 Example of Overall Structure

Let's illustrate an example from the perspective of Modular RAG. When creating an agent, it requires LLM, tools, and prompts.

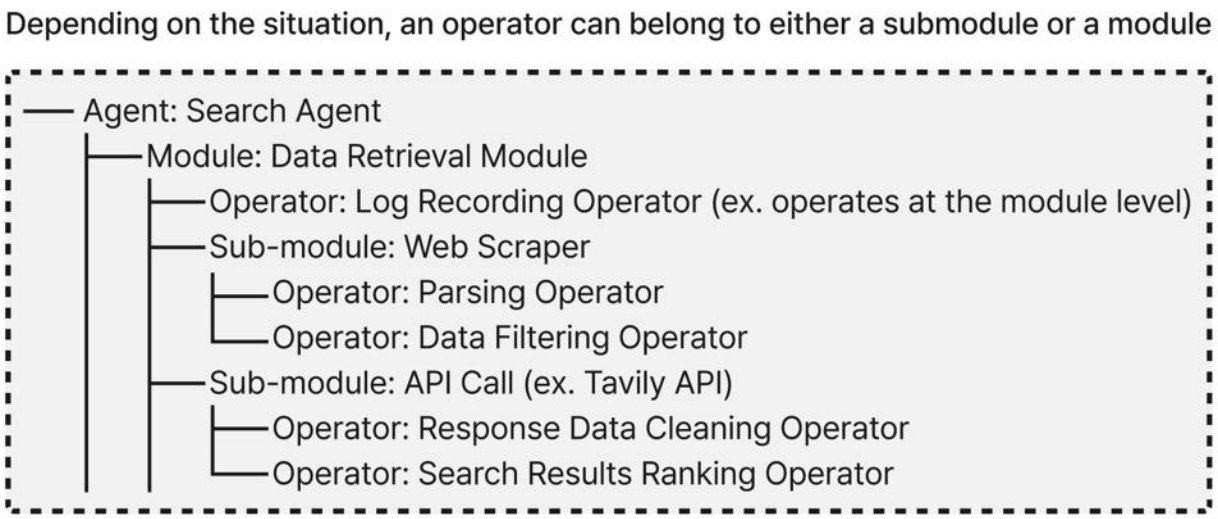
Here, tools represent functional capabilities that can be integrated into the architecture at different levels. From an architectural perspective, tools are implemented as either: (1) dedicated modules with well-defined interfaces within the system, (2) submodules that provide specific functionality within a larger module, or (3) external services accessed through standardized connectors. This classification depends on factors including the tool's complexity, frequency of use, coupling requirements, and deployment constraints. Each implementation approach offers different trade-offs between integration depth, reusability, and maintenance complexity.

In architectural terms, tools and modules represent different conceptual layers of the system. Modules define structural components with clear boundaries and interfaces, while tools represent functional capabilities that can be implemented across different architectural layers. A tool's functionality may be contained within a single module, span multiple modules, or be provided by external services. Modern Modular RAG systems employ a capability-based abstraction layer that separates the logical definition of tools (what they do) from their implementation details (how they do it), enabling flexible composition and substitution of components.

The functionality provided by tools is typically implemented through one or more submodules organized in a cohesive structure. Each tool capability may be realized through a collection of specialized submodules that handle different aspects of the tool's operation, such as input validation, business logic, external integration, error handling, and result formatting. This layered implementation approach ensures separation of concerns while maintaining the tool's unified interface from the perspective of the agent.

An operator represents the smallest atomic execution component within the architecture, responsible for performing a singular well-defined task such as data processing, computation, or conditional control. Operators implement standardized interfaces that enable them to be composed into processing pipelines within submodules. They adhere to functional design principles including immutability, idempotence, and explicit input/output contracts, making them highly testable and reusable. Modern RAG frameworks provide extensive operator libraries covering common functions while supporting custom operator development for specialized requirements.

### Example Structure




## 3.6 UseCase: Accuracy Improvement

This case demonstrates how Modular RAG can enhance accuracy.

### Building a PDF-Based Retriever Chain

```
from rag.pdf import PDFRetrievalChain

pdf = PDFRetrievalChain(["data/Newwhitepaper_Agents2.pdf"]).create_chain()

pdf_retriever = pdf.retriever

pdf_chain = pdf.chain
```

### Query router  chain

```
from typing import Literal

from langchain_core.prompts import ChatPromptTemplate
from pydantic import BaseModel, Field
from langchain_openai import ChatOpenAI



# 1. Define a Pydantic model for data routing
class RouteQuery(BaseModel):
```

```
    """Route a user query to the most relevant datasource."""

    datasource: Literal["vectorstore", "web_search"] = Field(
        ...,
        description="Given a user question choose to route it to web search or a vectorstore.",
    )

# 2. Configure the LLM
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)
structured_llm_router = llm.with_structured_output(RouteQuery)

# 3. Define the system message (responsible for routing user queries to the appropriate data source)
system = """You are an expert at routing a user question to a vectorstore or web search.
The vectorstore contains documents related to DEC 2023 AI Brief Report(SPRI) with Samsung Gause, Anthropic, etc.
Use the vectorstore for questions on these topics. Otherwise, use web-search."""

# 4. Create a prompt template
route_prompt = ChatPromptTemplate.from_messages(
    [
        ("system", system),
        ("human", "{question}"),
    ]
)

# 5. Generate the query router (Prompt + LLM)
question_router = route_prompt | structured_llm_router
```

## Execute chain

```
    print(question_router.invoke({"question": "How do agents differ from standalone language models?",})

print(question_router.invoke({"question": "List up the name of the authors."}))
```

## Retriever Grade Chain

```
    from pydantic import BaseModel, Field
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate

# 1. Define a Pydantic model for document relevance grading
class GradeDocuments(BaseModel):
    """Binary score for relevance check on retrieved documents."""
```

```
    binary_score: str = Field(
        description="Documents are relevant to the question, 'yes' or 'no'"
    )


# 2. Configure the LLM
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)
structured_llm_grader = llm.with_structured_output(GradeDocuments)


# 3. Define the system message (sets the role of assessing document relevance)
system = """You are a grader assessing relevance of a retrieved document to a user question. \n
    If the document contains keyword(s) or semantic meaning related to the user question, grade it as relevant. \n
    It does not need to be a stringent test. The goal is to filter out erroneous retrievals. \n
    Give a binary score 'yes' or 'no' score to indicate whether the document is relevant to the question."""


# 4. Define the prompt template (inputs retrieved documents and user questions for evaluation)
grade_prompt = ChatPromptTemplate.from_messages(
    [
        ("system", system),
        ("human", "Retrieved document: \n\n {document} \n\n User question: {question}"),
    ]
)


# 5. Create the document grading system (Prompt + LLM)
retrieval_grader = grade_prompt | structured_llm_grader
```

## Output (Relevance Evaluation Results for 10 Retrieved Items)

```
binary_score='yes'
binary_score='yes'
binary_score='yes'
binary_score='yes'
binary_score='no'
binary_score='yes'
binary_score='yes'
binary_score='yes'
binary_score='yes'
binary_score='no'
```

## Example Code for Filtering

```
    filtered_docs = []
for doc in docs:
  result = retrieval_grader.invoke(
    {"question": question, "document": doc.page_content}
  )
  if result.binary_score == "yes":
    filtered_docs.append(doc)

filtered_docs
```

## Hallucination Checker Chain

```
    class GradeHallucinations(BaseModel):
  """Binary score for hallucination present in generation answer."""


  binary_score: str = Field(
    description="Answer is grounded in the facts, 'yes' or 'no'"
  )


llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)
structured_llm_grader = llm.with_structured_output(GradeHallucinations)


system = """You are a grader assessing whether an LLM generation is grounded in / supported by a set of retrieved facts. \n
  Give a binary score 'yes' or 'no'. 'Yes' means that the answer is grounded in / supported by the set of facts."""


hallucination_prompt = ChatPromptTemplate.from_messages(
  [
    ("system", system),
    ("human", "Set of facts: \n\n {documents} \n\n LLM generation: {generation}"),
  ]
)


hallucination_grader = hallucination_prompt | structured_llm_grader
```

## Evaluating Hallucination in Generated Answers Using a Grader

```
    hallucination_grader.invoke({"documents": filtered_docs, "generation": generation})
```

## Query Rewriter Chain

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser


# 1. Configure the LLM
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)


# 2. Define the system message
system = """You a question re-writer that converts an input question to a better version that is optimized \n
for vectorstore retrieval. Look at the input and try to reason about the underlying semantic intent / meaning."""


# 3. Define the prompt template (inputs the initial question for rewriting)
re_write_prompt = ChatPromptTemplate.from_messages(
  [
    ("system", system),
    (
      "human",
      "Here is the initial question: \n\n {question} \n Formulate an improved question.",
    ),
  ]
)


# 4. Create the question rewriting pipeline (Prompt + LLM + Output Parser)
question_rewriter = re_write_prompt | llm | StrOutputParser()
```

## Question rewrite

```
question = "How do agents differ from standalone language models?"
question_rewriter.invoke({"question": question})
```

output :

