RAG vs AI Agents

A practical explanation of two important AI system patterns for knowledge work, automation, and digital services. This presentation walks through what each pattern is, how it works, when to use it, and what the real-world trade-offs are.

AI ARCHITECTURE | KNOWLEDGE WORK | AUTOMATION

Agenda

What We'll Cover

01

What is RAG?

Retrieval-Augmented Generation explained clearly

02

What is an AI Agent?

Autonomous reasoning and action loops

03

RAG vs Agent at a Glance

Side-by-side comparison of key characteristics

04

When to Use Each

Decision criteria and practical guidance

05

How Each Works

Step-by-step mechanics of both patterns

06

Examples, Risks & Takeaways

Real-world use cases and limitations

What is RAG?

Retrieval-Augmented Generation (RAG) is an AI pattern that enhances a language model by connecting it to an external knowledge base at query time. Instead of relying solely on what the model learned during training, RAG retrieves relevant documents or data chunks and passes them as context to the model before generating a response.

This means the model's answers are grounded in current, specific, and verifiable information — not just static training data. RAG is the foundation of most enterprise knowledge assistants, internal chatbots, and document Q&A systems.

RAG = Retrieve relevant knowledge first, then generate an answer grounded in that knowledge.

What is an AI Agent?

Core Idea

An **AI Agent** is a system where a language model acts as a reasoning engine that can plan, decide, and take actions across multiple steps to complete a goal. Unlike a single-turn Q&A system, an agent loops: it observes the environment, reasons about what to do next, executes a tool or action, and evaluates the result.

Agents can call APIs, run code, search the web, write files, and chain many sub-tasks together — all autonomously.

Key Properties

→ Goal-oriented

Works toward a defined objective across multiple steps

→ Tool-using

Calls external tools, APIs, databases, or services

→ Memory-aware

Maintains state and context across the conversation

→ Self-correcting

Evaluates results and adjusts its approach

RAG vs Agent at a Glance

Understanding where each pattern sits on the spectrum from simple lookup to autonomous action helps you choose the right architecture for your use case.

Dimension	RAG	AI Agent
Core pattern	Retrieve → Generate	Reason → Act → Observe → Repeat
Number of steps	Single turn (one retrieval)	Multi-step, dynamic loop
External tools	Vector DB / document store only	APIs, code, files, browsers, DBs
Autonomy level	Low — deterministic pipeline	High — self-directed reasoning
Predictability	High — consistent outputs	Lower — emergent behaviour
Typical latency	Fast (seconds)	Slower (multi-step processing)
Best for	Q&A, document search, support bots	Automation, research, workflows

When to Use RAG

RAG is the right choice when your primary goal is accurate, knowledge-grounded responses drawn from a defined corpus of documents or data. It excels in scenarios where the answer already exists somewhere and the challenge is finding and presenting it clearly.

📄 Document Q&A

Internal policies, manuals, legal contracts, product documentation

🎧 Customer Support

Answering FAQs grounded in your knowledge base, reducing hallucinations

🔍 Enterprise Search

Semantic search across HR, finance, or technical documents

📰 Content Grounding

News summaries, research assistants, report generation with citations

✓ Use RAG when: the task is question-answering, the knowledge source is well-defined, and you need consistent, auditable results.

When to Use an AI Agent

Agents shine when the task requires multiple decisions, tool use, or real-world actions that cannot be solved in a single retrieval step. If the path to the answer is unknown upfront, an agent can plan its own route.

Process Automation

Booking systems, order processing, multi-system data entry that spans several tools

Deep Research

Multi-source research tasks: searching, summarising, comparing, and synthesising findings

Code Generation

Writing, running, debugging, and iterating on code based on high-level requirements

Complex Workflows

Multi-step business processes requiring conditional logic and dynamic decision-making

Use an Agent when: the task involves multiple steps, requires tools, or the exact path to completion is not known in advance.

How RAG Works

RAG follows a clean, linear pipeline. Each stage transforms the user's question into a grounded, context-rich answer. Understanding this flow helps you diagnose problems and optimise each component independently.

