

Building Your Own RAG Application with XLRT.ai and MongoDB Atlas

Artificial Intelligence (AI) is entering a transformative phase, advancing beyond generating insights to executing complex, multistep workflows in real time. This evolution marks the rise of agentic AI—a groundbreaking innovation where AI systems act as autonomous agents, capable of decision-making, learning, and self-directed actions.

Agentic AI: From Thought to Action

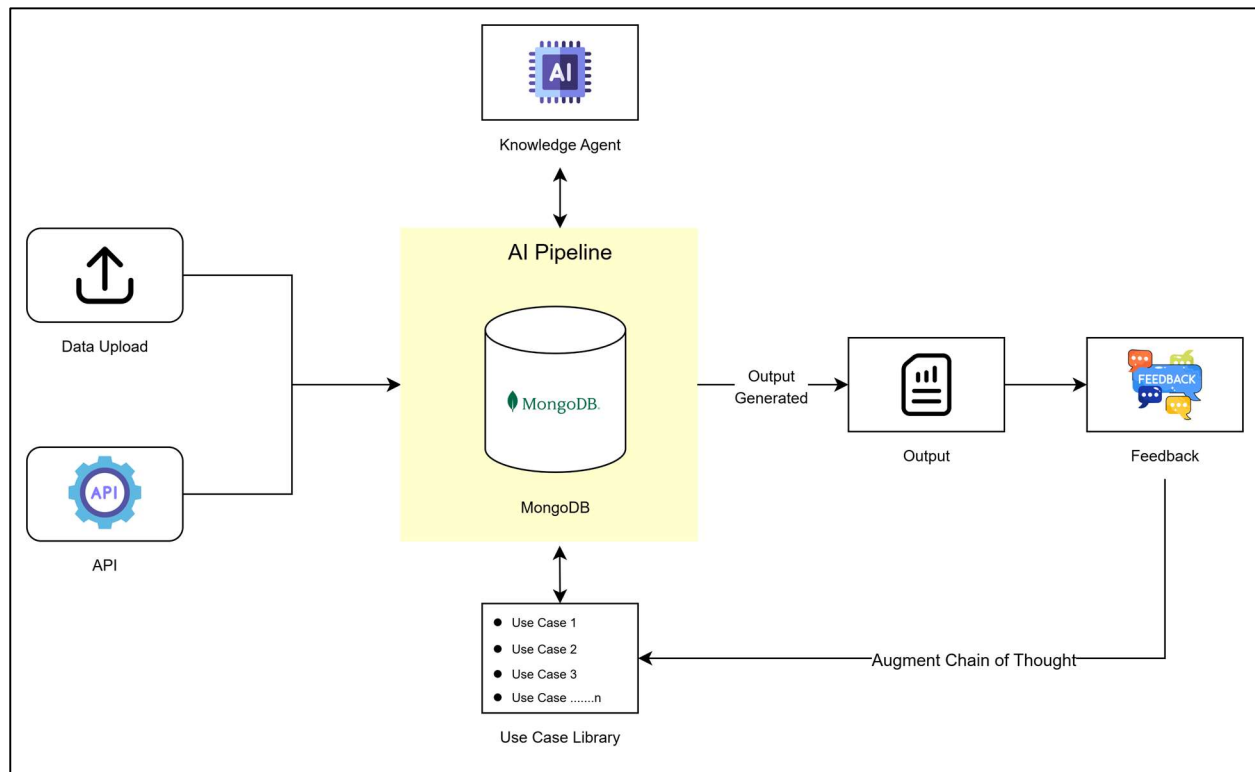
Unlike traditional or generative AI tools that primarily generate content or respond to queries, agentic AI systems leverage powerful foundation models to orchestrate tasks across digital ecosystems. These systems autonomously initiate and manage workflows, enhancing efficiency and innovation while redefining industry standards.

At the core of agentic AI is the intelligent agent, powered by large language models (LLMs) that operate dynamically:

1. **Perceive:** Gather and contextualize real-time data from documents, text inputs, or screenshots.
2. **Reason:** Analyze data, identify patterns, and determine optimal actions.
3. **Act:** Execute decisions, automate tasks, or trigger workflows.
4. **Learn:** Refine processes through feedback, improving adaptability and performance.

By integrating perception, reasoning, and action into seamless workflows, agentic AI revolutionizes how businesses operate in real-time.