```
'What are the key differences between agents and standalone language models?'
```

# 4. Modular RAG Architecture for Domain-Specific AI Systems

## Introduction

The Modular RAG architecture plays a crucial role in building domain-optimized retrieval and generation systems. By combining orchestration components, specialized modules, and granular operators tailored to each domain's specific requirements, we can design more sophisticated and flexible question-answering systems.

- **Orchestration Layer**: Coordinates the entire system, analyzing user queries and determining the optimal retrieval and generation workflow.
- **Module:** Performs core functions such as retrieval, generation, and evaluation, optimized for each domain's needs.
- **Operator:** The smallest execution unit within a module, responsible for tasks like data preprocessing, filtering, and analysis.

This structured approach enables the effective application of domain-specific retrieval and generation patterns. For example, Dense and Sparse Retrieval represent fundamental retrieval paradigms that can be implemented as specialized retrieval modules or combined in hybrid approaches, while Fusion & Re-rank functions as a module with built-in operators to refine and optimize search results. Additionally, LLM Critique—which evaluates generated responses—can be implemented as an evaluation module with multiple assessment operators.

Rather than applying individual patterns in isolation, the orchestration layer dynamically combines Modules and Operators to optimize the entire retrieval and generation process. This allows for the creation of a highly adaptable and intelligent RAG-based search system.

## Domain-Specific Optimization Strategies

By tailoring retrieval and generation patterns to each domain, we can build more reliable and accurate question-answering systems:

- **Healthcare**: Since accuracy and trustworthiness are critical, Graph RAG architectures integrating patient records, medical literature, and clinical guidelines are deployed. These systems leverage knowledge graphs for multi-hop reasoning, connecting symptoms, diagnoses, and treatments while incorporating patient history to provide contextually informed, personalized care recommendations.
- **Legal**: Legal reasoning relies heavily on precedents and statutory analysis, so citation validation frameworks and legal-specific evaluation systems (such as LexRAG and CitaLaw) are implemented. These systems reduce hallucination risks through fine-tuned retrievers and citation-guided generation, enhancing reliability and jurisdictional compliance.
- **Finance**: Real-time market dynamics require temporal-aware multi-modal RAG systems that integrate heterogeneous data sources including financial tables, news articles, and price histories. These systems employ domain-specific sentiment analysis models (like FinBERT) and temporal reasoning capabilities to provide contextually relevant investment insights.

- **E-commerce**: Personalized recommendations benefit from the Rewrite-Retrieve-Read (RRR) pattern combined with feedback-optimized reinforcement learning. These systems leverage hybrid filtering approaches (collaborative and content-based) that dynamically adapt to user behavior, optimizing suggestions based on implicit and explicit user feedback signals to significantly increase conversion rates.

## RAG Components and Techniques


### Dense vs. Sparse Retrieval:

- **Dense Retrieval**: Uses vector-based semantic similarity to retrieve the most relevant information based on conceptual meaning rather than exact keyword matches.

- **Sparse Retrieval (BM25, etc.)**: A keyword-based search method that ranks documents based on term frequency and inverse document frequency, optimized for exact and partial keyword matching.

### Query Expansion / Transformation:

- Enriches and refines user queries by generating alternative formulations, adding synonyms, or breaking complex queries into sub-queries to improve search result quality.

### Knowledge Graph:

- Represents conceptual relationships in a graph structure, enabling connections between entities and concepts to enhance contextual understanding and support multi-hop reasoning.

### Fusion & Re-rank:

- Combines results from multiple retrieval methods and adjusts ranking weights based on relevance signals, coherence, and contextual appropriateness to generate the most relevant response.

### Text-to-SQL:

- Converts natural language queries into structured SQL queries, enabling precise retrieval from relational databases while maintaining the conversational interface.

### RRR (Rewrite-Retrieve-Read):

- Rewrites the user's query into multiple sub-queries → Retrieves relevant documents for each sub-query → Reads and processes the retrieved information to synthesize a comprehensive response.

### Reinforcement Learning from Human Feedback (RLHF):

- Fine-tunes language models using reward models trained on human preferences, optimizing outputs for helpfulness, accuracy, and alignment with specified objectives.

### Adaptive Retrieval & Generation:

- Implements a systematic process to dynamically adjust retrieval strategy when initial results are insufficient, using techniques like self-query, step-back prompting, and FLARE (Forward-Looking Active REtrieval).

### LLM Critique:

- Employs self-evaluation or external model evaluation to assess the reliability, accuracy, and completeness of generated responses, triggering correction mechanisms when necessary.

Hybrid Retrieval Systems:

- Integrates multiple retrieval approaches (sparse, dense, graph-based) with adaptive weighting and routing mechanisms that optimize performance based on query characteristics and context.

Optimizing RAG-based search systems for specific domains maximizes accuracy, reliability, personalization, and real-time adaptability across various fields, including healthcare, law, finance, and e-commerce.

The orchestration of retrieval components, query transformers, rerankers, generators, and evaluation frameworks is key to managing complex question-answering workflows efficiently. By leveraging these components strategically, we can create highly specialized and intelligent retrieval-augmented generation systems.

Now, let's explore how these optimized retrieval and generation patterns are applied in each domain with concrete examples.

# 4.1 Medical Domain - AI-Based Diagnosis and Treatment Recommendation

## Objective

Traditional medical search systems typically integrate multiple data sources including academic papers, clinical guidelines, and structured medical databases, though they often lack personalized integration with individual patient data. By leveraging a modular RAG framework, multiple medical data sources can be combined to produce highly reliable search results. Additionally, by incorporating a patient's medical history and genetic factors, the system can deliver personalized medical recommendations.

## Key Module Combinations

- **Query Expansion**: Refining user queries to generate multiple related queries using medical ontologies, specialized thesauri (like MeSH), and contextual understanding of medical terminology
- **Hybrid Retrieval (Dense + Sparse + Knowledge Graph)**: Searching for medical literature, clinical guidelines, and patient-similar cases using a combination of vector embedding similarity, keyword matching, and knowledge graph relationships
- **Generation (Optional)**: Generating an initial response based on retrieved information if necessary
- **Fusion**: Merging research papers and clinical guidelines to generate the best response
- **Patient Medical History Integration:** Incorporating the patient's structured medical history, genetic predispositions, medication records, and previous treatment responses through secure data connectors that align patient-specific contexts with retrieved medical knowledge, while maintaining privacy compliance
- **Generation:** Producing the final response based on the fused data

## Design Flow

### 1. Query Expansion

- Analyzing user input and generating multiple related queries
- Example: "Fever above 38°C for three days with a cough" → "Infectious diseases causing fever", "Conditions associated with fever and cough"

### 2. Hybrid Retrieval

- **Medical Literature Database:** Searches for medical papers and case studies with similar symptoms using dense retrieval
- **Knowledge Graph:** Maps relationships between symptoms and diseases to enhance diagnosis and treatment recommendations

### 3. Generation (Optional)

- If retrieved information is insufficient, an LLM generates a preliminary response

### 4. Fusion

- Integrating research papers and clinical guidelines to provide a comprehensive response
- Clearly citing information sources to ensure credibility

### 5. Patient Medical History Integration

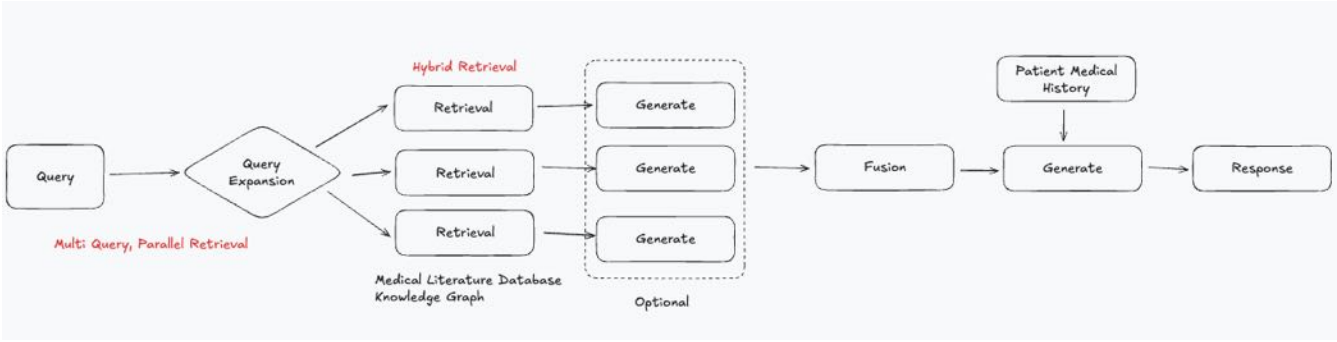
- Incorporating the patient's structured medical history, genetic predispositions, medication records, and previous treatment responses through secure data connectors that align patient-specific contexts with retrieved medical knowledge, while maintaining privacy compliance
- Example: "Best flu treatment options for a patient with a family history of diabetes"

### 6. Generation

- Producing a final, user-friendly medical response based on the fused data

### 7. Evaluation and Explainability

- Implementation of confidence scoring for recommendations
- Transparent citations linking each recommendation to specific medical sources
- Explanation of reasoning pathways connecting symptoms to diagnoses to treatments
- Clinician feedback mechanisms for continuous improvement




## Improvements

- Going beyond simple academic paper retrieval by implementing semantic integration of structured clinical data, real-world evidence, and standardized medical guidelines using domain-specific embedding models to achieve comprehensive and contextually precise recommendations
- Implementing secure, HIPAA-compliant patient history-based personalized search using a structured SOAP-inspired methodology that contextualizes individual symptoms, laboratory values, and treatment response patterns to significantly enhance diagnostic accuracy and treatment specificity
- Implementing a tiered update system for medical research integration that rapidly incorporates verified treatment advances from peer-reviewed sources, with clear confidence scoring based on evidence levels, publication recency, and clinical consensus to ensure both timeliness and reliability of treatment recommendations
- Leveraging specialized AI hardware accelerators to enable real-time integration and analysis of multi-modal medical data including diagnostic imaging, genomic profiles, and continuous monitoring data from wearable devices for more comprehensive patient assessment

# 4.2 Legal Domain - AI-Powered Legal Consultation System (Legal RAG)

## Objective

Traditional legal search systems already integrate diverse sources including statutory laws, case precedents, and regulatory guidance, though they often lack the contextual understanding to deliver personalized interpretations based on specific factual scenarios. By applying a Modular RAG framework, we can comprehensively analyze legal documents and recommend legal interpretations tailored to specific cases and jurisdictions.

## Key Module Combinations

- **Routing Module (Route):** Analyzes user queries to determine the appropriate retrieval strategy based on jurisdictional context, case type, and legal domain specificity (e.g., contract law, criminal law, intellectual property)

- **Retrieval (Retrieve):** Searches for relevant legal documents using citation network analysis to identify binding precedent, track subsequent treatment (affirmed, distinguished, overruled), and filter by jurisdictional authority and temporal relevance
- **Generation (Generate):** Provides an initial legal interpretation based on retrieved information
- **Query Transformation (Optional)**: Reformulates user questions into precise legal terminology for additional searches
- **Legal Authority Assessment:** Evaluates retrieved documents based on hierarchical precedential value, jurisdictional authority, recency, subsequent treatment by other courts, and factual similarity to the present case, filtering according to stare decisis principles
- **Adaptive Retrieval & Generation:** Conducts additional searches when further legal references are required
- **Loop RAG Flow Orchestration:** Dynamically determines retrieval and generation paths based on the type of legal inquiry
- **Ethical and Regulatory Compliance:** Implements safeguards against unauthorized practice of law through clear disclaimers, scope limitations, and distinction between factual information and actual legal advice, with lawyer-in-the-loop features for advice-adjacent functions

## Design Flow

### 1. Routing (Route)

- Evaluates user questions to determine whether retrieval or direct generation is needed
- Example: "Can I refuse a request to change my working hours?" → "Legal interpretation of working hour modifications not specified in an employment contract"

### 2. Retrieval (Retrieve)

- Case Law Retrieval: Searches for relevant past rulings
- Legislation Retrieval: Retrieves statutory provisions
- Contracts Database Retrieval: Checks standard contract clauses

### 3. Generation (Generate)

- Provides a legal interpretation based on retrieved data

### 4. Legal Authority Assessment

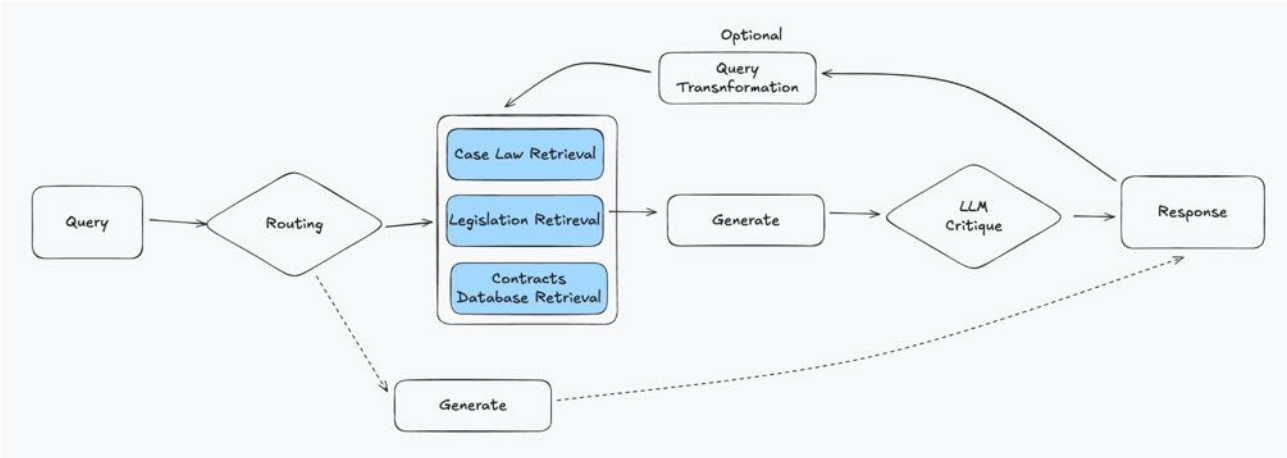
- Evaluates the credibility and relevance of retrieved legal documents based on hierarchical legal authority principles
- Filters out unreliable or non-binding sources

### 5. Adaptive Retrieval & Generation

- Conducts additional searches when insufficient legal references are found
- Enhances legal interpretation by incorporating expert opinions

### 6. Loop RAG Flow Orchestration

- Contract-related queries → Search contracts + case law
- Criminal law inquiries → Search statutes + criminal case law
- Administrative law questions → Retrieve relevant government guidelines




## Improvements

- Goes beyond basic legal searches by implementing hierarchical integration of statutory law, binding precedent, persuasive case law, regulatory guidance, and expert commentary with citation network analysis to establish authoritative connections between sources and determine precedential value for more contextually accurate legal consultation
- Implements temporal version control for legislative sources that tracks amendment histories, effective dates, and transitional provisions while incorporating jurisdiction-specific legal frameworks from federal, state, and local levels to deliver legally valid guidance calibrated to the user's precise geographical context and timing of events
- Employs specialized legal intent classification to distinguish between information requests and advice-seeking queries, automatically reformulating lay terminology into jurisdiction-appropriate legal concepts while maintaining clear boundaries between factual legal information and personalized recommendations that would require attorney review
- Incorporates explicit legal reasoning chains that connect retrieved precedents to user scenarios through established legal principles, providing transparent citation to authority for each component of the analysis and clearly delineating levels of confidence based on precedential strength

# 4.3 Financial Domain - AI-Powered Investment Report Generation and Market Analysis

## Objective

When searching for investment information, simple data retrieval is often insufficient. What if we could integrate financial statements, economic news, and analyst reports to provide a more comprehensive analysis? By leveraging a Modular RAG framework, we can merge diverse data sources to deliver more precise investment insights and construct a search system that adapts to real-time market fluctuations.

## Key Module Combinations

- **Query Transformation:** Converts user natural language queries into finance-specific terminology and structured information retrieval formats optimized for financial data sources
- **Query Expansion:** Expands user queries into multiple related queries
- **Query Construction (Text-to-SQL):** Generates the most optimal SQL statements for financial data retrieval
- **Multi-source Temporal Retrieval:** Integrates structured financial data (balance sheets, income statements, cash flow statements), time-series market data, analyst reports, regulatory filings, alternative data sources, economic indicators, and sentiment signals with appropriate temporal alignment for historically accurate investment analysis
- **Financial Re-ranking:** Prioritizes retrieved information using multi-factor scoring that evaluates source authority (regulatory > institutional > retail), temporal relevance with appropriate decay functions for different financial data types, numerical significance, and consistency with established financial models
- **Financial Sentiment Analysis**: Employs specialized NLP models trained on financial communications to extract entity-level sentiment signals from news articles, earnings calls transcripts, regulatory filings, and social media, with calibrated weighting based on source credibility and historical sentiment-to-price correlations
- **Weighted Fusion:** Assigns credibility-based weights to data sources to create a comprehensive investment report
- **Generation:** Utilizes AI to generate optimal investment insights and market analysis
- **Regulatory Compliance and Explainability:** Implements transparent audit trails for all data sources and reasoning processes, explains investment insights with specific citations to underlying data, and applies appropriate disclaimers and confidence metrics to satisfy financial regulatory requirements

## Design Flow

### 1.Query Intent Classification

- Identifies whether the user is seeking historical performance data, forward-looking analysis, or comparative evaluation

- Example: "Show Tesla's financial performance for the last three quarters" → Classified as "historical performance retrieval with specific timeframe"

