ContextRAG

Give LLMs the Context They Need

Context Extension Platform Architecture

ContextRAG

Context Extension Platform Architecture Document (Expanded Edition)

ContextRAG is a context-extension platform designed to enhance interactions with Large Language Models by
introducing project-aware intelligence. Instead of relying solely on the knowledge contained inside LLM training
data, ContextRAG retrieves relevant knowledge from documentation, uploaded project artifacts, and past
conversations before sending prompts to an LLM.

The system ensures that AI responses are enriched with contextual information related to the developer’s
project environment, enabling more accurate, relevant, and persistent knowledge interactions.

1. Problem Being Solved

Developers commonly interact with LLMs such as ChatGPT, Claude, or Gemini. However, these systems do not
possess knowledge of the specific project the developer is working on.

As a result:

• Developers must manually copy context into prompts
• Previous conversations are forgotten
• Architectural knowledge is lost between sessions
• The model cannot reason about the developer’s actual codebase

ContextRAG solves this by introducing a retrieval layer and contextual memory system that continuously augments
user prompts with relevant project information.

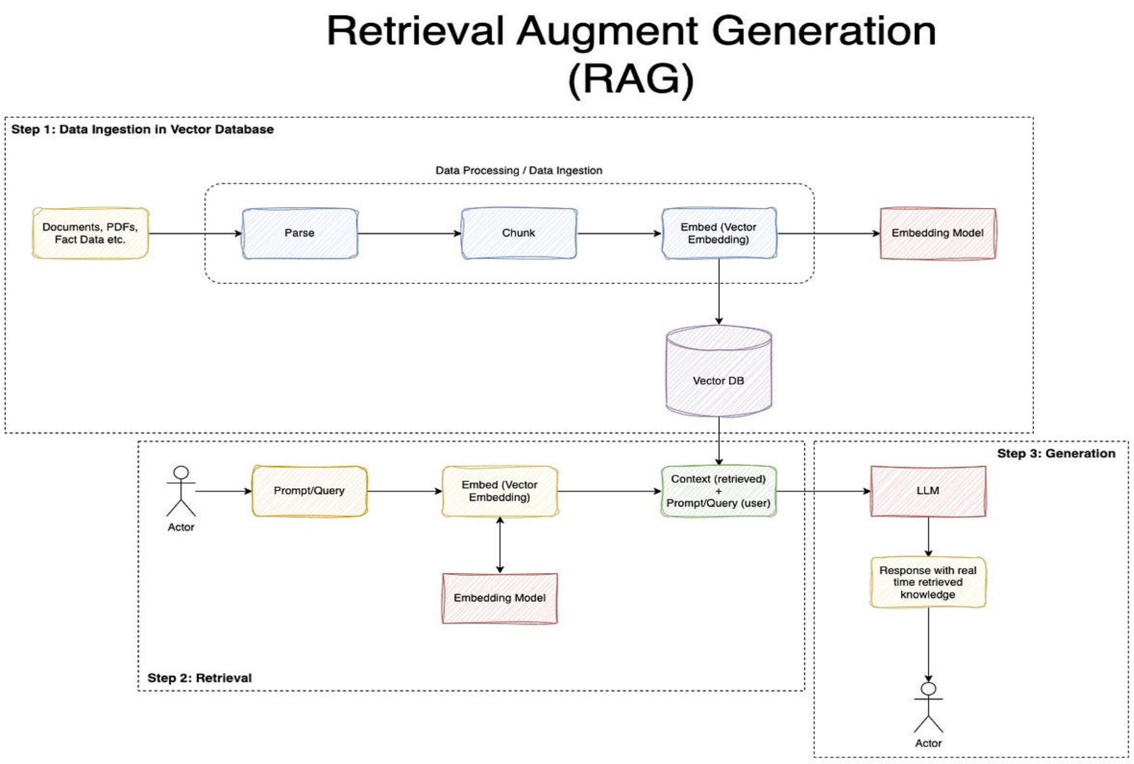
2. Core Architectural Concept: Retrieval Augmented Generation

The platform is based on the Retrieval Augmented Generation (RAG) architecture.

Instead of sending a raw query directly to an LLM, ContextRAG performs the following process:

1. Convert the query into embeddings
2. Retrieve relevant context from a vector database
3. Combine the query and retrieved context
4. Send the enriched prompt to the LLM

This allows the model to generate responses that are grounded in project-specific knowledge.