How XLRT.ai Revolutionizes Financial Document Analysis



Credit Memo Creation Example

Credit memo creation is a critical yet time-consuming process in **loan underwriting**, requiring inputs from relationship managers, credit analysts, and other stakeholders. Traditional workflows often involve manual compilation, analysis, and iterative reviews—introducing delays and errors.

With **XLRT.ai**, this process is transformed:

1. **Role-Specific Agents:** XLRT.ai trains agents tailored for specific personas like **Credit Analysts** and **Relationship Managers**.
2. **Chain-of-Thought Reasoning:** Agents apply structured, step-by-step logic to ensure decisions are accurate and consistent.
3. **Domain-Specific AI:** XLRT grounds the large language models (LLMs) by selectively retrieving relevant financial knowledge and numerical data through Graph Retrieval-Augmented Generation (Graph RAG). This process uses a financial ontology—a graph structure representing key financial line items and their relationships—combined with client-specific data from uploaded documents such as annual reports and bank

statements. The result is highly contextualized insights tailored precisely to each client's unique financial conditions.

4. **Use Case Library:** Use cases like credit commentary would reside here.

Key Benefits of XLRT.ai for Financial Documents

1. **Enhanced Efficiency:** Automate workflows to reduce manual effort, enabling faster decision-making.
2. **Improved Accuracy:** Domain-tuned agents ensure insights are precise and consistent.
3. **Competitive Advantage:** Streamlined processes allow businesses to focus on strategic goals and client satisfaction.
4. **Cost Optimization:** Reduce dependency on large manual workforces while maintaining high-quality results.

Let's explore the various intelligence modules that make up this architecture in more detail.

Unveiling the Power of Knowledge Agents with Graph RAG

In the ever-evolving landscape of artificial intelligence, effective context retrieval and understanding relationships within complex data structures remain pivotal challenges. Knowledge Agents, when integrated with Graph Retrieval-Augmented Generation (Graph RAG), address these challenges by transforming how we interact with and interpret structured data.

Understanding Knowledge Agents

At their core, Knowledge Agents operate as intelligent systems designed to navigate, analyze, and infer insights from complex datasets. By leveraging graph-based representations, these agents can represent and reason through intricate relationships and dependencies.

In our implementation, **line items** such as "Revenue," "Expenses," and "Net Profit" are **represented as nodes**, while their **causal relationships**—like "Revenue" impacting "Net Profit" or "Expenses" reducing profitability—are **captured as edges**. For instance, if the "Revenue" node value increases, the connected "Net Profit" node may also be positively influenced, while an increase in the "Expenses" node could negatively affect "Net Profit." This interconnected framework enables the system to represent complex financial dependencies dynamically, mirroring real-world business scenarios.

Understanding the Financial Ontology and Graph Store

At the core of XLRT's approach is a sophisticated graph network known as the ontology. This network defines how various financial line items—represented as nodes—are interconnected through significant relationships, or edges. For example, the concept of “Revenue” is a node that influences “Net Profit” through defined edges.

Client-specific financial data, such as annual reports and bank statements, populate these nodes with the corresponding numerical values for each reporting period. Each node thus represents a line item, like “Revenue,” with values filled in from the client’s uploaded data, while edges articulate the relationships between these line items, such as how “Revenue” impacts “Net Profit.”

The graph store consolidates two essential components:

- The domain-specific ontology structures these relationships, providing a foundational understanding of the financial ecosystem.
- The actual yearly financial data sourced from the client’s documents, accurately mapped to their respective nodes within the ontology.

To deliver precise and context-aware insights, the Large Language Model (LLM) is grounded by selectively retrieving relevant knowledge and numerical information from this graph store using Graph Retrieval-Augmented Generation (Graph RAG). This targeted grounding ensures outputs that are both actionable and tailored to the client’s financial context.

Harnessing Graph RAG for Context Retrieval

Graph Retrieval-Augmented Generation (Graph RAG) combines the principles of graph theory with advanced AI retrieval and generation techniques. This integration allows the system to:

- 1. Retrieve Context for Any Line Item:**
 - a. By querying the graph, the system extracts relevant nodes and edges, providing a contextual snapshot of the data surrounding a particular line item.
 - b. This enables a deeper understanding of how individual components interact within the larger dataset.
- 2. Analyze Causality:**
 - a. Graph RAG facilitates tracing cause-and-effect relationships within the data. For example, it can identify how a change in one line item might influence others, uncovering causal chains and dependencies.
- 3. Identify Illogical Correlations:**

- a. By examining relationships between nodes, the system can detect inconsistencies or correlations that defy logical reasoning. This capability ensures data integrity and highlights areas requiring further investigation.