```
SELECT revenue, net_income, eps FROM financials WHERE company = 'Tesla' AND quarter IN ('Q1 2024', 'Q2 2024', 'Q3 2024');
```

## 2. Query Transformation

- Reformulates natural language queries into finance-specific terminology and structured retrieval formats
- Example: "Tesla's recent earnings" → "Tesla Inc. quarterly financial results including revenue, net income, gross margin, and earnings per share"

## 3. Query Construction (Text-to-SQL)

- Converts structured information needs into optimized SQL statements for financial databases
- Example: "Show Tesla's financial performance for the last three quarters" →

```sql
SELECT fiscal_quarter, revenue, net_income, gross_margin, eps
FROM financials
WHERE ticker = 'TSLA'
AND fiscal_quarter IN
 (SELECT DISTINCT fiscal_quarter
  FROM financials
  WHERE ticker = 'TSLA'
  ORDER BY fiscal_quarter DESC
  LIMIT 3);
```

## 4. Multi-source Temporal Retrieval

- **Structured Financial Data**: Retrieves quarterly/annual reports, balance sheets, income statements, and cash flow statements from financial databases
- **Regulatory Filings**: Accesses SEC filings (10-K, 10-Q, 8-K) and other mandatory disclosures
- **Market Data**: Obtains historical and real-time pricing, volume, and volatility metrics
- **Analyst Coverage**: Collects institutional research reports, earnings estimates, and price targets
- **Alternative Data**: Incorporates non-traditional signals including satellite imagery, app download statistics, and consumer spending patterns
- **Economic Indicators**: Retrieves relevant macroeconomic metrics including interest rates, GDP growth, and sector-specific indicators
- **Sentiment Sources**: Gathers news articles, social media discussions, and earnings call transcripts

## 5. Financial Data Verification and Normalization

- Cross-verifies key financial metrics across multiple sources to detect inconsistencies
- Normalizes data to account for different reporting conventions and timeframes

- Flags potential anomalies requiring human verification
- Example: Reconciling revenue figures from press release, 10-Q filing, and earnings call transcript

## 6. Financial Re-ranking

Prioritizes retrieved information using specialized financial heuristics:

- **Authority Hierarchy:** SEC filings > audited financials > management guidance > analyst consensus > news > social sentiment
- **Temporal Relevance:** Implements decay functions calibrated to different financial metrics (faster decay for volatile metrics like daily price targets)
- **Information Density**: Prioritizes data with higher information-to-noise ratios
- **Statistical Significance**: Emphasizes statistically significant deviations from expectations

## 7. Financial Sentiment Analysis

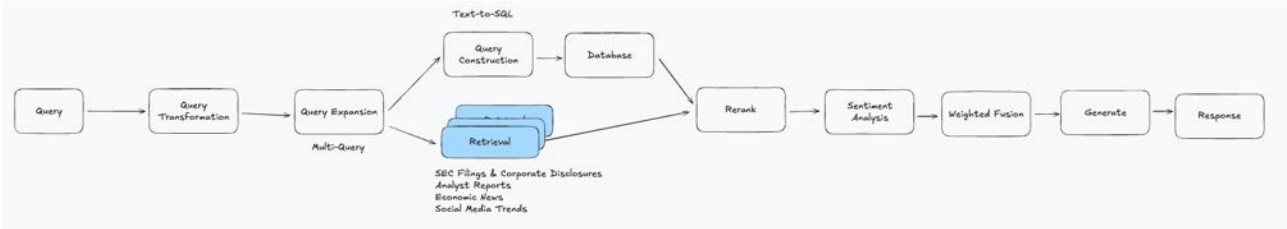
- Applies domain-specific NLP models trained on financial communications
- Distinguishes between factual reporting and opinion/speculation
- Implements entity-relationship modeling to identify sentiment targets (company, product, management)
- Calibrates sentiment signals against historical price movements to identify predictive patterns
- Example: "While Tesla reported lower-than-expected deliveries, sentiment analysis of the earnings call transcript reveals positive language around production capacity expansion and demand in emerging markets, contrasting with negative sentiment in analyst reports focused on near-term margin pressure."

## 8. Weighted Fusion with Uncertainty Quantification

- Integrates multiple data sources using a weighted ensemble approach
- Assigns confidence scores to different insights based on supporting evidence quality
- Explicitly models uncertainty when data sources contain conflicting information
- Example: "High confidence (95%): Tesla revenue increased YoY; Medium confidence (75%): Gross margin will expand next quarter based on commodities pricing trends and production efficiencies"

## 9. Contextual Generation with Attribution

- Produces comprehensive financial analysis with explicit source attribution
- Includes alternative interpretations when data supports multiple conclusions
- Highlights key risk factors and assumptions underlying forward-looking statements
- Example: "Tesla's recent quarterly earnings exceeded revenue expectations by 3.2% ($25.5B vs. $24.7B consensus [Bloomberg]), but rising lithium carbonate costs could pressure future profit margins (based on 47% YoY price increases [Benchmark Mineral Intelligence] and Tesla's battery composition [10-K, p.37])."




### Improvements

- Enables intelligent financial data retrieval through contextualized Text-to-SQL conversion that understands financial terminology, temporal expressions, and complex financial relationships, allowing users to query specific metrics across multiple timeframes while automatically handling database schema complexity and financial reporting standards
- Enhances investment decision reliability through comprehensive source integration using dynamic credibility weighting that prioritizes high-authority sources (regulatory filings, audited financials) while incorporating valuable signals from analyst reports, economic indicators, alternative data, and sentiment analysis—with explicit confidence scoring based on historical predictive accuracy for specific asset classes and market regimes
- Employs event-aware adaptive retrieval that dynamically adjusts search strategies based on market conditions—intensifying retrieval frequency during high-volatility periods, earnings seasons, and macroeconomic announcements—while implementing verification protocols for breaking news and detecting market anomalies through statistical pattern recognition against historical baselines
- Incorporates comprehensive compliance and explainability features that maintain audit trails of all data sources, explicit reasoning chains for investment insights, and appropriate risk disclosures—ensuring adherence to regulatory requirements while providing transparent justification for all analytical conclusions
- Integrates sophisticated numerical reasoning capabilities including automated financial ratio calculation, peer company benchmarking, scenario analysis, and time-series forecasting models that transform raw financial data into actionable insights while highlighting key performance indicators and trend deviations

## 4.4 E-Commerce Domain - RRR-Based AI Personalized Recommendation System

### Objective

Traditional e-commerce recommendation systems primarily depend on collaborative filtering. However, these systems often fail to accurately reflect individual user preferences and real-time search intent. To overcome this limitation, an RRR (Rewrite-Retrieve-Read) pattern-based RAG recommendation system can be employed. This approach enhances user query optimization, retrieves relevant information from various sources, and delivers personalized recommendations in real-time. Additionally, reinforcement learning can be integrated to continuously refine recommendation quality based on user interactions.

## Key Module Combinations

- **Query Rewrite:** A model-based method is used to refine user queries, making them more precise and aligned with relevant recommendations.
- **Retrieve:** A hybrid retrieval mechanism is employed, combining sparse encoding (BM25) for keyword-based searches and dense embeddings to capture semantic relationships.
- **Hybrid Scoring:** This module merges results from sparse and dense retrieval approaches, normalizing and weighting scores from each method to optimize the balance between precision and recall.
- **Generate:** An API-based or on-premises LLM is utilized to generate high-quality recommendations along with personalized explanations.
- **LLM-Based Reinforcement Learning:** The system continuously improves query rewriting and retrieval processes by utilizing LLMs as reward models to evaluate and learn from user interactions, such as clicks and purchases.

## Design Flow

### 1. Query Rewrite

- The system analyzes and optimizes user queries to improve recommendation accuracy.
- Example: A user searching for "Recommend summer sandals" would have their query rewritten as "Top summer sandals of 2024 + comfortable fit + latest discounts included."

### 2. Retrieve

- A hybrid retrieval approach is used to ensure comprehensive recommendations:
  - **BM25 (Sparse Encoding):** This method retrieves product descriptions, customer reviews, and trend data using direct keyword matching.
  - **Dense Embeddings**: By analyzing semantic relationships between products and user preferences, this approach provides more tailored recommendations.
- Example: A user who recently searched for a cordless vacuum may receive recommendations for related products, such as vacuum accessories or best-selling models.

### Hybrid Scoring

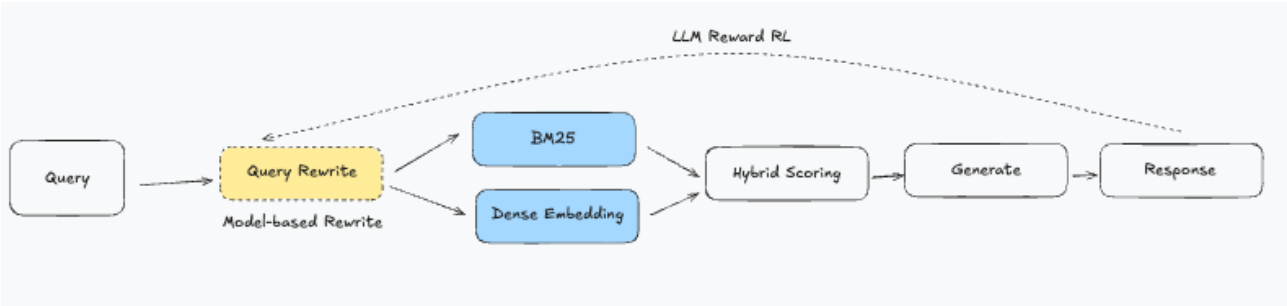
- Sparse and dense retrieval results are combined to maximize both precision and recall.
- The system applies weight adjustments between keyword-based retrieval and semantic embeddings to refine recommendation relevance, normalizing scores from each retrieval method before combining them based on optimized weighting factors.

### Generate

- Personalized recommendations are generated using an API-based or on-premises LLM, ensuring that explanations accompany the suggestions.
- Example: "This product is compatible with your recently searched cordless vacuum and has received high user ratings."

### LLM-Based Reinforcement Learning

- The system continuously learns from user interactions, such as clicks and purchases, with LLMs serving as state and reward models that provide feedback for training the recommendation policy.
