Reference Architecture Brief: Retrieval-Augmented Generation

8 October 2024 - ID G00801718 - 29 min read

By: Sumit Agarwal

Initiatives:Analytics and Artificial Intelligence for Technical Professionals; Architect, Implement and Scale Data and Analytics Solutions

To address LLM challenges like inaccurate and irrelevant responses, data and analytics architects should use the RAG architecture. This research serves as a blueprint for scalable generative AI solution development — integrating LLMs with enterprise data to enhance solution accuracy and relevance.

Architecture Brief

Respondents to the 2024 Gartner Growth Agenda Survey ranked generative AI (GenAI) as the most critical technology for driving high enterprise revenue growth over the next three years. [1] This finding, combined with the 2023 Gartner Voice of the Client Content Survey revealing that 78% of clients had used GenAI in some capacity for work, underscores the significant interest and impact potential of this technology. [2]

The current wave of innovation began with the release of GPT-3.5 and ChatGPT in late 2022, creating a substantial opportunity to leverage organizational unstructured data. While the technology itself was not new, the enhanced conversational capabilities and the ability to provide instructions within prompts proved transformational. Nevertheless, it was difficult to seamlessly integrate GenAI solutions into enterprise workflows. That difficulty was caused by hallucinations in responses, the high cost of developing and maintaining organizational large language models (LLMs), and inaccuracies due to limitations in the LLM knowledge base. Retrieval-augmented generation (RAG) addresses these issues by using the context provided in prompts to reduce inaccuracies, while employing LLMs for synthesis and text generation.

A prompt serves as the interface to the LLM. While a prompt can be as simple as a user's question, enterprise use cases are rarely that simple. To ensure a relevant response to the initial query, users might need to provide the LLM with additional context or instructions (or system prompts). Integrating the user's query with these instructions and retrieving supporting content from a corpus of unstructured data in a scalable and consistent manner demands a complex solution with multiple components and capabilities.

This research presents a reference architecture that integrates essential capabilities necessary for the successful development and deployment of a generative AI solution using the RAG technique. The solution integrates LLMs with enterprise data. Organizations are advised to adopt this reference architecture as a guiding template or blueprint to define their vision and as a strategic journey map, progressively incorporating capabilities as they scale their solution implementations.

Architecture Use Cases

The RAG reference architecture has applications in the following scenarios: [3]

- **Up-to-date and relevant information processing** — LLMs are pretrained on vast amounts of data, but this data is limited to a specific point in time. Consequently, the models may provide inaccurate answers if their knowledge base does not include information relevant to the user's question. RAG offers a solution by augmenting additional, up-to-date information within the prompt related to the user's query.

- **Content access control** — LLMs lack the ability to enforce data access controls in their responses. RAG offers a solution by incorporating data based on user access within the context of the prompt. The controls would need to be applied as part of the retrieval process.

- **Domain-specific responses** — Most LLMs are general-purpose and lack a focus on specific industries or domains. Additionally, organizations possess proprietary data that typically isn't included in the model's training data. RAG offers a solution by contextualizing responses using the organization's business domain data and may work as an alternative to LLM fine-tuning.

- **Mitigating hallucinations** — LLMs are predictive models and can produce inaccurate responses, or hallucinations. Guiding responses with specific context provided in the prompt helps reduce inaccuracies. Additionally, including content sources as part of the prompt and the resulting response provides users with a reference to the source of the information.

Architecture Diagram

The reference architecture in Figure 1 groups the core capabilities into three functional categories:

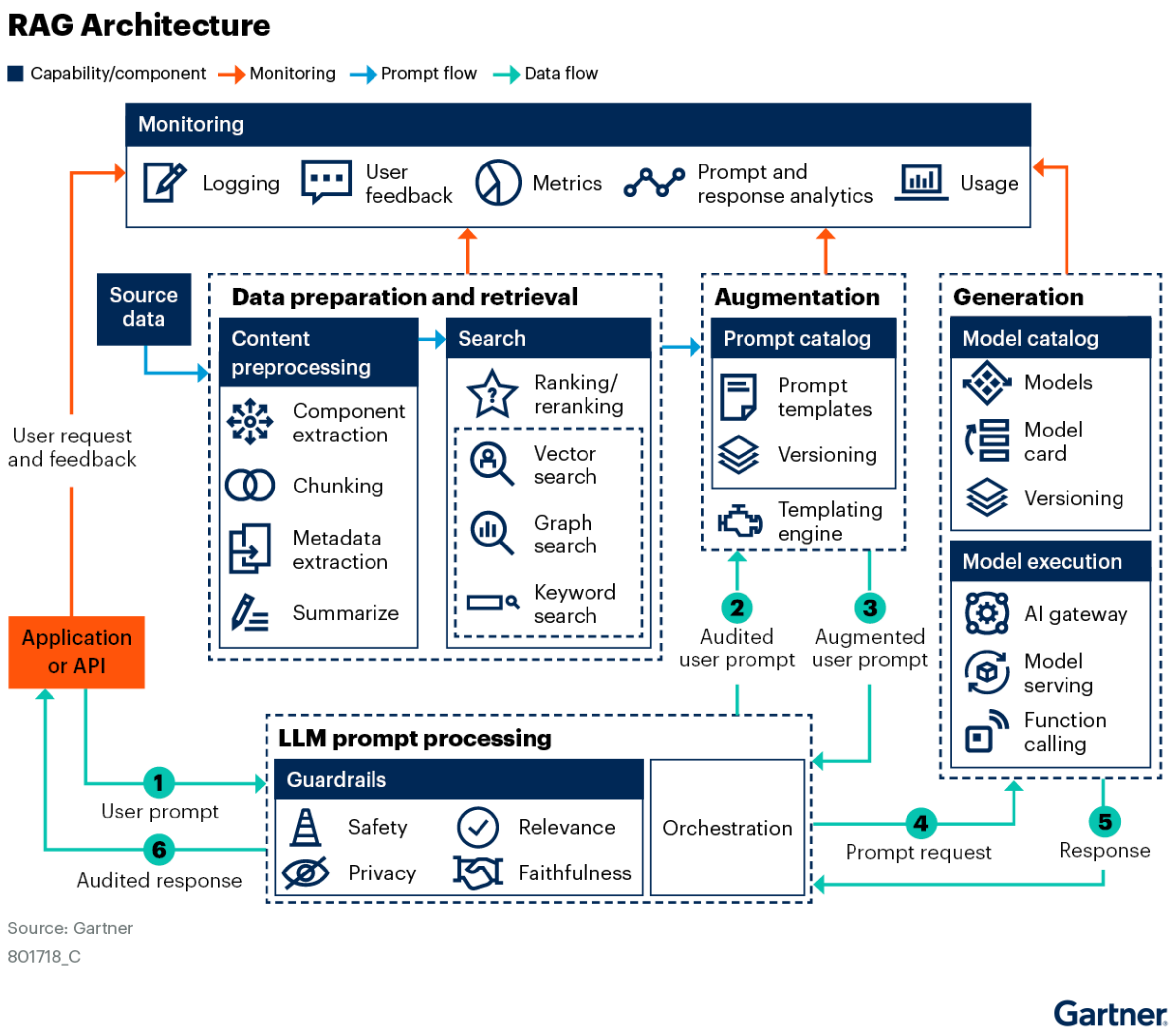
1. **Data preprocessing and retrieval:** This involves retrieving relevant data related to the context of the user's question. The data source may include internal or external documents.

2. **Augmentation:** This step enhances the user's question with the selected context and provides instructions for the LLM.

3. **Generation:** In this phase, the LLM generates a response based on the augmented prompt.

▶ Download an Editable Version of This Figure

Figure 1: RAG Reference Architecture



Architecture Capabilities & Components

While the three functional categories mentioned above provide the foundational capabilities, an enterprise RAG implementation also requires prompt processing and engineering. This includes application of guardrails and orchestration of various prompt interactions and monitoring to ensure that the user's prompt and the LLM's response comply with privacy, safety and other regulatory requirements. The solution must also be consistent and flexible to accommodate a broad range of user requirements, source content formats and structures.

The resulting reference architecture for enterprise RAG implementations include the following core components:

- **Content preprocessing:** The source documents need various preprocessing steps to enable effective search of the relevant content at the required granularity.
- **Retrieval:** The search functionality extracts the content relevant to the user's query. Multiple search techniques may need to be implemented along with a ranking function to combine the results from the various search engines.
- **Prompt template catalog and templating engine:** Prompt templates provide prompt structures or schema based on various use cases. A catalog will help manage the different templates that would map to different models. The associated templating engine will apply the selected template to create the prompt.
- **Model catalog:** A centralized repository for selecting and managing LLMs.
- **Model execution:** Centralizes the integration and access to LLMs, including hosting the self-deployed LLMs.
- **Guardrails:** Guardrails are mechanisms designed to ensure the safe and ethical use of LLMs by filtering inappropriate content, enforcing privacy policies and maintaining response accuracy.
- **Monitoring:** RAG solutions require continuous oversight to identify issues with accuracy, content gap, usage and other measures to ensure the effective operation and trustworthiness of the AI system.
- **Orchestration:** There are several components that need to be connected to enhance the user query into an augmented, audited prompt and the LLM response into a validated response.