Indexing (Offline)

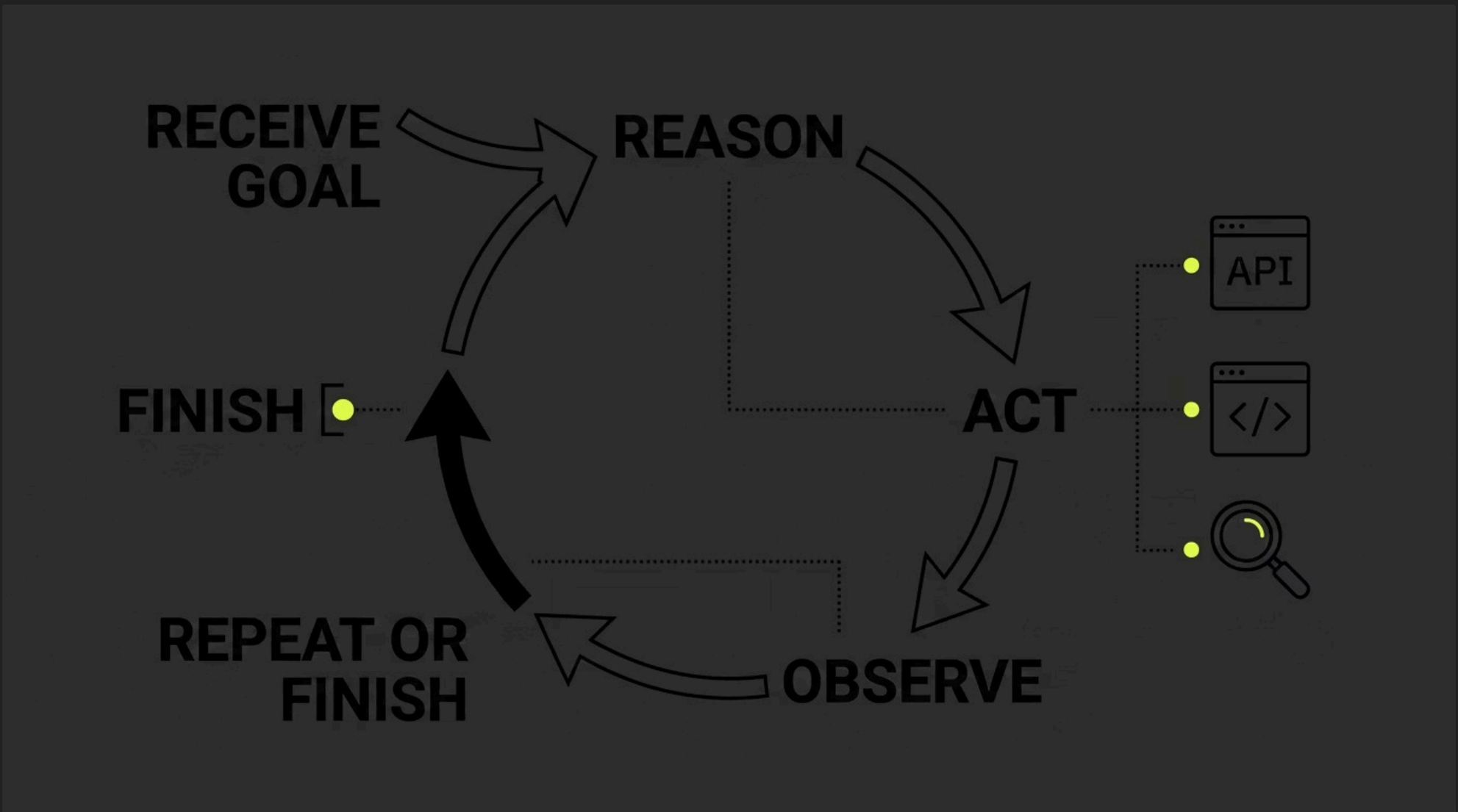
Documents are split into chunks, converted to vector embeddings, and stored in a vector database. This happens once (or periodically) and enables fast semantic search at query time.

Querying (Online)

The user's query is embedded and compared against stored vectors. The top-k most relevant chunks are retrieved and inserted into the LLM's prompt as additional context before the model generates its answer.

How Agents Work

Agent architectures are built around a ReAct loop — a continuous cycle of Reasoning and Acting. The agent does not follow a fixed script; it decides what to do next based on what it has observed so far.