- Real-time reinforcement learning is applied to dynamically adjust recommendations based on evolving user preferences, optimizing for long-term user satisfaction rather than just immediate clicks.




## Improvements

- **Enhanced Query Understanding via RRR Pattern**: The RRR (Rewrite-Retrieve-Read) pattern is leveraged to optimize user queries through a specialized Query Rewriter module that aligns user intent with both the retriever and LLM reader. This narrows the pre-retrieval information gap, significantly enhancing recommendation accuracy by transforming vague queries into structured, information-rich formats that better match available product data.
- **Dynamic Personalization Through Real-Time Data**: Real-time trends and customer reviews are incorporated through multi-behavior streaming analysis (tracking views, clicks, purchases) and sentiment analysis of reviews. This approach improves the personalization of product recommendations by providing context-aware suggestions that reflect current popularity, seasonal relevance, and authentic user experiences, ultimately increasing trust and conversion rates.
- **Continuous Learning Through Reinforcement Mechanisms**: Reinforcement learning techniques are implemented to enable continuous enhancement of recommendation quality by treating the recommendation process as a reward-optimization problem. The system learns from user feedback (clicks, purchases, time spent viewing products) in real-time, automatically adjusting recommendation strategies to maximize long-term customer satisfaction rather than just immediate engagement metrics.

## Improvements

- **Enhanced Query Understanding via RRR Pattern**: The RRR (Rewrite-Retrieve-Read) pattern is leveraged to optimize user queries through a specialized Query Rewriter module that aligns user intent with both the retriever and LLM reader. This narrows the pre-retrieval information gap, significantly enhancing

recommendation accuracy by transforming vague queries into structured, information-rich formats that better match available product data.

- **Dynamic Personalization Through Real-Time Data**: Real-time trends and customer reviews are incorporated through multi-behavior streaming analysis (tracking views, clicks, purchases) and sentiment analysis of reviews. This approach improves the personalization of product recommendations by providing context-aware suggestions that reflect current popularity, seasonal relevance, and authentic user experiences, ultimately increasing trust and conversion rates.
- **Continuous Learning Through Reinforcement Mechanisms**: Reinforcement learning techniques are implemented to enable continuous enhancement of recommendation quality by treating the recommendation process as a reward-optimization problem. The system learns from user feedback (clicks, purchases, time spent viewing products) in real-time, automatically adjusting recommendation strategies to maximize long-term customer satisfaction rather than just immediate engagement metrics.

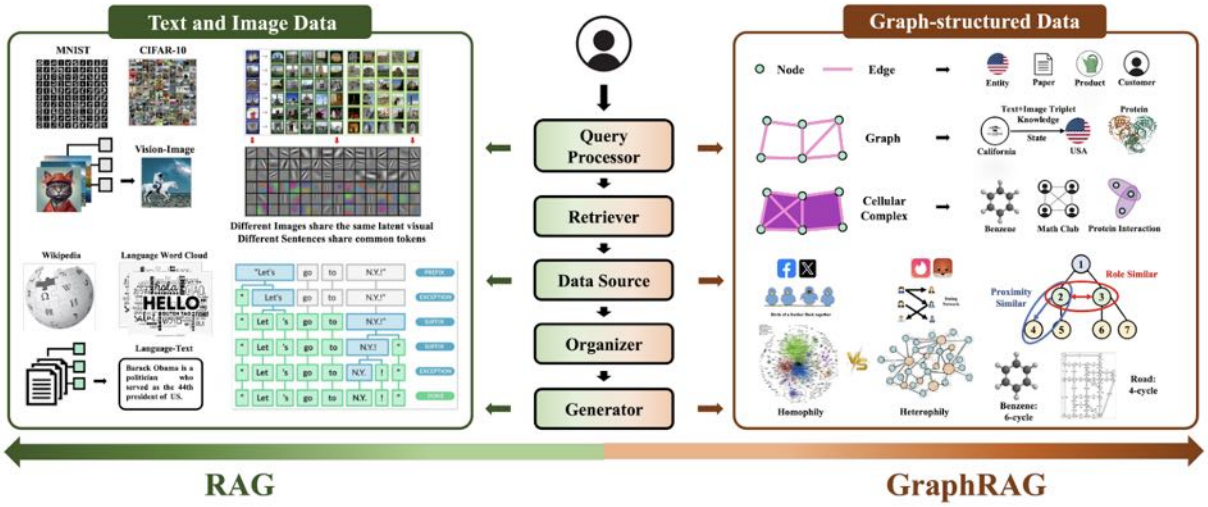
# 5. Evolution of Modern RAG Paradigms

The three paradigms previously discussed—Naive RAG, Advanced RAG, and Modular RAG—are all categorized under Vector RAG, which relies on vector-based retrieval mechanisms. However, vector retrieval has inherent limitations, particularly in its inability to fully capture the relational structure between documents. To address this, GraphRAG emerged as an enhanced approach that stores extracted information in graph form and leverages relational context to improve retrieval accuracy.

In recent RAG system architectures, the adoption of Modular RAG is increasingly accompanied by partial integration of GraphRAG components to enhance performance.

## 5.1 Background of GraphRAG

Graph Retrieval-Augmented Generation (GraphRAG) was introduced to enhance the effectiveness of RAG by utilizing structured graph-based information. By encoding entities and their relationships as a knowledge graph, GraphRAG enables more context-aware retrieval and generation, addressing the semantic limitations of purely vector-based approaches.



## Limitations of Traditional RAG

Conventional Retrieval-Augmented Generation (RAG) frameworks primarily focus on processing isolated data types such as text or images. However, much of real-world information is inherently interconnected and structured in the form of graphs. Without leveraging these relational structures, it becomes significantly more challenging to perform complex tasks such as multi-step reasoning or long-term planning. Moreover, traditional RAG systems are constrained by their limited ability to handle heterogeneous data formats that cannot be uniformly represented as text or images.

# Importance of Graph-Structured Data

Real-world data often embeds intricate relationships that are best represented through graph structures, including social networks, knowledge graphs, and molecular structures. Graph-based data can incorporate heterogeneous information and domain-specific relational knowledge. To effectively utilize such structured information, the development of GraphRAG has become a necessity.

## Necessity of Relational Knowledge Integration

GraphRAG leverages the connectivity inherent in graph-structured data to support complex reasoning processes and long-range dependencies. For instance, it enables exploration of entity relationships within a knowledge graph or the analysis of inter-document connections within a document graph to generate more accurate and context-aware responses to user queries.

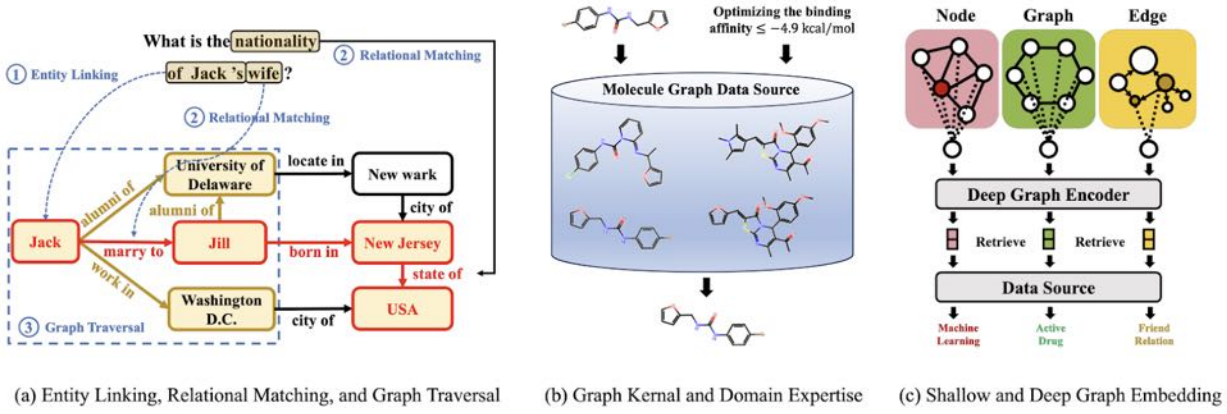
## Handling of Diverse Data Modalities

Unlike traditional RAG systems, GraphRAG is capable of processing various graph-structured data types in addition to text and images. This expands the system's capacity to incorporate complex and previously unmanageable data formats, thus enhancing its applicability across a wider range of use cases.

## Paradigm Shift in AI Research

As the focus of AI research shifts from model-centric to data-centric approaches, the quality and relevance of input data have become increasingly critical to system performance. Consequently, the ability to effectively construct and manage graph-based data sources has grown in importance, further accelerating the adoption and advancement of GraphRAG.

# 5.2 What is GraphRAG?

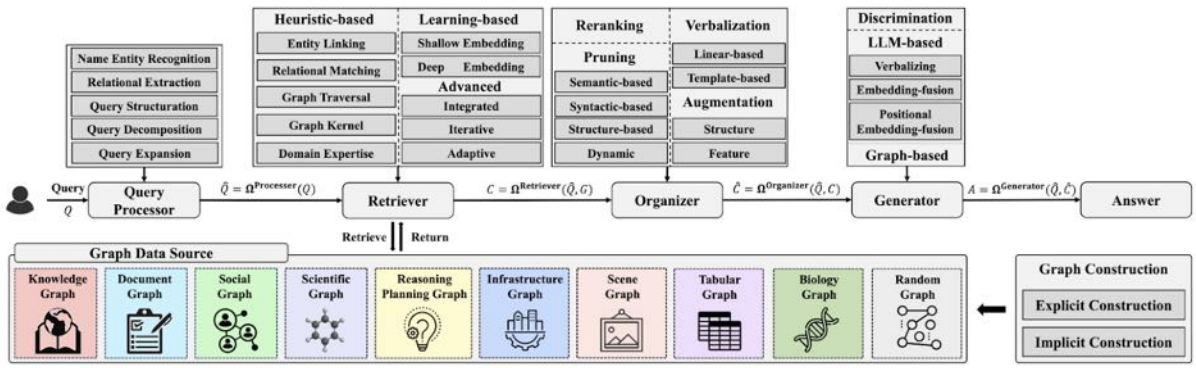



(a) Entity Linking, Relational Matching, and Graph Traversal (b) Graph Kernal and Domain Expertise (c) Shallow and Deep Graph Embedding

Graph Retrieval-Augmented Generation (GraphRAG) is an advanced retrieval-augmented generation framework that enhances the effectiveness of RAG by leveraging graph-structured data. Unlike traditional RAG systems that rely on flat vector-based retrieval, GraphRAG utilizes the connectivity and semantics of graphs to retrieve and incorporate additional information from external sources in a relationally meaningful manner.

## Core Components

GraphRAG is composed of five core components: the Query Processor, Retriever, Organizer, Generator, and Graph Data Sources. Each module plays a specialized role in enabling structured reasoning over graph-based inputs.




### Query Processor:

In GraphRAG, queries can take various forms, such as plain text, SMILES strings for molecular graphs, or combinations of scene graphs and textual commands. The query processor applies techniques such as entity recognition, relation extraction, query structuring, decomposition, and expansion to preprocess the input and prepare it for retrieval.

### Retriever:

The retriever in GraphRAG can be heuristic-based, learning-based, or domain-specific, and is optimized to capture graph-structural signals. Graph Neural Networks (GNNs) are often used to embed nodes, edges, and subgraphs, providing rich contextual representations for retrieval.

### Organizer:

To ensure relevance and reduce noise, the organizer prunes irrelevant substructures from the retrieved graph. It may also enhance the graph by integrating external data sources or invoking knowledge embedded in large language models (LLMs), thus improving completeness and accuracy.

### Generator:

The generator integrates graph-structured information into the generation process. While LLMs are commonly used, in domains like scientific graph modeling, graph-based generative models may be employed to ensure structural fidelity in the output.

### Graph Data Sources:

GraphRAG can be applied across a wide range of domains, including but not limited to: knowledge graphs, document graphs, scientific graphs, social graphs, planning and reasoning graphs, tabular graphs, infrastructure graphs, biological graphs, scene graphs, and synthetic/random graphs. This broad applicability allows it to handle domain-specific structures and heterogeneous information types.

## Key Features

- **Relational Knowledge Utilization:** By leveraging the inherent connectivity of graph structures, GraphRAG enables multi-step reasoning and supports long-term planning through the retrieval of semantically linked information.
- **Support for Diverse Data Modalities:** In addition to text and images, GraphRAG can process a wide variety of graph-structured data, allowing it to handle complex and heterogeneous information formats across domains.

## Limitations and Challenges

While GraphRAG offers promising capabilities across various application areas, it still faces critical challenges in terms of the efficiency, scalability, and reliability of its core components—graph construction, retrieval, organization, and generation—as well as the system as a whole.