Content Preprocessing

A successful implementation of a RAG solution relies on a pertinent collection of documents to enhance contextual understanding and generate responses that are both relevant and accurate. These documents can encompass various file formats, structures and data types, incorporating embedded elements such as images, tables, graphs, charts, footnotes and citations. Additionally, the content within these documents may span multiple languages and even include handwritten notes. To ensure optimal usability within the RAG solution, these documents necessitate preprocessing steps for effective consumption.

Preprocessing includes various steps and related capabilities:

- **Component extraction:** While LLMs excel at synthesizing text, they require the content to be in plain text. Various document components may need different actions. For example, a table might need conversion to a JSON or XML structure. Headers and footers may need to be excluded from the RAG solution. Privacy-related data elements might need masking or redacting. Extracting various components provides the flexibility to apply different preprocessing actions at a more granular level.
- **Metadata extraction:** The metadata associated with the source content is essential for applying consumption access controls and tracking lineage. Metadata may include the source location, owner, classification, purpose, change action and archival date, among others. Metadata is not limited to the document as a whole; the components extracted in the previous step would have additional metadata. For example, an image might include metadata, such as title, document page, source, reference link and description.
- **Chunking:** Chunking breaks up the document into smaller, manageable pieces (or chunks). There are different chunking approaches, including fixed-size chunking, semantic chunking and recursive chunking. Smaller document chunks improve semantic search results when trying to find relevant content for the user's query. The selection of chunking approaches and sizes need to align with the use-case requirements.
- **Summarization:** Storing summaries of source content, or specific components or sections, and using them for search enables improved accuracy and optimizes the retrieval process. This is particularly useful for large documents.

Example Technologies

Component extraction:

- Camelot (open source)
- LlamaIndex (open source)
- Shelf.io
- Unstructured.io

Metadata extraction:

- LangChain (open source)
- LlamaIndex (open source)
- Pryon
- Shelf.io

Chunking:

- LangChain (open source)
- LlamaIndex (open source)

Summarization:

- LLMs

Key Characteristics

Content preprocessing has the following characteristics:

- **Use-case-agnostic:** Content extraction and metadata extraction, for a scalable implementation, should be independent of the specific consumption or use case. The use-case implementation may involve using part or all of the subcomponents. For example, a document decomposed into subcomponents may be stored as a graph. RAG may be one of the use cases, but the graph should be consumable for additional analysis as well.
- **Integration of privacy and access controls:** The content may include personally identifiable information (PII) data along with confidential content. The content integration into a RAG solution must inherit any access controls and incorporate required data privacy controls, such as masking or anonymization.
- **Optimizations:** The content preprocessing may include generic optimization-related data transformations (e.g., summarization or chunking).

Related Architecture Research

- Use These 8 Practices to Improve RAG Systems

Retrieval

Search functions are used to retrieve relevant information from a predefined corpus or database based on the user's query, which is then used to augment the prompt with appropriate context. The effectiveness of search relies on the clarity of the user's query and how well the content is organized and preprocessed (refer to the previous section: "Content Preprocessing"). Common search techniques and algorithms include:

- **Vector search:** Vector search, also commonly known as semantic search, matches content based on semantic similarity rather than just keyword matches. For example, a vector search for "football" might include results for "soccer," while a keyword search would likely exclude it. This example illustrates how understanding the meaning and context of a search query can yield more relevant results. Vector search uses embeddings, which are a mathematical vector representation of data in a multidimensional space. This approach leverages vector representations to find relevant results based on the similarity between vectors.
- **Graph search:** Graph search leverages a knowledge graph built from source documents related to the use case. This approach identifies additional context by examining the relationships or dependencies between entities. Graph search also performs better on large documents by decomposing them into more granular components and organizing these components as entities and relationships. While graph-based search is an established technique, organizing documents into a knowledge graph represents an emerging approach.
- **Keyword search:** It retrieves documents or records containing specific words or phrases provided by the user. It matches the exact keywords in the query with the text in the corpus. This method is straightforward and efficient for finding information that explicitly includes the search terms.