Each cycle the agent refines its understanding. Tools available to the agent — search, calculator, code executor, file writer, web browser — are declared upfront. The model decides which tool to call, with what parameters, and what to do with the result.

Chapter 3

Practical Examples

Seeing both patterns applied to real scenarios makes the architectural difference intuitive. Below, the same domain — a company's internal knowledge — is served by two different patterns depending on what the user actually needs.

RAG in Practice: HR Knowledge Assistant

Scenario

An employee asks: *"How many days of annual leave am I entitled to after 3 years of service?"*

What RAG Does

1. Converts the question to a vector embedding
2. Searches the HR policy document store semantically
3. Retrieves the relevant leave entitlement clause
4. Injects it into the LLM prompt as context
5. LLM generates a clear, policy-grounded answer with reference

Why RAG is the Right Choice Here

Single-turn answer

No multi-step reasoning needed — retrieve and respond

Grounded in policy

Answer cites the actual HR document, reducing risk of hallucination

Auditable

Source chunk is traceable — compliance teams can verify the answer

Fast and predictable

Consistent responses across thousands of queries

Agent in Practice: Automated Research Report

A manager asks: *"Give me a competitive analysis of our top three rivals — pricing, product features, and recent news."*

1

Plan

Agent breaks task into sub-goals: search competitor websites, retrieve pricing pages, find recent news

2

Search

Calls web search tool three times — once per competitor — and stores intermediate results

3

Analyse

Reads retrieved content, extracts relevant data points, identifies gaps and re-queries if needed

4

Synthesise

Produces a structured markdown report with comparison table and key insights

This task is impossible for a standard RAG pipeline — it requires dynamic planning, multi-tool use, and iterative self-correction. Only an agent architecture can handle it end-to-end.

Risks and Limitations

Both patterns carry distinct risks that must be understood before deploying in production. Architectural strength in one area often creates vulnerability in another.

RAG Risks

Retrieval Quality

If the wrong chunks are retrieved, the answer will be confidently wrong. Embedding quality and chunking strategy are critical.

Context Window Limits

Only a limited number of chunks fit into the prompt. Long documents may lose important detail.

Stale Knowledge

The vector index must be kept up-to-date. Outdated documents lead to outdated answers.

Agent Risks

Unpredictable Behaviour

Multi-step reasoning can produce unexpected paths or decisions — hard to debug and reproduce.

Tool Misuse

Agents can call tools incorrectly, over-use them, or take unintended real-world actions (e.g. sending emails).

Cost & Latency

Each tool call adds latency and LLM token cost. Long agent loops can become expensive.

Can RAG and Agents Work Together?

Absolutely — and in practice, the most powerful systems combine both. RAG becomes one of many tools an agent can invoke. The agent decides when to retrieve from the knowledge base, when to call an API, and when to write code.

RAG as a Tool

The agent calls the RAG retriever as a named tool: `search_knowledge_base("leave policy")`. The retrieved context flows back to the agent's reasoning loop.

Complementary Strengths

RAG provides grounded, factual answers from documents. The agent provides planning, tool orchestration, and multi-step reasoning. Together they handle tasks neither could alone.

Agentic RAG Pattern

Advanced architectures use query rewriting, iterative retrieval, and self-reflection — where the agent evaluates whether retrieved chunks are sufficient before generating the final answer.

Key Takeaways

Choose the pattern that matches the complexity and autonomy requirements of your use case. When in doubt, start simple.

🔎 RAG = Grounded Answers

Use RAG when you need accurate, document-backed responses. It's fast, predictable, and auditable — ideal for knowledge assistants and support bots.

🤖 Agent = Autonomous Action

Use agents when tasks require planning, multi-step execution, and tool use. Expect higher power — and higher complexity and cost.

🔗 Combine Them

The strongest production systems layer RAG inside agent architectures — grounded knowledge retrieval within a flexible reasoning loop.

⚠️ Know the Risks

RAG fails on retrieval quality; agents fail on unpredictability and cost. Mitigate both through evaluation, monitoring, and human-in-the-loop design where it matters.

> Start with RAG. Add agentic behaviour only when the task genuinely requires it. Complexity should be earned, not assumed.