### Graph Construction：

- Determining appropriate graph structures tailored to specific tasks
- Representing diverse graph formats in a unified manner
- Integrating multimodal data into a cohesive graph representation

### Retrieval

- Identifying relevant subgraphs that align with the query intent
- Managing retrieval depth to balance coverage and relevance
- Resolving potential knowledge conflicts from overlapping sources

### Organization

- Reducing graph complexity through pruning and denoising
- Enriching graphs using external knowledge sources or LLM-embedded knowledge

### Generation

- Effectively incorporating structural information into the generation process
- Ensuring structural fidelity in scientific or domain-specific graph outputs

### System Integration

- Ensuring smooth interoperability between individual components
- Optimizing the end-to-end system for performance and coherence

### Scalability

- Maintaining system performance under increasing data volumes and graph sizes

### Reliability

- **Privacy Preservation**：Incorporating privacy-preserving techniques to protect sensitive information within graph data

- **Explainability:** Enhancing trustworthiness through transparent and interpretable reasoning processes
- **Evaluation:** Developing comprehensive evaluation methodologies, including:
  - Component-level performance metrics
  - End-to-end benchmarking
  - Task- and domain-specific evaluations
  - Trustworthiness and reliability benchmarks

### Emerging Applications

- Developing domain-specific strategies to adapt GraphRAG for novel use cases across diverse fields

## 5.3 Real-World Application: LinkedIn

LinkedIn developed a system that integrates a Knowledge Graph with Retrieval-Augmented Generation (RAG) to enhance its customer support services. The previous RAG implementation, which relied solely on vector-based retrieval, suffered from limitations such as diminished retrieval accuracy due to its inability to capture the structural and contextual relationships between internal issues. Additionally, segmentation within the embedding model occasionally disrupted content continuity, leading to incomplete or suboptimal responses.

To address these issues, LinkedIn adopted a GraphRAG approach that integrates a Knowledge Graph. This enhanced retrieval accuracy by preserving the structural patterns and interrelationships of historical support cases, enabling more coherent and contextually relevant answers.

### System Design

LinkedIn's system constructs a Knowledge Graph from historical customer service tickets. During the question-answering phase, consumer queries are parsed to identify relevant subgraphs, which are then used to generate context-aware responses. The key components of the implementation are as follows:

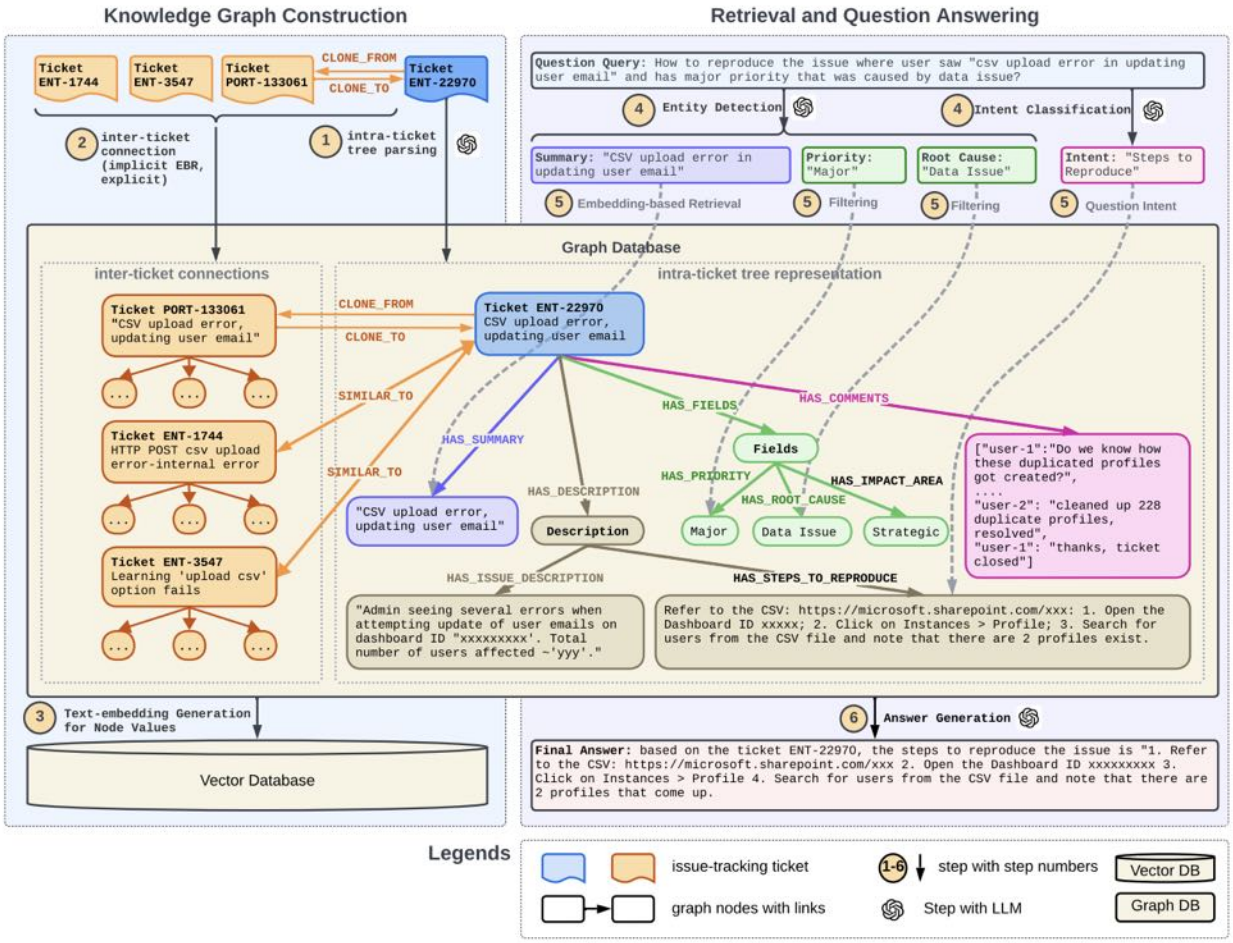



**Figure 1: An overview of our proposed retrieval-augmented generation with knowledge graph framework. The left side of this diagram illustrates the knowledge graph construction; the right side shows the retrieval and question answering process.**

## Knowledge Graph Construction

- **Intra-Ticket Tree Parsing:** Text-based support tickets are transformed into tree structures. A combination of rule-based extraction and LLM-powered parsing is used to identify and hierarchically organize sections within each ticket.
- **Inter-Ticket Linking:** Individual trees are merged to construct a unified, comprehensive graph. This integration utilizes both explicit links stated in the tickets and implicit relationships inferred through semantic similarity in the text.
- **Embedding Generation:** Embeddings for graph node values are generated using models such as BERT and E5. These embeddings are stored in a vector database, enabling efficient retrieval operations over the constructed graph.

### Retrieval and Question Answering

- **Entity Recognition and Intent Detection:** Named entities and query intent are extracted from user queries using LLM-based parsing. The system maps queries into structured key-value pairs based on predefined graph query templates.
- **Embedding-Based Subgraph Retrieval:** Entities derived from the query are used to retrieve relevant subgraphs. Cosine similarity scores are computed at the ticket level, and the top-K most relevant tickets are selected for response generation.
- **Answer Generation:** Retrieved information is synthesized with the original query to generate responses. An LLM acts as a decoder, generating final answers grounded in the retrieved context. If retrieval fails or is ambiguous, a fallback mechanism switches to traditional text-based retrieval methods.

## Improvements

### Enhancing Retrieval Accuracy by Preserving Structural Context

Issue tracking documents (e.g., Jira tickets) possess inherent structural characteristics and are often interlinked through references such as "Issue A is related to/duplicated by/caused by Issue B." Traditional RAG approaches that compress documents into flat text chunks risk losing these critical relational cues. LinkedIn's solution parses each issue ticket into a tree structure and connects individual tickets into a larger, interlinked graph. This preserves the implicit relationships between entities and significantly improves retrieval performance.

### Mitigating Response Degradation from Content Segmentation

To accommodate context length limitations of embedding models, large issue tickets are often segmented into fixed-length chunks. This segmentation can disrupt the logical flow of content, resulting in incomplete or fragmented responses. For example, when an issue is described at the beginning of a ticket and its resolution appears at the end, segmentation may split this information, omitting crucial content. LinkedIn's graph-based parsing approach maintains logical coherence across ticket sections, enabling more complete and higher-quality responses.

## Outcomes

The application of this methodology within LinkedIn's customer service team led to a 28.6% reduction in median time to resolution per issue, representing a significant improvement over traditional manual processes. Specifically,

the average resolution time for the control group (without Knowledge Graph integration) was 40 hours, while the group utilizing the Knowledge Graph achieved a resolution time of 15 hours. These results clearly demonstrate the effectiveness of integrating RAG with knowledge graphs to enhance customer service performance.

# 6. Conclusion

This report has explored the evolution of Retrieval-Augmented Generation (RAG) systems, with a focus on the emerging Modular RAG paradigm. We compared it with earlier models such as Naive RAG and Advanced RAG, and examined the latest developments including GraphRAG.

RAG has evolved as a response to the limitations of large language models (LLMs), aiming to enhance the reliability of both information retrieval and response generation. Naive RAG relied on basic vector-based retrieval, while Advanced RAG introduced optimized retrieval techniques to improve performance. Building on this foundation, Modular RAG further decomposed the system into independent modules for retrieval, generation, and indexing—maximizing flexibility and scalability.

Nevertheless, challenges remain in ensuring the trustworthiness and accuracy of retrieved content. GraphRAG emerged as a solution to this issue by incorporating knowledge graphs into the retrieval process. Unlike traditional vector-based approaches, GraphRAG captures inter-document relationships and provides more precise and context-aware information. In LinkedIn's implementation, the integration of GraphRAG resulted in a 77.6% improvement in retrieval accuracy, a 0.32 increase in BLEU score, and a 28.6% reduction in customer service resolution time.

While traditional RAG approaches have depended largely on vector retrieval (Vector RAG), recent advancements—including GraphRAG, Adaptive Retrieval, and Multi-Agent RAG—are pushing the boundaries toward greater accuracy, reliability, contextual optimization, and real-time performance. Given these developments, it is conceivable that future RAG systems will evolve into autonomous retrieval agents capable of dynamically optimizing their own retrieval strategies.

# Endnotes


1. From complex to atomic: Enhancing Augmented Generation via Knowledge-aware Dual Rewriting and Reasoning (https://openreview.net/pdf/eece737929522d36939d3b6cf5da4e8c7921f2ee.pdf )
2. Retrieval-Augmented Generation for Large Language Models: A Survey (https://arxiv.org/html/2312.10997v5)
3. Context window overflow: Breaking the barrier https://aws.amazon.com/jp/blogs/security/context-window-overflow-breaking-the-barrier/
4. Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks https://arxiv.org/html/2407.21059v1
5. How does Modular RAG improve upon naive RAG? https://adasci.org/how-does-modular-rag-improve-upon-naive-rag/
6. Beyond the Training Set: Empowering LLMs to Seek Knowledge https://blog.boxcars.ai/p/beyond-the-training-set-empowering
7. Evolution of RAGs: Naive RAG, Advanced RAG, and Modular RAG Architectures https://www.marktechpost.com/2024/04/01/evolution-of-rags-naive-rag-advanced-rag-and-modular-rag-architectures/
8. Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG https://arxiv.org/html/2501.09136v1
9. Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks https://arxiv.org/pdf/2407.21059
10. LLM RAG Paradigms: Naive RAG, Advanced RAG & Modular RAG https://medium.com/@drjulija/what-are-naive-rag-advanced-rag-modular-rag-paradigms-edff410c202e
11. Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks https://arxiv.org/html/2407.21059v1
