## **1. Product Requirements Document (PRD)**

### **Product Name: Agentic RAG Workflow Engine**

* **Version:** 1.0
* **Status:** In Development

### **Introduction**

Traditional Retrieval-Augmented Generation (RAG) systems follow a fixed linear path: retrieve, then generate. While effective, this approach struggles with complex, multi-step queries that require reasoning and dynamic adaptation. Developers and AI engineers need a more powerful workflow that can reason about a query, decide on the best retrieval strategy, and synthesize information intelligently.

This document outlines the requirements for an **Agentic RAG Workflow Engine**, a system that uses a multi-agent approach to deconstruct user queries, perform intelligent retrieval from a vector database, and generate accurate, source-cited answers.

### **Goals & Objectives**

* **Primary Goal:** To provide developers with a robust and modular framework for building sophisticated RAG applications that can handle complex queries with higher accuracy than traditional RAG pipelines.
* **Objectives:**
  + Increase the relevance of retrieved context by using an agent to refine and plan the retrieval process.
  + Reduce LLM hallucinations by ensuring the final answer is strictly grounded in the retrieved, cited sources.
  + Provide a clear, traceable workflow that shows how an answer was generated (source documents, agent steps).
  + Enable developers to easily configure and extend the workflow with different models (LLMs, embeddings) and vector databases.

### **User Personas**

* **AI Engineer:** Needs to build and deploy a reliable RAG service for an internal knowledge base. They care about accuracy, scalability, and observability.
* **Developer:** Wants to integrate advanced Q&A capabilities into their application without needing deep expertise in vector databases or prompt engineering. They value ease of use and a simple API.

### **Functional Requirements**

#### **Feature: Command-Line Interface (CLI) Client**

* **Description:** A user-friendly CLI for developers to interact with the engine.
* **User Stories:**
  + As a developer, I want to be able to ask a question from my terminal using a simple command.
  + As a developer, I want to be able to point the CLI to a directory of documents to be indexed.
  + As a developer, I want to receive a generated answer along with the source file names and page numbers it came from.

#### **Feature: Agentic Workflow Orchestrator (MCP Server)**

* **Description:** The core of the system. This orchestrator manages a sequence of "agents" or "steps" to process a query. It's not a rigid pipeline but a workflow manager.
* **User Stories:**
  + As an AI Engineer, I want the system to first use a "Query Analysis Agent" to break down my complex question into sub-questions or identify key entities.
  + As an AI Engineer, I want the orchestrator to then trigger a "Retrieval Agent" based on the analysis to fetch relevant document chunks.
  + As an AI Engineer, I want a final "Synthesis Agent" to take the retrieved context and the original query to generate a coherent, cited answer.

#### **Feature: Intelligent Document Ingestion & Chunking**

* **Description:** A process for indexing source documents into the vector database.
* **User Stories:**
  + As a developer, I want the system to handle common document types like .pdf, .md, and .txt.
  + As an AI Engineer, I want the system to use a semantic chunking strategy (not just fixed-size chunks) to keep related paragraphs together.

#### **Feature: Modular Components**

* **Description:** The ability to easily swap out core components like the LLM, embedding model, and vector database via configuration.
* **User Stories:**
  + As an AI Engineer, I want to switch from using OpenAI's gpt-4 to a local Llama 3 model for generation by changing a single line in a config file.
  + As a developer, I want to switch from using a local ChromaDB to a cloud-based Pinecone vector store without changing the application code.

### **Non-Functional Requirements**

* **Performance:** A query should be processed and an answer returned within 5-10 seconds for a moderately sized knowledge base (<1,000 documents).
* **Scalability:** The system should be designed to handle a growing number of documents in the vector store without a significant degradation in retrieval speed.
* **Observability:** The system must log the entire workflow for a given query: the initial query, the analysis by the first agent, the retrieved chunks, and the final prompt sent to the LLM.

### **Success Metrics 📈**

* **Retrieval Precision/Recall:** Measure the quality of the retrieved document chunks.
* **Answer Faithfulness:** Percentage of generated answers that are verifiably supported by the cited sources.
* **User Satisfaction:** Qualitative feedback from developers on the ease of use and quality of results.