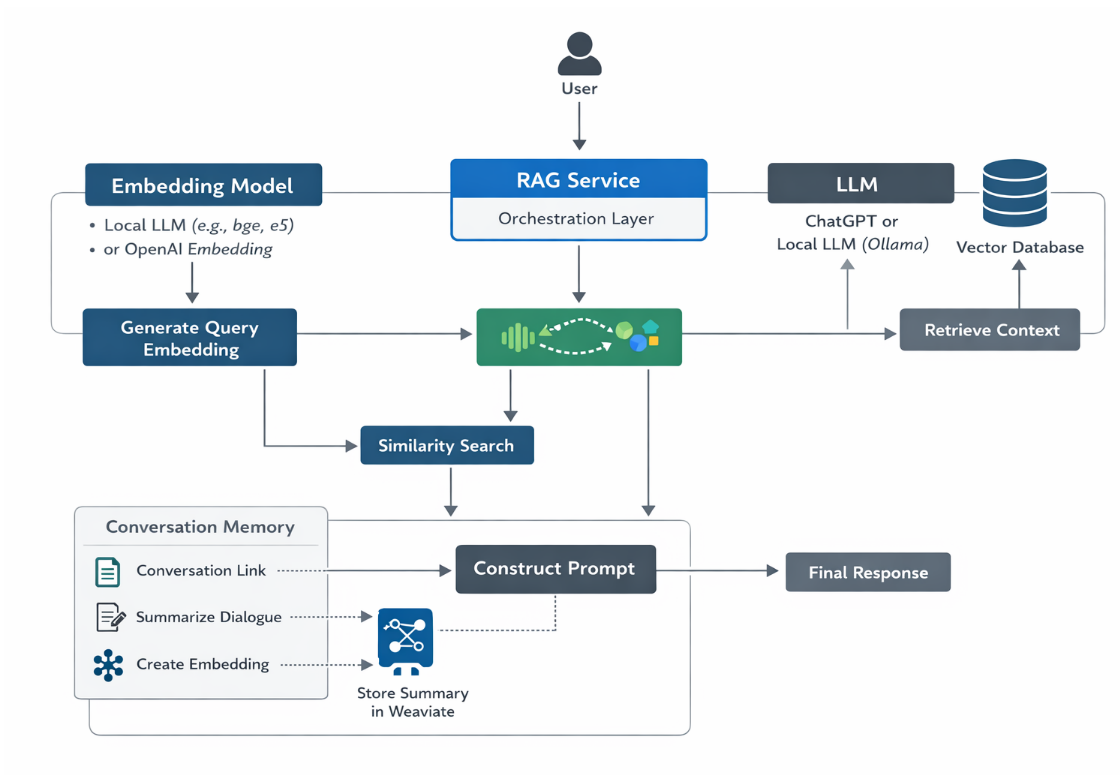

Retrieval Augment Generation (RAG)
Step 1: Data Ingestion in Vector Database
Data Processing / Data Ingestion
Documents, PDFs, Fact Data etc.
Parse
Chunk
Embed (Vector Embedding)
Embedding Model
Vector DB
Actor
Prompt/Query
Embed (Vector Embedding)
Context (retrieved) + Prompt/Query (user)
Embedding Model
Step 2: Retrieval
Step 3: Generation
LLM
Response with real time retrieved knowledge
Actor

3. ContextRAG System Architecture

The ContextRAG platform consists of the following major components:

• Custom Multi-LLM UI
• Java RAG Orchestration Service
• Embedding Model
• Vector Database (Weaviate)
• LLM Providers
• Kafka Event Pipeline
• Conversation Summarization Layer

The Java RAG Service acts as the orchestration layer responsible for retrieving context and constructing prompts.



4. Knowledge Ingestion Pipeline

Developers must be able to upload project knowledge so that the system can use it during retrieval.