The above search techniques may be combined to increase the effectiveness of the search. For example, graph and vector search may be combined or a keyword search may be applied to vector search output. Emerging techniques, such as hypothetical document embeddings (HyDE), or query rewrite may be used as additional approaches to potentially improve the search outcomes.

A combination of search techniques require a component for:

- **Ranking or reranking:** The above search techniques may return multiple results, and results from various content sources might need to be merged. This process requires an initial ranking methodology. To further enhance response accuracy, a reranking model can provide a similarity score between the selected context and the user query, enabling the sorting of search results based on this score. Think of it this way: the initial search narrows the scope of relevant content from a broad set, while reranking refines and orders these search results to improve response accuracy even further.

Example Technologies

Vector search:

- Amazon Web Services (AWS)
- DataStax
- Elasticsearch
- Google
- Microsoft
- MongoDB
- Pinecone Systems
- Qdrant
- Redis
- Weaviate

Graph search:

- Graphlit
- Microsoft GraphRAG
- Ontotext

Keyword search:

- BM25 function
- Database query

Ranking/Reranking:

- Cohere Rerank
- Google Vertex AI Agent Builder (ranking API)
- NVIDIA NeMo Text Retriever NIM (Text Retriever NIM)

Additional search engines using hybrid search techniques:

- Amazon Kendra
- Azure AI Search
- Google Vertex AI Search

Key Characteristics

Retrieval, including search, has the following key characteristics:

- **Source data organization:** The effectiveness of various search techniques depends on how the source data is organized. For instance, breaking documents into smaller chunks yields better results with a vector search. However, chunks that are too small can result in less relevant results.
- **Access control integration:** Integration of user access with the search tools may enable a better managed access control process for the content consumption.
- **A combination of enterprise and localized capabilities:** The data retrieval for RAG solutions may be based on an enterprise graph or a use-case-specific vector database. The solution design should accordingly assess the performance and capacity requirements along with integration requirements impacting change data management and update frequency.
- **Scale and performance:** The search indexes grow in size and compute as the volume of data increases. In addition, the search engine should be able to support an increase in search requests. Based on the above design decisions, the search index update frequency, latency and sizing would need to be defined.

Related Architecture Research

- Vector Databases Have Value Beyond Large Language Model Integration
- How Large Language Models and Knowledge Graphs Can Transform Enterprise Search
- How to Evaluate the Applicability of Knowledge Graphs for Your Use Cases

Prompt Template Catalog

A prompt serves as an interface to the LLM and includes three core segments: the instructions or system prompt, the context related to the user's query and the user's query itself. Different use cases will require different instructions, and various LLMs may have slight variations in this structure. For example, the instructions for text summarization would differ from those for a question-and-answer use case. Additionally, the prompt template for Anthropic's Claude model may have a different structure compared to OpenAI's GPT model. A prompt template provides a schema to populate at runtime and includes predefined instructions as part of the prompt engineering work. As the number of RAG implementations increases, a collection of prompt templates or a catalog becomes necessary for better management. A prompt template catalog includes the following components:

- **Prompt templates:** Each template includes the LLM name, provider, version, instructions with placeholders for fields to be populated at runtime and placeholders for context and the user's query. It may also include additional metadata, such as the template owner, use-case summary (or a link to use-case documentation) and applicable dates.
- **Versioning:** Prompt templates need versioning to align with business-related updates and LLM updates. This enables auditing of changes to the templates and allows incremental updates as new LLMs and new versions become available.

Example Technologies

- Amazon Bedrock Prompt Management
- Azure AI Studio
- Custom solution based on JSON or YAML format
- Galileo
- Humanloop
- PromptLayer

Key Characteristics

A prompt template catalog has the following characteristics:

- **Scalability and flexibility:** Provides a centralized repository to manage prompt templates across all enterprise RAG implementations. This capability allows for the efficient management and retrieval of a large number of templates as the number of RAG implementations increases, supporting rapid development and deployment.
- **Collaboration:** Supports collaboration among team members, enabling shared access and collective management of templates. This enhances reusability and consistency of prompts.
- **Mapping and lineage:** Provides a mapping between use cases or applications, prompt templates and LLMs. This mapping simplifies the introduction of new templates and LLMs and offers a lineage and dependency map as LLMs are updated or replaced by different providers.
- **Documentation:** Includes documentation for each template, explaining its purpose, structure and usage.
- **Guidelines and best practices:** Serves as a central repository for guidelines and best practices for creating and managing prompt templates.