Key Advantages of Graph RAG-Powered Knowledge Agents

1. **Enhanced Contextual Awareness:**

- a. By representing data as a graph, the system offers unparalleled insight into the structure and interconnectivity of datasets.

2. **Improved Decision-Making:**

- a. With the ability to analyze causality and detect illogical correlations, decision-makers can act with greater confidence and precision.

3. **Scalable and Adaptable:**

- a. Graph-based representations are inherently scalable, making them suitable for datasets of varying sizes and complexities.

Why use LangChain's MongoDBGraphStore?

MongoDBGraphStore is a core component within the LangChain MongoDB integration that enables GraphRAG by storing financial entities (nodes) and their relationships (edges) as records in a MongoDB collection. Each record represents an entity with fields referencing related entities, accurately modeling the graph structure of financial data. The graph traversal and querying are executed using MongoDB's powerful **\$graphLookup** aggregation stage, allowing efficient retrieval of relevant interconnected financial knowledge.

- The **entity extraction model** is an LLM-based model provided at MongoDBGraphStore initialization.
- It is used to **convert client uploaded financial statements into graph entities and relationships** by extracting named entities and possible connections between them.
- The model typically uses custom prompts and instructions to guide the extraction, customized via `entity_prompt` and related parameters.
- During graph creation (`add_documents()`), **this model is called to automatically extract and upsert entities and relationships into the MongoDB collection.**
- It enables creating dynamic, interconnected knowledge graphs from unstructured data **without manual graph schema or edge management.**

Steps to Implement Graph RAG with MongoDB and LangChain

1. Prepare your Environment

```
pip install --quiet --upgrade pymongo langchain_mongodb  
langchain_ollama
```

2. Define variables

```
MONGODB_URI = "<connection-string>"  
DB_NAME = "financial_kg_db" # MongoDB database to store the knowledge  
graph  
COLLECTION = "FINANCIALS" # MongoDB collection to store the knowledge  
graph
```

<connection-string> would be the **connection string from your Atlas cluster**.

A **Atlas cluster** running MongoDB version 6.0.11, 7.0.2, or later. It should use the following format:

```
mongodb+srv://<db_username>:<db_password>@<clusterName>.<hostname>.  
mongodb.net
```

3. Instantiate the LangChain MongoDB Graph Store

```
from langchain_mongodb.graphrag.graph import MongoDBGraphStore  
  
graph_store = MongoDBGraphStore.from_connection_string(  
    connection_string=MONGODB_URI,  
    database_name=DB_NAME,  
    collection_name=COLLECTION_NAME,  
    entity_extraction_model=chat_model # LLM - model of your choice  
)
```

4. Add data to the Graph Store

```
# Add documents, to extract entities and relationships from, to the graph store

graph_store.add_documents(documents)
```

5. Query the Graph Store

```
# Query the knowledge graph to get related entities and context for prompts
Query = "What are the key financial metrics for Q4? "

Context = graph_store.query(query, max_hops=3)
```

6. Combine Context for Prompt Augmentation

Once the relevant data is retrieved, we combine the retrieved context from Graph RAG to augment the prompt for the LLM using LangChain and your model of choice served through Ollama.

Python Example for Context Construction:

```
from langchain_ollama import ChatOllama
from langchain.prompts import PromptTemplate

# Set up Ollama as the LLM with your model of choice
llm = ChatOllama(model="<model of your choice>")

# Define a prompt template
template = """
You are an AI financial analyst.
Analyze the following data and provide insights:

{context}

User Query: {query}
"""

prompt = PromptTemplate(input_variables= ["context", "query"],
template=template)
chain = prompt | llm

response = chain.invoke({"context": context, "query": Query})
```

```
print("Generated Insights:\n", response)
```

Augmenting Chain of Thought (CoT) Prompts with Scoring for Improved Accuracy

In the world of large language models (LLMs), ensuring the accuracy and reliability of generated responses is paramount, especially in complex tasks requiring detailed reasoning. While Chain of Thought (CoT) prompting encourages step-by-step reasoning, there are instances when the model's output doesn't fully align with the correct context or logic. To address this, we can augment CoT by incorporating a dynamic feedback loop based on scoring mechanisms that assess the quality of the LLM's response and adjust the prompt accordingly. This iterative process allows the system to refine its reasoning and improve accuracy through multiple cycles.