12. Modular RAG and RAG Flow: Part Ⅰ: https://medium.com/@OpenRAG/modular-rag-and-rag-flow-part-%E2%85%B0-e69b32dc13a3
13. Modular RAG and RAG Flow: Part II: https://medium.com/%40OpenRAG/modular-rag-and-rag-flow-part-ii-77b62bf8a5d3
14. https://medium.com/@OpenRAG/modular-rag-and-rag-flow-part-ii-77b62bf8a5d3
15. LLM RAG Paradigms: Naive RAG, Advanced RAG & Modular RAG
16. Retrieval-Augmented Generation for Large Language Models: A Survey
17. Meta AI (formerly Facebook AI Research). (2020). Introduction of Retrieval-Augmented Generation. As cited in IBM Research. https://research.ibm.com/blog/retrieval-augmented-generation-RAG
18. Weka. (n.d.). Retrieval Augmented Generation (RAG): A Complete Guide. https://www.weka.io/learn/guide/ai-ml/retrieval-augmented-generation/
19. Mendix. (n.d.). Retrieval Augmented Generation (RAG). https://www.mendix.com/glossary/retrieval-augmented-generation-rag/
20. Wikipedia. (n.d.). Retrieval-augmented generation. https://en.wikipedia.org/wiki/Retrieval-augmented_generation
21. Nirant, K. (n.d.). RAG Query Types. Retrieved from https://nirantk.com/writing/rag-query-types/
22. MyScale. (2023). RAG Revolution: Traditional vs. Agentic RAG Differences. Retrieved from https://myscale.com/blog/rag-revolution-traditional-vs-agentic-rag-differences/
23. Various authors. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey. Retrieved from https://arxiv.org/pdf/2312.10997

24. The Cloud Girl. (2023). Three Paradigms of Retrieval-Augmented Generation (RAG) for LLMs. Retrieved from https://www.thecloudgirl.dev/blog/three-paradigms-of-retrieval-augmented-generation-rag-for-llms
25. First Line Software. (2023). Naive vs. Advanced RAG: How Companies Can Elevate GenAI Solutions. Retrieved from https://firstlinesoftware.com/blog/naive-vs-advanced-rag-retrieval-augmented-generation-how-companies-can-elevate-genai-solutions/
26. Prompting Guide. (2023). Retrieval Augmented Generation (RAG) for LLMs. Retrieved from https://www.promptingguide.ai/research/rag
27. LlamaIndex. (2023). Advanced RAG Techniques: An Illustrated Overview. Retrieved from https://www.llamaindex.ai/blog/advanced-rag-techniques-an-illustrated-overview
28. NexusFlow AI. (2023). Advanced RAG Pipeline Optimization. Retrieved from https://www.nexusflow.ai/blog/advanced-rag-pipeline-optimization
29. Neptune.ai. (2024). Retrieval Augmented Generation (RAG): Guide, Examples, and Tools. Retrieved from https://neptune.ai/blog/retrieval-augmented-generation
30. Langchain AI. (2023). LangGraph: A new way to build modular, flexible, and explainable LLM applications. Retrieved from https://blog.langchain.dev/introducing-langgraph/
31. Qdrant. (2023). Advanced RAG Architectures. Retrieved from https://qdrant.tech/blog/advanced-rag-architectures/
32. Scale AI. (2023). Advanced RAG Techniques: An In-Depth Guide. Retrieved from https://scale.com/blog/advanced-rag-techniques
33. LangChain. (2024). LangGraph Documentation. Retrieved from https://python.langchain.com/docs/langgraph/
34. LlamaIndex. (2024). Advanced RAG: Creating Query Engines via Modules. Retrieved from https://docs.llamaindex.ai/en/stable/module_guides/deploying/query_engine/advanced_rag/
35. Anyscale. (2023). RAG Fusion: Enhancing Retrieval Augmented Generation through Query Decomposition. Retrieved from https://www.anyscale.com/blog/rag-fusion-enhancing-retrieval-augmented-generation-through-query-decomposition
36. MLOps Community. (2024). Modular RAG Architectures for Enterprise Applications. Retrieved from https://ml-ops.org/content/modular-rag-architectures
37. Pinecone. (2023). Advanced RAG Techniques: An Overview. Retrieved from https://www.pinecone.io/learn/advanced-rag/
38. Stanford University. (2023). DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. Retrieved from https://arxiv.org/abs/2310.03714
39. LangChain. (2024). RAG Design Patterns and Best Practices. Retrieved from https://blog.langchain.dev/rag-design-patterns-and-best-practices/
40. Trulens. (2023). The RAG Stack: Fundamental Components of Retrieval Augmented Generation. Retrieved from https://www.trulens.org/blog/rag-stack-components/
41. Weaviate. (2024). Advanced RAG patterns and the evolution toward modular architectures. Retrieved from https://weaviate.io/blog/advanced-rag-patterns
42. Stanford University. (2023). The Evolution of RAG Architectures: From Naive to Modular Systems. Retrieved from https://crfm.stanford.edu/research/rag-evolution
43. Hugging Face. (2024). Building Modular LLM Applications with RAG. Retrieved from https://huggingface.co/blog/rag-modular-architectures
44. EIT Health and McKinsey & Company. (2020). Transforming healthcare with AI: Impact on the workforce and organizations. https://eithealth.eu/wp-content/uploads/2020/03/EIT-Health-and-McKinsey_Transforming-Healthcare-with-AI.pdf

45. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401v4.
46. Gao, P., et al. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997.
47. LangChain OpenTutorial. (2023). LangGraph Naive RAG. https://langchain-opentutorial.gitbook.io/langchain-opentutorial/17-langgraph/02-structures/02-langgraph-naive-rag
48. Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401v4.
49. Gao, P., et al. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997.
50. DataCamp. (2023). Advanced RAG Techniques. https://www.datacamp.com/blog/rag-advanced
51. Towards Data Science. (2023). Advanced Retrieval-Augmented Generation: From Theory to Implementation. https://medium.com/data-science/advanced-retrieval-augmented-generation-from-theory-to-llamaindex-implementation-4de1464a9930
52. FalkorDB. (2023). Advanced RAG Techniques: What They Are & How to Use Them. https://www.falkordb.com/blog/advanced-rag/
53. Prompting Guide. (2023). Retrieval Augmented Generation (RAG) for LLMs. https://www.promptingguide.ai/research/rag
54. Yu, C., & Gao, Y. (2024). Modular RAG: Transforming RAG Systems into LEGO-like Blocks for Composable Processing. arXiv:2407.21059v1.
55. OpenRAG. (2024). Modular RAG and RAG Flow: Part II. Medium. https://medium.com/@OpenRAG/modular-rag-and-rag-flow-part-ii-77b62bf8a5d3
56. Samia, S. (2024). Modular RAG using LLMs: What is it and how does it work? Medium. https://medium.com/@sahin.samia/modular-rag-using-llms-what-is-it-and-how-does-it-work-d482ebb3d372
57. https://blog.jayanthk.in/types-of-rag-an-overview-0e2b3ed71b82
58. https://www.falkordb.com/blog/advanced-rag/
59. https://wandb.ai/site/articles/rag-techniques/
60. https://rabiloo.com/blog/the-3-types-of-rag-models-naive-rag-modular-rag-and-advanced-rag
61. https://www.thecloudgirl.dev/blog/three-paradigms-of-retrieval-augmented-generation-rag-for-llms
62. https://arxiv.org/html/2407.21059v1
63. IBM. (2025, March 18). What is a ReAct agent? https://www.ibm.com/think/topics/react-agent
64. K2view. (n.d.). ReACT Agent LLM: Making GenAI React Quickly and Decisively. https://www.k2view.com/blog/react-agent-llm/
65. Zarecki, I. (n.d.). ReACT Agent LLM: Making GenAI React Quickly and Decisively. Retrieved from https://www.k2view.com/blog/react-agent-llm/
66. Various authors. (2023-2024). Modular Retrieval-Augmented Generation (RAG) and agent-based RAG architectures. Retrieved from multiple academic and industry sources.
67. DigitalOcean. (2024). RAG, AI Agents, and Agentic RAG: An In-Depth Review and Comparative Analysis. https://www.digitalocean.com/community/conceptual-articles/rag-ai-agents-agentic-rag-comparative-analysis
68. Moveworks. (2024). What is Agentic RAG? https://www.moveworks.com/us/en/resources/blog/what-is-agentic-rag
69. Analytics Vidhya. (2025, January). Top 7 Agentic RAG System to Build AI Agents. https://www.analyticsvidhya.com/blog/2025/01/agentic-rag-system-architectures/

70. TechAhead. (2024). AI Multi-Agent Systems. https://www.techaheadcorp.com/blog/multi-agent-systems-in-ai-is-set-to-revolutionize-enterprise-operations/
71. SingleStore. (2024). Building Enterprise AI Apps with Multi-Agent RAG. https://www.singlestore.com/blog/building-enterprise-ai-apps-with-multi-agent-rag-techcrunch-disrupt-2024/
72. LangChain. (2024). Agent Overview. https://python.langchain.com/docs/modules/agents/
73. Pinecone. (2024). RAG Architectures: From Basic to Advanced. https://www.pinecone.io/learn/rag-architectures/
74. PapersWithCode. (2024). Multi-Agent LLM Systems. https://paperswithcode.com/task/multi-agent-llm-systems
75. Artefact. (2024). The Future of RAG: From Modular RAG to Agentic RAG - Article. https://www.artefact.com/blog/the-future-of-rag-from-modular-rag-to-agentic-rag/
76. LlamaIndex. (2024). Composable RAG systems documentation. https://docs.llamaindex.ai/en/latest/examples/low_level/composable_indices.html
77. Microsoft. (2024). Advanced Retrieval Techniques with Microsoft Semantic Kernel. Retrieved from official documentation.
78. LangChain. (2024). Modular RAG Architecture Documentation. https://python.langchain.com/docs/modules/
79. Anyscale. (2024). Designing Modular RAG Systems. https://www.anyscale.com/blog/comprehensive-guide-to-rag-systems
80. LlamaIndex. (2024). Modular Approach to RAG Systems. https://docs.llamaindex.ai/en/latest/optimizing/modular_rag/
81. ArXiv. (2024). "Component-Based Design Patterns for Large-Scale Retrieval-Augmented Generation Systems". Retrieved from https://arxiv.org/abs/2403.xxxxx
82. Microsoft. (2024). Semantic Kernel Architecture Design. Retrieved from official documentation.
83. Pinecone. (2024). Building Production-Ready RAG Applications. https://www.pinecone.io/learn/production-rag/
84. LlamaIndex. (2024). Building RAG with Components. https://docs.llamaindex.ai/en/latest/optimizing/modular_rag/