Related Architecture Research

- What Technical Professionals Need to Know About Large Language Models

Prompt Templating Engine

The prompt templates containing the model instructions and the prompt structure needs to be mapped to the user's query and the related context retrieved from the various data sources, to instantiate the augmented prompt. A template engine enables the mapping of the user or application content with the variables or the placeholders in the predefined template.

Example Technologies

- Jinja
- Mustache

Key Characteristics

The prompt templating engine should have the following characteristics:

- **Configuration-driven and flexible:** The templating engine should be configurable to define custom variables for the templates.
- **Scale:** As the LLM token lengths limits increase, the prompt may be able to include more context. This will increase the prompt size. The template engine should be able to scale to the required prompt size.
- **No-code or low-code:** The templating engine shouldn't need additional code for mapping the data to the template variables. However, it may provide additional extensions for invocation of functions or APIs as an option.

Related Architecture Research

None

Model Catalog

Enterprise RAG implementations typically start with a single LLM. However, optimizing for specific use cases, cost, domain specificity, data sensitivity, regional constraints and other factors often necessitates the use of multiple LLMs. These can include a mix of commercial and open-source models, general-purpose and specialized models, as well as large and small models. A model catalog serves as a registry, detailing the specifications, capabilities and use cases of various models.

It should include the following components:

- **Models:** These can be LLMs physically hosted within the enterprise environment or APIs for accessing external LLMs.
- **Model cards:** Documentation for each LLM, including information about the model developer, owner, license and cost, architecture, data sources, benchmarks and performance metrics, use cases, API signature, fine-tuning options and other relevant details.
- **Model versions:** Management of different model versions as LLMs are updated with new data.

Example Technologies

Model catalogs:

- Amazon Bedrock
- Azure AI Studio model catalog
- Google Vertex AI Model Garden
- Hugging Face Model Hub
- Replicate

Key Characteristics

The model catalog should have the following characteristics:

- **Platform-agnostic:** The model catalog should not be limited to LLMs. It should serve as a registry for all AI models, regardless of the development platform, to ensure flexibility and avoid dependency on a single platform for model development.
- **Curated models:** The LLMs included in the catalog should be curated by the organization’s core generative AI team.
- **Ownership and usage approval framework:** The catalog should include a framework for ownership and usage approval to better manage costs and data privacy. However, research-focused organizations using various general-purpose models might require a more open access process for the models available in the catalog.

- **Versioning linked to prompt templates:** The versioning of LLMs should be linked to the versions of prompt templates. Updates to LLMs have previously necessitated changes in prompt templates.

Related Architecture Research

- How to Explain AI and Machine Learning Models to Your Stakeholders

Model Execution

LLMs can be accessed through an API from a third-party LLM provider, as a cloud platform as a service (PaaS), or implemented as self-hosted solutions due to data privacy or sensitivity constraints. This involves three core components:

- **AI gateway:** This intermediary layer provides controlled access to LLMs (and other AI models) for the applications that consume them. AI gateways manage runtime traffic between the applications or processes and AI API providers, implement and manage prompt-based policy controls, track AI service usage and costs, route requests across multiple LLMs, and manage access to AI subscriptions, including protecting API keys issued by AI providers. This reduces the risk of unexpected costs from AI providers and prevents private data in API traffic from being compromised or misused. An AI gateway also allows an organization to decouple client applications from specific API providers and manage the usage of multiple AI models from different providers.
- **Model serving:** Serving LLMs presents unique challenges related to their size, requiring more storage, memory and often graphics processing unit (GPU) compute. For example, the open-source Llama 3.1-8B model, the smallest in the series, requires approximately 16 GB of storage and 20 GB of GPU memory for inference at 16-bit precision (FP16). Additional factors, such as loading times, throughput optimization, memory and cache management, compute management and parallelism, also need to be considered as part of the model serving solution.
- **Function calling:** This feature enables LLMs to generate code snippets that can be used to external functions or APIs based on prompts or natural language instructions. This allows interactions with the LLM to perform generated actions, such as retrieving additional data, sending emails or invoking internal web services. This helps engineers use LLMs for solutions that require integration with other systems or software algorithms. However, not all LLMs or LLM APIs have function-calling capabilities, and it is not a required feature for a RAG implementation.