The Challenge of Ensuring Correctness

Despite the advancements in LLMs, the models may still produce hallucinations or incorrect outputs. This occurs when the model misunderstands context, overlooks critical details, or makes erroneous logical jumps. While CoT helps structure reasoning, the model may still require correction during the output generation process. To mitigate this, it's crucial to assess the quality of the response at each stage and provide feedback that guides the model toward a more accurate solution.

Augmenting CoT with Scoring-Based Feedback

The process of augmenting CoT with scoring involves evaluating the model's response against predefined correctness metrics and adjusting the reasoning steps based on the results. This approach ensures that the model's output is continuously refined, leading to more reliable and accurate conclusions. The key steps involved in this process include:

1. **Initial CoT Prompting:** The model is first prompted to reason through a problem using a structured CoT approach. For example:
 - a. "First, identify the key facts in the prompt."
 - b. "Next, examine the relationships between these facts."
 - c. "Finally, generate a conclusion based on the reasoning so far."
2. **Response Generation:** The LLM generates a response based on the CoT steps. The response may be factually correct or may contain errors, which are identified by the scoring mechanism.

3. **Scoring the Response:** A scoring system is applied to assess the accuracy of the LLM's response. The scoring can be based on various criteria such as:
 - a. **Contextual Consistency:** How well does the response align with the context provided in the prompt and reasoning steps?
 - b. **Factual Accuracy:** Does the response adhere to known facts, rules, or data?
 - c. **Logical Soundness:** Are the intermediate reasoning steps logically connected, and does the conclusion follow from the reasoning?
4. **Prompt Adjustment Based on Score:** If the score indicates that the response is inaccurate or flawed, the prompt is dynamically adjusted. The system updates the CoT prompt, highlighting areas where the reasoning might be wrong or missing. This updated prompt is then fed back to the LLM for a refined response. For example:
 - a. **Original Prompt:** "What is the revenue growth in Q4 2024 for Company X?"
 - b. **First Response:** "Company X experienced a 10% growth."
 - c. **Score:** 60% (indicating a potential error, as the report shows 8% growth).
 - d. **Adjusted Prompt:** "Given the Q4 2024 report for Company X, analyze the revenue growth and verify the percentage change compared to the previous quarter."
5. **Re-iteration of Reasoning:** The LLM is then prompted again with the updated CoT steps to reprocess the information, hopefully generating a more accurate response.
6. **Iterative Refinement:** This process is repeated until the scoring mechanism evaluates the response as satisfactory or sufficiently accurate. The final output is the result of multiple cycles of reasoning and validation.

Benefits of Scoring-Driven CoT Augmentation

1. **Continuous Feedback Loop:** By scoring responses and adjusting prompts dynamically, the model continuously refines its output. This iterative process ensures greater accuracy over time.
2. **Contextual Error Correction:** The scoring mechanism allows for targeted correction, pinpointing areas where the model has misunderstood or misrepresented the context, and directly addressing these flaws.
3. **Reduction of Hallucinations:** With scoring-based feedback, the model is less likely to hallucinate or generate factually incorrect responses. Errors are identified early in the process, allowing the system to self-correct.
4. **Enhanced Decision Support:** In applications requiring high reliability, such as legal, financial, or medical contexts, scoring-driven CoT augmentation ensures that the model provides responses that can be trusted and are backed by rigorous reasoning.

Example of Scoring and Augmentation in Action

Let's consider a financial analysis task where the model is asked to analyze the performance of a company over multiple quarters.

Prompt: "Analyze the performance of Company X from Q1 to Q4 2024 and identify trends in revenue, expenses, and profit margins."