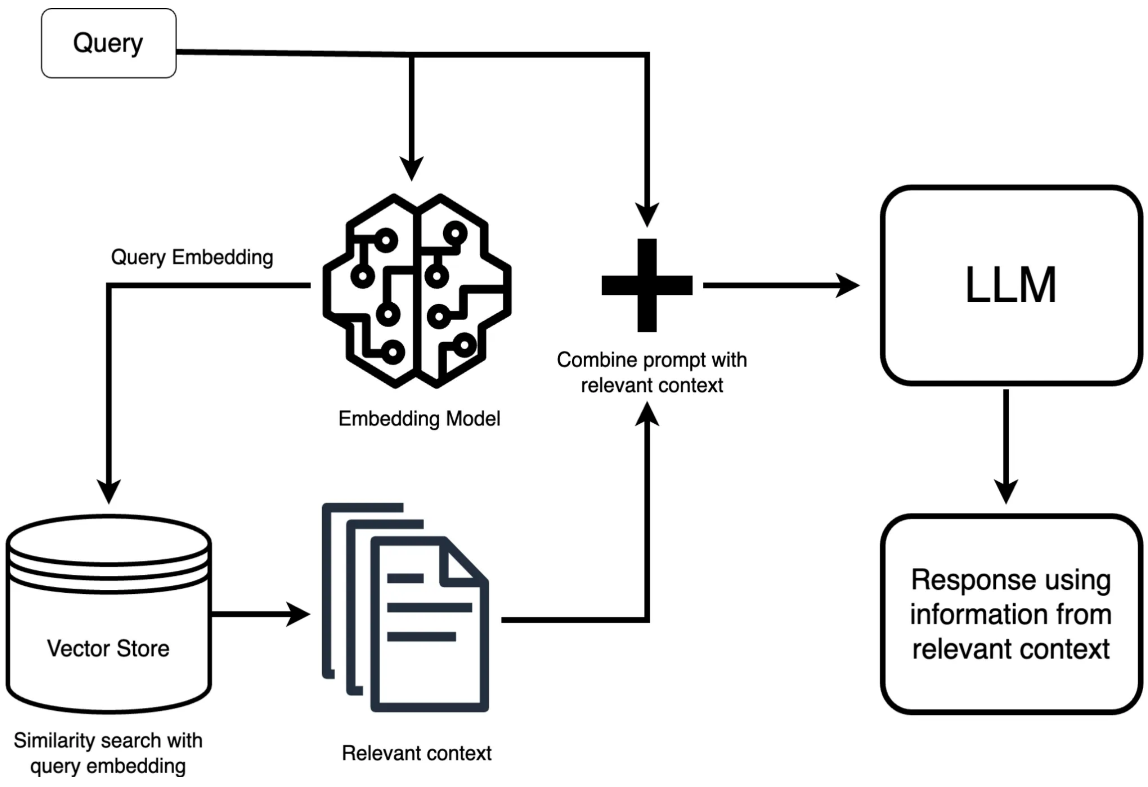
Examples of knowledge sources include:

- documentation
- architecture diagrams
- code snippets
- design notes
- repository files

The ingestion process includes:

1. Parsing documents
2. Chunking the content
3. Generating embeddings
4. Storing vectors inside Weaviate

Once stored, the knowledge becomes searchable during prompt construction.


Query
Query Embedding
Embedding Model
Combine prompt with relevant context
LLM
Vector Store
Similarity search with query embedding
Relevant context
Response using information from relevant context

5. Event-Driven Knowledge Ingestion

Knowledge ingestion is implemented as an asynchronous event pipeline using Kafka.

This allows:

• non-blocking ingestion
• high throughput processing
• horizontal scalability
• integration with CI/CD pipelines