Example Technologies

AI gateway:

- Aguru
- Cloudflare
- IBM
- Kong
- Lunar.dev
- Martian
- MLflow Deployments Server (open source)
- Portkey
- Radiant
- Solo.io

Model serving:

- Amazon Bedrock
- Anthropic
- Azure Machine Learning (endpoints)
- Databricks Mosaic AI Model Serving
- Google Vertex AI endpoints
- Hugging Face
- Mistral AI
- NVIDIA Triton Inference Server with TensorRT-LLM (open source)
- OpenAI
- Ray Serve with vLLM (open source)

- vLLM (open source)

Function calling (includes select models from the below model developers):

- Anthropic Claude
- Databricks DBRX
- Google Gemini
- Llama 3 70B Instruct (open source)
- Mistral AI
- NexusRavenV2 13B (open source)
- OpenAI GPT

Key Characteristics

Model execution components have the following characteristics:

- **Consistent logging, monitoring and guardrails:** An enterprise RAG implementation framework requires a uniform approach to logging, monitoring and implementing guardrails across all use cases. While a common services framework can provide these specific functions, a centralized invocation of these services through the AI gateway ensures that all LLM API calls are validated and logged before being passed to an external LLM. At a minimum, this acts as a safeguard against any privacy or safety risks.
- **Model serving framework:** This framework manages the execution of multiple models of varying sizes and resource requirements. It must integrate with resource management systems like Kubernetes to allow flexible allocation of compute and memory resources, manage instances and optimize overall resource usage.
- **Function calling integration:** While not required for every use case and not supported by all LLMs, function calling adds extensibility to the solution. It enables the invocation of various services, potentially extending a conversational front end to trigger a flexible workflow defined at runtime. Function calling leverages instructions provided in the prompt for actions. Use cases requiring function calling should integrate prompt templates via the prompt template catalog.

Related Architecture Research

- Hype Cycle for AI in Software Engineering, 2024
- Innovation Insight: AI Agents

RAG Monitoring

To effectively monitor machine learning (ML) model-focused solutions, we track data drift and concept drift. However, these metrics are insufficient for monitoring LLMs. LLM data is vast, largely unstructured and primarily sourced from various web resources. Additionally, their predictions are unstructured and do not follow predefined distributions. Therefore, monitoring LLM-based solutions requires additional metrics.

A RAG implementation includes several components: the user prompt, the system prompt, the context based on the organizational content search and the LLM response. Monitoring such a solution requires capabilities to measure and evaluate both technical metrics and user satisfaction. A comprehensive monitoring solution includes the following components:

- **Logging:** Log all incoming requests, output responses and intermediate steps for prompt creation and LLM API calls, including request payloads. This enables troubleshooting and system performance analysis to identify optimization opportunities.
- **Prompt and response analytics:** Analyze user queries, generated prompts and responses to understand query patterns and topics. This analysis can help identify outlier topics that may require additional source data or user guidance if the topics do not align with the use case or violate organizational policy. This serves as a leading indicator to identify gaps in the solution.
- **User feedback:** Integrate the collection of explicit user feedback within the solution. While this is a lagging indicator, combining it with prompt and response analytics provides a more holistic view of the efficacy of the RAG solution.
- **Metrics:** Measure system performance to gauge the effectiveness of the solution. Metrics may include operational metrics, such as response time, throughput, resource utilization and error percentage; and performance metrics, such as accuracy, user satisfaction, session length and prompt length.

- **Usage monitoring:** Track usage to drive change management efforts, assess the effectiveness of the solution and manage costs, including implementing a chargeback model.

Example Technologies

RAG monitoring:

- Aporia
- Arize AI
- Dataiku
- DataRobot
- Domino Data Lab
- Evidently AI (open source)
- Fiddler
- Galileo
- Helicone
- LangChain (open source)
- Langfuse (open source)
- Scale AI
- Splunk
- Superwise
- TruEra
- Weights & Biases
- WhyLabs

Key Characteristics

RAG monitoring components have the following characteristics:

- **Combination of centralized services and customizations:** Monitoring implementations blend common centralized services with use-case-specific customizations. Common services may include logging, metrics, topic analysis and usage analysis, while user feedback collection may require a tailored approach for each use case.

- **Privacy controls:** Consider privacy constraints when logging details. The user prompt or prompt context may contain PII or other confidential information, which could be included in the logs. Therefore, logging should have different levels based on content classification and access controls to ensure data privacy.

- **Comprehensive dashboard:** Consolidate all monitoring components into a comprehensive dashboard that analyzes and provides insights into system health, user feedback, usage and enabling troubleshooting aids for gaps and anomalies. This centralized view facilitates effective monitoring and management of the RAG system.