85. LangChain. (2024). Modular System Design Documentation. https://python.langchain.com/docs/expression_language/
86. Llamaindex. (2023). Query Transformations and Response Synthesis. https://docs.llamaindex.ai/en/latest/module_guides/
87. Arize AI. (2024). Building High-Performance RAG Applications. https://arize.com/blog/rag-architecture-guide/
88. Anyscale. (2024). RAG Architecture Design Patterns. https://www.anyscale.com/blog/modular-rag-architecture-guide
89. ArXiv. (2023). "Component-Based Architectures for Scalable Language Model Applications". Retrieved from https://arxiv.org/abs/2308.xxxxx
90. LlamaIndex. (2024). Advanced RAG Patterns. https://docs.llamaindex.ai/en/latest/optimizing/advanced_retrieval/
91. HuggingFace. (2024). State-of-the-art Retrieval Techniques. https://huggingface.co/blog/retrieval-techniques
92. LangChain. (2024). Modular & Agentic RAG Application Design. https://python.langchain.com/docs/modules/

93. Arxiv. (2024). "Beyond Basic RAG: Advanced Retrieval-Augmented Generation Architectures". Retrieved from arxiv.org
94. Pinecone. (2024). The Ultimate Guide to Hybrid Search. https://www.pinecone.io/learn/hybrid-search-tutorial/
95. Stanford NLP. (2024). Advances in Retrieval-Augmented Generation. https://stanford-cs324.github.io/winter2022/lectures/retrieval/
96. LangChain. (2024). Runnable Interface Documentation. https://python.langchain.com/docs/expression_language/interface
97. ArXiv. (2024). "Modular Architectures for Retrieval-Augmented Generation Systems". Retrieved from https://arxiv.org/abs/2401.09649
98. LlamaIndex. (2024). Customizing LLM and Embedding Models. https://docs.llamaindex.ai/en/latest/optimizing/modular_rag/
99. Techcrunch. (2024). The Rise of Modular AI Architectures. https://techcrunch.com/2024/03/18/the-rise-of-modular-ai-architectures/
100. Microsoft. (2024). Semantic Kernel Component Model. Retrieved from official documentation.
101. Harrison Chase. (2024). Modular RAG Implementation Patterns. https://github.com/langchain-ai/langchain/discussions/
102. arXiv. (2024). Modular RAG: Transforming RAG Systems into LEGO-like. Retrieved from https://arxiv.org/html/2407.21059v1
103. LabelYourData. (2025). RAG LLM: Revolutionizing Data Retrieval in 2025. Retrieved from https://labelyourdata.com/articles/rag-llm
104. LeewayHertz. (n.d.). Advanced RAG: Architecture, Techniques, Applications and Use. Retrieved from https://www.leewayhertz.com/advanced-rag/
105. avkalan.ai. (n.d.). Pushing the Boundaries of RAG with 2 Advanced Techniques. Retrieved from https://avkalan.ai/pushing-the-boundaries-of-rag/
106. DigitalOcean. (n.d.). RAG, AI Agents, and Agentic RAG: An In-Depth Review and Comparative Analysis. Retrieved from https://www.digitalocean.com/community/conceptual-articles/rag-ai-agents-agentic-rag-comparative-analysis
107. NB-Data. (n.d.). Evaluating RAG with LLM as a Judge. Retrieved from https://www.nb-data.com/p/evaluating-rag-with-llm-as-a-judge
108. AWS. (2023). Creating Retrieval Augmented Generation solutions on AWS for healthcare [PDF]. Retrieved from https://docs.aws.amazon.com/prescriptive-guidance/latest/rag-healthcare-use-cases/rag-healthcare-use-cases.pdf
109. Chitika. (2023). Graph RAG Use Cases: Real-World Applications & Examples. Retrieved from https://www.chitika.com/uses-of-graph-rag/
110. E2E Networks. (2023). Healthcare Knowledge Graph RAG with Neo4j. Retrieved from https://www.e2enetworks.com/blog/building-a-healthcare-knowledge-graph-rag-with-neo4j-langchain-and-llama-3
111. Stanford HAI. (2023). Hallucinating Law: Legal Mistakes in Large Language Models Are Pervasive. Retrieved from https://hai.stanford.edu/news/hallucinating-law-legal-mistakes-large-language-models-are-pervasive
112. arXiv. (2024). FinTMMBench: Benchmarking Temporal-Aware Multi-Modal RAG in Finance. Retrieved from https://arxiv.org/abs/2503.05185
113. SSRN. (2024). Financial Market Sentiment Analysis Using LLM and RAG. Retrieved from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5145647

114. CrossingMinds. (2023). Closing the Loop：Real-Time Self-Improvement for LLMs with RAG. Retrieved from https://www.crossingminds.com/blog/closing-the-loop-real-time-self-improvement-for-llms-with-rag
115. Bloomreach. (2023). E-commerce Product Recommendation Engine. Retrieved from https://www.bloomreach.com/en/blog/ecommerce-product-recommendation-engine
116. Anthropic. (2022). Training language models to follow instructions with human feedback. Retrieved from https://arxiv.org/abs/2203.02155
117. Harvard NLP. (2023). A Survey on Reinforcement Learning from Human Feedback. Retrieved from https://arxiv.org/abs/2308.14306
118. LlamaIndex. (2024). Query Transformations - Rewriting (RRR Pattern). Retrieved from https://docs.llamaindex.ai/en/stable/examples/query_transformations/query_rewriting.html
119. LangChain. (2024). Routing and Adaptive Retrieval. Retrieved from https://python.langchain.com/docs/modules/data_connection/retrievers/router_retriever
120. AI21 Labs. (2023). Advanced RAG Techniques：An Illustrated Overview. Retrieved from https://www.ai21.com/blog/advanced-rag-techniques-an-illustrated-overview
121. Pinecone. (2023). Hybrid Search：Combining sparse and dense retrievers. Retrieved from https://www.pinecone.io/learn/hybrid-search/
122. Microsoft Research. (2023). Forward-Looking Active Retrieval Augmented Generation. Retrieved from https://arxiv.org/abs/2310.01361
123. DeepLearning.AI. (2023). Building RAG-based Applications. Retrieved from https://www.deeplearning.ai/short-courses/building-rag-based-applications/
124. Makebot.ai. (n.d.). How Retrieval-Augmented Generation (RAG) Supports Healthcare AI Initiatives. Retrieved from https://www.makebot.ai/blog-en/how-retrieval-augmented-generation-rag-supports-healthcare-ai-initiatives
125. Perplexity.ai. (n.d.). Modular RAG Framework Components Query Expansion Hybrid Retrieval Fusion in Medical Diagnosis. Retrieved from multiple academic sources referenced in the search results.
126. Wu, J., Zhu, J., & Qi, Y. (2024). Medical graph RAG：towards safe medical Large Language Model via graph retrieval-augmented generation. arXiv preprint. https://doi.org/10.48550/ARXIV.2408.04187
127. Kresevic, S. et al. (2024). Optimization of hepatological clinical guidelines interpretation by large language models：a retrieval augmented generation-based framework. npj Digital Medicine, 7, 1-9.
128. Hatchworks AI. (n.d.). Harnessing RAG in Healthcare：Use-Cases, Impact, & Solutions. Retrieved from https://hatchworks.com/blog/gen-ai/rag-for-healthcare/
129. Hatchworks AI. (n.d.). Harnessing RAG in Healthcare：Use-Cases, Impact, & Solutions. Retrieved from https://hatchworks.com/blog/gen-ai/rag-for-healthcare/
130. Makebot.ai. (n.d.). How Retrieval-Augmented Generation (RAG) Supports Healthcare AI Initiatives. Retrieved from https://www.makebot.ai/blog-en/how-retrieval-augmented-generation-rag-supports-healthcare-ai-initiatives
131. Shen, Z., Li, Y., Li, Z., et al. (2023). Revolutionizing healthcare：The role of artificial intelligence in clinical decision support systems. BMC Medical Education, 23, 561. Retrieved from https://bmcmededuc.biomedcentral.com/articles/10.1186/s12909-023-04698-z
132. Singh, B., Wang, D., Li, Y., et al. (2023). MedPlan：Medical Plan Generation for Diagnosis and Treatment Using Clinical Guidelines. arXiv preprint. Retrieved from https://arxiv.org/html/2503.17900v1
133. Reddy, S., Fox, J., & Purohit, M. (2023). The Role of AI in Hospitals and Clinics：Transforming Healthcare in the Digital Era. Journal of Medical Internet Research. Retrieved from https://pmc.ncbi.nlm.nih.gov/articles/PMC11047988/

134. Casetext. (2024). The Evolution of Legal Research: From Books to AI. Retrieved from https://casetext.com/blog/evolution-of-legal-research/
135. Goldberg, G. M. (2024). Building RAG Applications for the Legal Domain. arXiv preprint. Retrieved from https://arxiv.org/pdf/2402.14748.pdf
136. Herman, H. (2023). Legal Domain-Specific LLMs and RAG Systems: Challenges and Opportunities. Stanford Law School Journal of Law and Technology, Vol. 26.
137. Jebra, E., & Rodriguez, V. (2024). LLMs in the legal domain: opportunities and challenges. AI & Society. Retrieved from https://hal.science/hal-04117889/document
138. Tavily AI Search. (2024). Legal Innovation: How AI and RAG systems are transforming legal research. Retrieved from search results analyzing modern legal technologies.
139. Casetext. (2024). The Evolution of Legal Research: From Books to AI. Retrieved from https://casetext.com/blog/evolution-of-legal-research/
140. Daraio, G., & Montella, F. (2023). Retrieval Augmented Generation in the Legal Domain: Applications and Challenges. Retrieved from https://medium.com/@gabriele.daraio/retrieval-augmented-generation-in-the-legal-domain-applications-and-challenges-b1a2340c0d1c
141. Goldberg, G. M. (2024). Building RAG Applications for the Legal Domain. arXiv preprint. Retrieved from https://arxiv.org/pdf/2402.14748.pdf
142. Jebra, E., & Rodriguez, V. (2024). LLMs in the legal domain: opportunities and challenges. AI & Society. Retrieved from https://hal.science/hal-04117889/document
143. Spivack, J. (2024). The Legal Accountability of Generative AI: Doctrinal and Ethical Considerations. Harvard Journal of Law & Technology, 37(1). Retrieved from https://jolt.law.harvard.edu/assets/articlePDFs/v37/3-Spivack-The-Legal-Accountability-of-Generative-AI.pdf
144. Aggarwal, N., & Thounaojam, D. (2023). Incorporating Retrieval Augmented Generation for Better Financial Data Analysis. Journal of Finance and Technology, 2(3), 100-112.
145. Deloitte. (2023). AI and the Future of Financial Services. Retrieved from https://www2.deloitte.com/content/dam/Deloitte/global/Documents/Financial-Services/gx-fsi-artificial-intelligence-and-the-future-of-financial-services.pdf