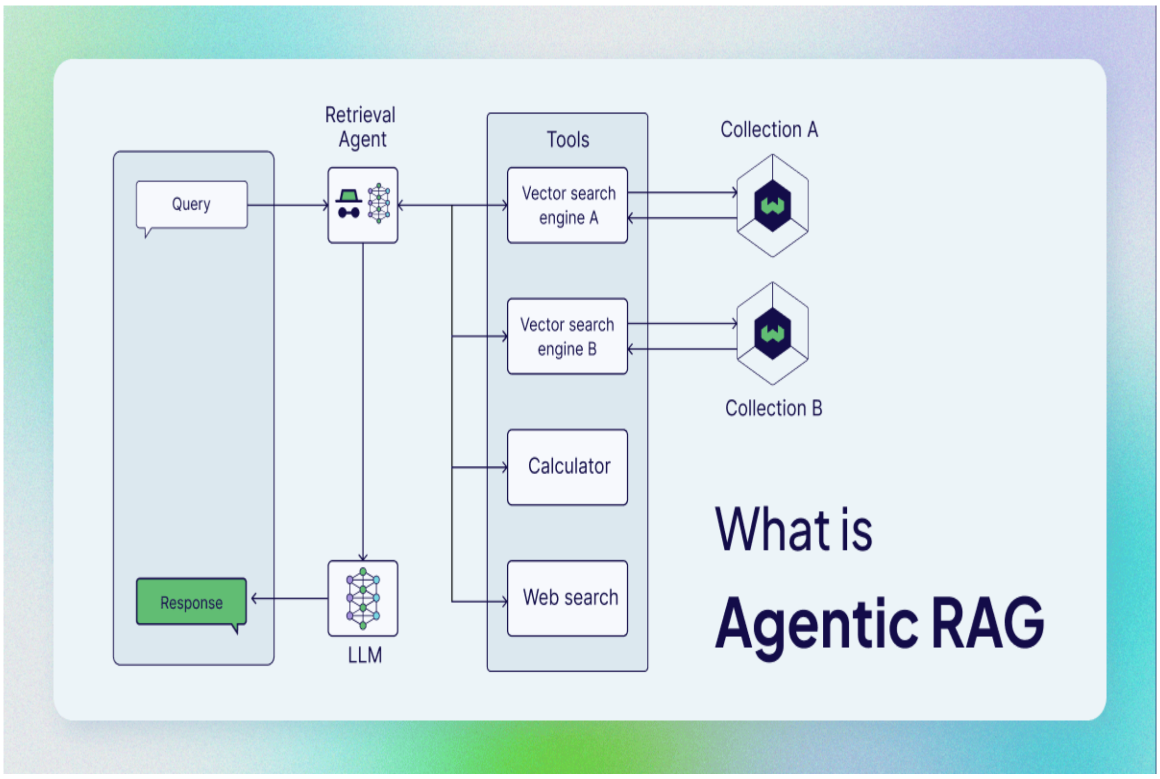
Knowledge sources publish ingestion events to Kafka. Consumers then process these events to generate embeddings
and update the vector database.

6. Conversation Memory Loop

ContextRAG includes a knowledge loop that converts conversations into reusable knowledge.

Every interaction between the user and the LLM generates conversation events which are published to Kafka.
A summarization layer consumes these events, summarizes the conversation, generates embeddings, and stores
the resulting vectors in the vector database.

This enables the system to remember prior discussions and incorporate them into future queries.



7. Multi-LLM Architecture

The platform supports multiple LLM providers to avoid vendor lock-in.

Possible providers include:

- OpenAI
- Claude
- Gemini
- Local LLMs via Ollama

The Custom UI allows developers to select the model provider dynamically.
The RAG service routes requests to the selected model while maintaining the same contextual retrieval pipeline.

8. API-First RAG Service

The Java RAG Service is designed as an API-first system.

This ensures that other clients can integrate easily, including:

- Web UI
- VSCode plugin
- IntelliJ plugin
- CLI tools
- future automation agents

The RAG service becomes the central orchestration layer for all context-aware AI interactions.

9. Scalability Strategy (New Section)

ContextRAG is designed with horizontal scalability in mind.

Key scalability strategies include:

- Stateless RAG orchestration services
- Distributed vector database architecture
- Kafka-based asynchronous pipelines
- Parallel ingestion consumers
- Multi-LLM routing layer

This architecture allows the system to scale from individual developer usage to large engineering organizations.

10. Security Considerations (New Section)

Since ContextRAG may process proprietary source code and architecture knowledge, security becomes a critical architectural concern.

Possible security layers include:

- API authentication and authorization
- encrypted vector storage
- secure document ingestion
- role-based project access
- audit logging for LLM interactions

Future enterprise deployments may integrate identity systems such as OAuth, SAML, or corporate SSO.

11. Deployment Models (New Section)

ContextRAG may support multiple deployment models.

Developer Local Mode:
- Local UI
- Local vector database
- Optional local LLM

Team Deployment:
- Shared vector database
- centralized RAG service
- multi-user access

Enterprise Deployment:
- distributed ingestion pipelines
- large-scale vector clusters
- centralized observability and monitoring

12. Long-Term Vision

While the current goal of ContextRAG is context extension for developers, the platform could evolve into a broader AI engineering intelligence layer.

Future capabilities may include:

- observability data ingestion
- production log analysis
- product analytics insights
- architecture optimization recommendations

This would allow ContextRAG to become an AI-assisted product lifecycle platform.