- **Integration with evaluation functions:** Evaluation of a RAG solution includes assessing the input prompts and output responses. While this is focused at testing, the same checks may be applied at runtime for alerts for any deviations or failures. The integration of evaluation functions or scripts within the monitoring framework allows for operationalization of these tests.

Related Architecture Research

- Introduce AI Observability to Supervise Generative AI

Guardrails

LLMs are powerful tools for generating humanlike text in response to user queries, and enterprises aim to harness this capability to integrate their content with LLMs. However, LLMs can sometimes produce “hallucinations” — confidently stated information that is incorrect or fabricated. This can lead to misleading or false information, especially in critical applications like healthcare or legal advice. Addressing hallucinations requires implementing guardrails to ensure the accuracy and reliability of the generated content. While hallucinations are a challenge, additional guardrails are also necessary:

- **Safety:** LLM guardrails for safety ensure the responsible and ethical use of large language models by blocking inappropriate usage through user queries. These guardrails are based on usage policies and the topical scope of the use case.

- **Privacy:** LLM guardrails for privacy protect sensitive information and ensure compliance with data protection regulations. Users may include such information within the prompt, creating the risk of it being shared with an external network or provider. Retrieved context may include privacy attributes as well. Specialized models trained to identify PII attributes within the text can mask, redact or block such attributes or the entire prompt.

- **Relevance:** LLM guardrails for relevance ensure that the user query and generated outputs are contextually appropriate and within the topical scope of the solution. A response low on relevancy would indicate a gap within the content corpus used for context retrieval.

- **Faithfulness:** LLM guardrails for response faithfulness ensure that the generated outputs are truthful and based on accurate information. Implement fact-checking mechanisms that cross-verify model responses with reliable data sources in real time. This may include validating the response with the context provided in the prompt.

Example Technologies

Guardrails:

- Arize
- DataRobot
- Deepchecks
- Fiddler
- Holistic AI
- Meta Llama Guard
- NVIDIA NeMo Guardrails
- ragas
- Scale AI

Key Characteristics

RAG guardrails have the following characteristics:

- **Problem identification and action definition:** Guardrails identify issues with the prompt or response. Technical teams, in partnership with the business owner or AI governance team, need to define the appropriate actions in response to such alerts. Actions may include logging the alert, redacting the violating information or completely blocking the message.

- **Alert classification framework:** Guardrails should include a framework for classifying alerts based on the use case and impacted stakeholders. For example, a customer-facing use case might classify guardrail-generated alerts or errors as blockers, while an internal knowledge base use case might generate a warning for an off-topic query.

- **Integration with user feedback:** Guardrails should integrate with user feedback. In the absence of user feedback, alerts triggered by guardrails can serve as leading indicators of solution gaps.

- **Scalability and performance:** Guardrails should scale with the volume of user interactions while maintaining system performance and low latency. As the number of use cases expands, centralizing guardrails can provide consistent implementation across the organization.

Related Architecture Research

- Use These 8 Practices to Improve RAG Systems
- Innovation Guide for Generative AI in Trust, Risk and Security Management
- Use Model Guardrails to Regulate Generative AI Output and Behavior

Orchestration

A RAG implementation includes several components along with the prompt flow. The orchestration framework provides interfaces to integrate various components, such as source content, and retrieval systems, such as vector databases, LLMs, guardrails and monitoring tools. In addition, as shown in Figure 1, the user's query is orchestrated through various steps:

- Augmented with the related context
- Audited by applying guardrails
- Passed to the LLM

- Audit of the response by applying guardrails
- Passing the response to the user

Example Technologies

- Databricks
- Dataiku
- DataRobot
- Domino Data Lab
- Haystack (open source)
- LangChain (open source)
- LlamaIndex (open source)

Key Characteristics

Orchestration includes the following characteristics:

- **Layer of abstraction:** The orchestration framework provides a layer of abstraction by providing interfaces to different components. This enables the modularity of the overall solution, while keeping the core solution decoupled from the specific LLM or vector database or guardrails provider.
- **Flexibility:** The orchestration framework must provide flexibility to swap out various components without impacting the solution functionality. In addition, it should enable different solution patterns based on the use-case requirements.
- **Scalability:** The orchestration framework should be scalable to integrate new components as the number of RAG implementations, volume of transactions and composability of solutions increase.