1. **Initial CoT Steps:**
 - a. "Identify the revenue, expenses, and profit margins for each quarter."
 - b. "Compare the trends across these metrics from Q1 to Q4."
 - c. "Summarize the key findings and any significant changes."
 2. **First Response:** "Company X saw a steady increase in revenue throughout 2024, with expenses slightly fluctuating. The profit margin remained stable."
 3. **Scoring:**
 - a. **Contextual Consistency:** 85% (Correct metrics mentioned but lack detailed comparison between quarters).
 - b. **Factual Accuracy:** 70% (The report shows a significant decrease in profit margin in Q3, not captured).
 - c. **Logical Soundness:** 80% (The reasoning is sound, but it misses a critical comparison of Q3).
 4. **Adjusted Prompt:**
 - a. "Reanalyze the revenue, expenses, and profit margins for each quarter, paying special attention to the significant drop in profit margin in Q3. Compare these trends across the four quarters."
 5. **Re-iteration:** The LLM processes the adjusted prompt and provides a more accurate response reflecting the changes observed in Q3.
-

Augmenting Chain of Thought Prompts with User Feedback for Personalized Financial Commentary

In a dynamic financial environment, users, each with different roles, often seek clarification on data in ways that align with their specific needs and objectives. Financial commentary, initially generated by an LLM, may provide valuable insights, but its relevance can be further enhanced by **user feedback**. Instead of correcting errors in the initial analysis, user feedback is used to tailor

the Chain of Thought (CoT) prompts to each user's specific context, ensuring that the generated commentary serves their distinct purpose.

This approach not only improves the adaptability of the system but also empowers users to explore financial data from various perspectives, creating a more personalized and user-centric experience.

Step-by-Step Process of Augmenting CoT Prompts with User Feedback

1. **Initial Generation of Financial Commentary:** Initially, the system generates a financial commentary using LLMs (e.g., Llama3, served via Ollama) based on the given financial data. This commentary offers general insights into the data, covering aspects such as trends, anomalies, and relationships between financial variables.
2. **User Feedback Based on Role:** After viewing the initial commentary, users from different roles (e.g., financial analysts, executives, auditors) may seek further clarification based on their specific needs. For instance:
 - a. **Financial Analyst:** "Can you elaborate on the correlation between Q3 expenses and revenue performance?"
 - b. **Executive:** "What impact does the revenue dip in Q3 have on our year-end projections?"
 - c. **Auditor:** "Can you provide a breakdown of the liabilities increase in comparison to equity and its impact on solvency?"

This feedback is not aimed at fixing the original commentary but rather at deepening the analysis to meet the specific user's requirements.

3. **Scoring and Evaluation:** While scoring in this context may not necessarily be tied to error correction, the feedback can still be evaluated for context. Based on the type of inquiry and the role of the user, the system identifies which elements of the financial data need to be explored in more depth, enabling personalized adjustments to the prompt.
4. **Dynamic Augmentation of CoT Prompts:** With the user's role and feedback in mind, the system augments the **CoT prompt** to guide the LLM toward providing a more focused response. For example:
 - a. For a **financial analyst**, the prompt may be adjusted to explore deeper correlations between different financial metrics.
 - b. For an **executive**, the prompt may be enhanced to link data trends to high-level business impacts.
 - c. For an **auditor**, the prompt could focus on compliance, risk, and financial health indicators.

5. **Re-running the LLM with Augmented Prompts:** After modifying the CoT prompt, the system feeds the augmented prompt back into the LLM, generating a more targeted and specific response based on the user's role and the nature of their query.
6. **Output and Display:** The LLM's refined response is then displayed to the user, offering a personalized, context-rich commentary. This allows each user to gain deeper insights relevant to their role and objectives.
7. **Iterative Refinement:** As users continue to interact with the system, further feedback can be incorporated, leading to more refined CoT prompts. This iterative process ensures that the system adapts and continues to serve the evolving needs of users in different roles.

Key Benefits of Feedback-Driven CoT Prompt Augmentation

1. **Personalized Experience:** By adjusting the CoT prompts based on the user's feedback, the system tailors its responses to meet the specific needs and objectives of different roles, improving the user experience.
2. **Targeted Analysis:** Each user receives an analysis that is relevant to their function, whether it's detailed trend analysis for analysts, high-level projections for executives, or compliance checks for auditors.
3. **Contextual Relevance:** The system adapts dynamically, ensuring that the generated commentary remains contextually relevant, no matter the user's query or purpose.
4. **Improved Decision-Making:** By receiving insights customized to their unique perspectives, users are empowered to make more informed and precise decisions.

Conclusion

By leveraging **XLRT.ai** and **MongoDB Atlas**, financial organizations can build robust RAG systems to handle critical document workflows. From **invoice analysis** to **credit memo creation**, this approach transforms unstructured data into actionable insights—enhancing efficiency, accuracy, and decision-making capabilities.

Ready to revolutionize your financial processes? Visit [XLRT.ai](https://xlrt.ai) to learn more.