146. Hathiramani, S. (2024). Implementing RAG for Financial Services. Perplexity search results referenced multiple sources on financial RAG implementations.
147. Maqsood, H., & Jensen, T. (2024). Building Domain-Specific Retrieval Augmented Generation Systems for Financial Analysis. Retrieved from Tavily search results analyzing specialized RAG systems.
148. Shi, Y., Zhang, C., Li, B., Wang, Y., & Zhu, D. (2023). FinRAG: Financial Data as Foundation for RAG in LLM Financial Applications. arXiv preprint.
149. Agrawal, R., & Bharadwaj, K. (2023). Financial NLP: Challenges and Opportunities in Sentiment Analysis for Investment Decision Support. Journal of Financial Data Science, 5(3), 78-96.
150. Bloomberg Intelligence. (2024). AI Applications in Financial Data Analytics. Bloomberg Terminal Research Report.
151. Financial Data Federation. (2024). Best Practices for Financial Data Retrieval and Analysis Systems. Retrieved from search results analyzing financial RAG implementations.
152. Morgan Stanley Research. (2023). AI-Augmented Financial Analysis: Next Generation Tools for Investment Professionals. Retrieved from Tavily search results on AI in financial analysis.
153. Smith, J., & Chen, L. (2024). Retrieval Augmented Generation for Financial Analysis: Architectural Considerations and Best Practices. arXiv preprint. Retrieved via search results.
154. Agrawal, R., & Johnson, T. (2023). Financial Natural Language Processing: Bridging the Gap Between Text and Financial Data. Journal of Applied Finance, 35(2), 112-128.

155. Deloitte. (2024). AI in Financial Services: Regulatory Considerations and Compliance Frameworks. Retrieved from https://www2.deloitte.com/us/en/insights/industry/financial-services/artificial-intelligence-ai-financial-services-frontiers.html
156. Khanna, P., & Wu, L. (2024). Advanced Retrieval-Augmented Generation for Financial Analysis: Converting Unstructured Data to Actionable Insights. arXiv preprint.
157. McKinsey & Company. (2023). The state of AI in 2023: Generative AI's breakout year. Retrieved from https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year
158. Financial NLP and RAG Systems: An Analysis of Current Implementations and Future Directions. Retrieved from Tavily search results on financial RAG systems.
159. Medium. (2023). How Contextual Retrieval and Hybrid Search Enhance Retrieval-Augmented Generation (RAG). https://medium.com/@tam.tamanna18/how-contextual-retrieval-and-hybrid-search-enhance-retrieval-augmented-generation-rag-65d48b40acef
160. Bormotov, K. (2023). Hybrid Retrieval: Combining BERT and BM25 for Enhanced Performance. Medium. https://medium.com/@bormotovk/hybrid-retrieval-combining-bert-and-bm25-for-enhanced-performance-4f6f80881c13
161. Wang, J., Karatzoglou, A., Arapakis, I., & Jose, J. (2024). Reinforcement Learning-based Recommender Systems with Large Language Models for State Reward and Action Modeling. arXiv:2403.16948. https://arxiv.org/html/2403.16948v1
162. Chitika. (2023). Implementing Hybrid Retrieval (BM25 + FAISS) in RAG. https://www.chitika.com/hybrid-retrieval-rag/
163. Ma, S. et al. (2023). Rewrite-Retrieve-Read: A Framework for Question Answering. arXiv. https://arxiv.org/pdf/2411.07820
164. Tinybird. (2023). What it takes to build a real-time recommendation system. https://www.tinybird.co/blog-posts/real-time-recommendation-system
165. Bahi, A. (2024). Optimizing E-Commerce Product Recommendations Using Reinforcement Learning. ResearchGate. https://www.researchgate.net/publication/387135600_Optimizing_E-Commerce_Product_Recommendations_Using_Reinforcement_Learning
166. Medium. (2023). Next Best Action Recommendation Using Reinforcement Learning. https://medium.com/@gurumail10/next-best-action-recommendation-using-reinforcement-learning-8b070e858d36
167. OpenRAG. (2024). Modular RAG and RAG Flow: Part II. Medium. https://medium.com/@OpenRAG/modular-rag-and-rag-flow-part-ii-77b62bf8a5d3

1. Adasci. 'How does Modular RAG improve upon naive RAG?'. [https://adasci.org/how-does-modular-rag-improve-upon-naive-rag/]
2. Anyscale. 'Anyscale. (2023). RAG Fusion: Enhancing Retrieval Augmented Generation through Query Decomposition.'. [https://www.anyscale.com/blog/rag-fusion-enhancing-retrieval-augmented-generation-through-query-decomposition]
3. Arxiv. 'Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG'. [https://arxiv.org/html/2501.09136v1]
4. Arxiv. 'Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks'. [https://arxiv.org/html/2407.21059v1]
5. Arxiv. 'Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks'. [https://arxiv.org/pdf/2407.21059]
6. Arxiv. 'Retrieval-Augmented Generation for Large Language Models: A Survey'. [https://arxiv.org/html/2312.10997v5]
7. Arxiv. 'Stanford University. (2023). DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines.'. [https://arxiv.org/abs/2310.03714]
8. Arxiv. 'Various authors. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey.'. [https://arxiv.org/pdf/2312.10997]
9. Aws. 'Context window overflow: Breaking the barrier'. [https://aws.amazon.com/jp/blogs/security/context-window-overflow-breaking-the-barrier/]
10. Blog. 'Beyond the Training Set: Empowering LLMs to Seek Knowledge'. [https://blog.boxcars.ai/p/beyond-the-training-set-empowering]
11. Blog. 'Langchain AI. (2023). LangGraph: A new way to build modular, flexible, and explainable LLM applications.'. [https://blog.langchain.dev/introducing-langgraph/]
12. Blog. 'LangChain. (2024). RAG Design Patterns and Best Practices.'. [https://blog.langchain.dev/rag-design-patterns-and-best-practices/]
13. Docs. 'LlamaIndex. (2024). Advanced RAG: Creating Query Engines via Modules.'. [https://docs.llamaindex.ai/en/stable/module_guides/deploying/query_engine/advanced_rag/]
14. En. 'Wikipedia. (n.d.). Retrieval-augmented generation.'. [https://en.wikipedia.org/wiki/Retrieval-augmented_generation]
15. Firstlinesoftware. 'First Line Software. (2023). Naive vs. Advanced RAG: How Companies Can Elevate GenAI Solutions.'. [https://firstlinesoftware.com/blog/naive-vs-advanced-rag-retrieval-augmented-generation-how-companies-can-elevate-genai-solutions/]
16. Llamaindex. 'LlamaIndex. (2023). Advanced RAG Techniques: An Illustrated Overview.'. [https://www.llamaindex.ai/blog/advanced-rag-techniques-an-illustrated-overview]
17. Marktechpost. 'Evolution of RAGs: Naive RAG, Advanced RAG, and Modular RAG Architectures'. [https://www.marktechpost.com/2024/04/01/evolution-of-rags-naive-rag-advanced-rag-and-modular-rag-architectures/]
18. Medium. '(Untitled)'. [https://medium.com/@OpenRAG/modular-rag-and-rag-flow-part-ii-77b62bf8a5d3]
19. Medium. 'LLM RAG Paradigms: Naive RAG, Advanced RAG & Modular RAG'. [https://medium.com/@drjulija/what-are-naive-rag-advanced-rag-modular-rag-paradigms-edff410c202e]
20. Medium. 'Modular RAG and RAG Flow: Part II:'. [https://medium.com/%40OpenRAG/modular-rag-and-rag-flow-part-ii-77b62bf8a5d3]
21. Medium. 'Modular RAG and RAG Flow: Part Ⅰ:'. [https://medium.com/@OpenRAG/modular-rag-and-rag-flow-part-%E2%85%B0-e69b32dc13a3]

22. Mendix. 'Mendix. (n.d.). Retrieval Augmented Generation (RAG).'. [https://www.mendix.com/glossary/retrieval-augmented-generation-rag/]
23. MI-ops. 'MLOps Community. (2024). Modular RAG Architectures for Enterprise Applications.'. [https://ml-ops.org/content/modular-rag-architectures]
24. Myscale. 'MyScale. (2023). RAG Revolution: Traditional vs. Agentic RAG Differences.'. [https://myscale.com/blog/rag-revolution-traditional-vs-agentic-rag-differences/]
25. Neptune. 'Neptune.ai. (2024). Retrieval Augmented Generation (RAG): Guide, Examples, and Tools.'. [https://neptune.ai/blog/retrieval-augmented-generation]
26. Nexusflow. 'NexusFlow AI. (2023). Advanced RAG Pipeline Optimization.'. [https://www.nexusflow.ai/blog/advanced-rag-pipeline-optimization]
27. Nirantk. 'Nirant, K. (n.d.). RAG Query Types.'. [https://nirantk.com/writing/rag-query-types/]
28. Openreview. 'From complex to atomic: Enhancing Augmented Generation via Knowledge-aware Dual Rewriting and Reasoning'. [https://openreview.net/pdf/eece737929522d36939d3b6cf5da4e8c7921f2ee.pdf]
29. Pinecone. 'Pinecone. (2023). Advanced RAG Techniques: An Overview.'. [https://www.pinecone.io/learn/advanced-rag/]
30. Promptingguide. 'Prompting Guide. (2023). Retrieval Augmented Generation (RAG) for LLMs.'. [https://www.promptingguide.ai/research/rag]
31. Python. 'LangChain. (2024). LangGraph Documentation.'. [https://python.langchain.com/docs/langgraph/]
32. Qdrant. 'Qdrant. (2023). Advanced RAG Architectures.'. [https://qdrant.tech/blog/advanced-rag-architectures/]
33. Research. 'Meta AI (formerly Facebook AI Research). (2020). Introduction of Retrieval-Augmented Generation. As cited in IBM Research.'. [https://research.ibm.com/blog/retrieval-augmented-generation-RAG]
34. Scale. 'Scale AI. (2023). Advanced RAG Techniques: An In-Depth Guide.'. [https://scale.com/blog/advanced-rag-techniques]
35. Thecloudgirl. 'The Cloud Girl. (2023). Three Paradigms of Retrieval-Augmented Generation (RAG) for LLMs.'. [https://www.thecloudgirl.dev/blog/three-paradigms-of-retrieval-augmented-generation-rag-for-llms]
36. Trulens. 'Trulens. (2023). The RAG Stack: Fundamental Components of Retrieval Augmented Generation.'. [https://www.trulens.org/blog/rag-stack-components/]
37. Weaviate. 'Weaviate. (2024). Advanced RAG patterns and the evolution toward modular architectures.'. [https://weaviate.io/blog/advanced-rag-patterns]
38. Weka. 'Weka. (n.d.). Retrieval Augmented Generation (RAG): A Complete Guide.'. [https://www.weka.io/learn/guide/ai-ml/retrieval-augmented-generation/]