Related Architecture Research

- What Technical Professionals Need to Know About Large Language Models
- Case Study: Composable Generative AI Integration (Ally Financial)

Key Architecture Principles & Patterns

Architects and practitioners must apply the following architecture principles and patterns as they design and implement RAG solutions:

- **Reusability:** Initially, most organizations developed RAG implementations to demonstrate use-case benefits and assess skills and technology. As new use cases emerge and the development focus expands from a core generative AI team to other teams, architecture teams should identify common components that must be reused across various solutions. This may include common logging, monitoring or guardrail services.
- **Modularity:** RAG implementations exist within an innovative and rapidly changing technology ecosystem, which includes a mix of open-source and commercial products, new startups, and established platforms (including cloud service providers [CSPs]). Architects must continually evaluate products and assess their impact on solution implementation. A modular approach allows for the replacement of one product with another without requiring major refactoring.
- **Configuration-driven:** A scalable and flexible RAG architecture should offer choices for LLMs, search engines and related prompt templates. This may also require dynamic chunking strategies and iterative optimizations. A configuration-driven approach to mapping use cases, search parameters, prompt templates and LLMs provides speed and agility in implementations and fine-tuning of the solution.
- **Tiered approach:** Generative AI systems face various risks related to response inaccuracies and privacy concerns. Applying guardrails to user queries and responses is critical for many use cases. As the reliability of guardrails continues to improve, implementing a tiered-approach redundancy in the invocation of guardrails at both the use-case level and a centralized gateway may provide additional layers of safety and security, even if it is not optimal.

Key Architecture Recommendations

To drive consistency, scale and trust in RAG implementations, follow these guidelines:

- **Understand the use case:** Analyze the use case, including expected user queries, to select appropriate functional components and determine performance requirements.
- **Define common architecture patterns:** Establish common architecture patterns and a shared services framework to ensure uniformity across RAG implementations.

- **Embed troubleshooting and audit aids:** Integrate tools for troubleshooting and auditing to analyze errors or incorrect responses. Consistent query responses depend on the integration and configuration of multiple components and data sources.

- **Trust in the system:** Testing, evaluation, monitoring and guardrails are critical to ensure the solution quality and continual working across all feasible inputs. However, open-ended inputs and nondeterministic output can create a trust gap in the system. Ensure that the solution implementation process includes stakeholder engagement supported by process rigor and technical design to imbibe trust.

- **Collaborate with risk and security teams:** Work with risk and security teams to assess potential risks and necessary security constraints. Define the use-case-specific technology stack and service levels accordingly.

- **Regularly evaluate technical decisions:** The technology landscape is constantly evolving. Maintain a regular cadence to review and evaluate technical decisions to ensure they remain current and effective.

Contributors

Joe Antelmi, Wilco van Ginkel, Gary Olliffe

Evidence

[1] **2024 Gartner Growth Agenda Survey.** This study was conducted to better understand which tactics and approaches differentiate high-growth companies (those who achieved revenue growth of 10% or more) from others. The research was conducted online from 4 October through 27 November 2023 among 288 executive leaders from North America (n = 105), Latin America (n = 35), Europe (n = 104) and Asia/Pacific (n = 44), across all commercial industries, excluding information technology. Respondents were from companies with at least $250 million or equivalent in annual revenue. Qualified respondents had personal knowledge of their company's financial performance and either led or participated in their company's growth initiatives. Disclaimer: Results of this survey do not represent global findings or the market as a whole but reflect the sentiments of the respondents and companies surveyed. There are no respondents from China in compliance with the Personal Information Protection Law (PIPL).

[2] **2023 Gartner Voice of the Client Content Survey.** This survey was conducted online with 820 engaged Gartner clients in IT and business leader roles from 9 May to 31 May 2023. The objective of the survey was to better understand client needs, and to gauge use and expectations of generative AI in their organizations. Participants represented a wide range of industries and came from across the world: 56% from North America, 27% from EMEA, 13% from APAC and 4% from Latin America. All participants had recently engaged with Gartner's content on gartner.com (within the last 90 days). Disclaimer: Results of this survey do not represent global findings or the market as a whole but reflect the sentiments of the respondents and companies surveyed.

[3] Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, arXiv.

Recommended by the Author

Some documents may not be available as part of your current Gartner subscription.

Use These 8 Practices to Improve RAG Systems

Reference Architecture Brief: MLOps Architecture

What Technical Professionals Need to Know About Large Language Models

Generative AI Adoption: Top Security Threats, Risks and Mitigations

Hype Cycle for Generative AI, 2024

Innovation Guide for Generative AI in Trust, Risk and Security Management