Progress® Agentic RAG

Build vs. Buy: The Reality of Production-Grade RAG

WHITEPAPER

Introduction

Most teams can assemble a retrieval-augmented generation (RAG) pipeline. With today's tools, it's relatively straightforward to connect internal content to a large language model (LLM) and generate grounded answers. A basic setup including ingesting documents, embedding them, retrieving relevant passages and passing them to an LLM can often be built quickly and demonstrate value early on.

Where teams struggle is not getting RAG to work but getting it to keep working. As soon as real users begin asking unpredictable questions, new data sources are introduced or new use cases are added, the system's complexity increases. Retrieval behaves differently depending on the query, content can change over time and small adjustments to improve one use case can quietly degrade another. What once looked like a simple pipeline becomes an extremely complex system that is difficult to adjust.

For technical teams, this is the moment when RAG starts to resemble shared infrastructure, rather than an application feature. Engineers are suddenly responsible for reliability, relevance, performance, cost control and trust often across multiple teams and use cases. For business leaders, this shift shows up as slower progress, inconsistent answers and growing uncertainty about whether AI can be relied on in real workflows.

DIY RAG implementations stall when they attempt to scale beyond a single team or application. Without a unifying control plane for ingestion, retrieval, evaluation and governance, RAG fragments into duplicated pipelines and bespoke logic. Knowledge becomes siloed again, just this time behind AI interfaces.

At the prototype stage, DIY RAG appears inexpensive and low risk. A small team can assemble a pipeline using open-source components, cloud services and an LLM API, often with minimal upfront investment. Early results are promising—answers are more relevant, demos succeed and confidence grows that the approach will scale.

The problem is that the cost and risk of DIY do not grow linearly; they compound as RAG moves from a single use case into production and the operational surface area expands rapidly. What was once a simple pipeline becomes a system that must be continuously tuned, monitored, defended against failure and proved that it evolve at the same pace the technology is advancing.

The first inflection point emerges when RAG is exposed to real usage. Query patterns become unpredictable and retrieval quality degrades across different types of questions. Engineering time shifts from building new features to diagnosing relevance issues, managing ingestion failures and reindexing content. And each fix tends to be local, increasing system complexity and making future changes riskier.

A second inflection point emerges when RAG is reused across multiple teams or applications. Pipelines are duplicated to avoid breaking existing workflows. Retrieval logic diverges to satisfy different use cases. Evaluation become inconsistent, and it becomes difficult to explain why one application produces reliable answers while another does not. Cost increases—not just in infrastructure, but in coordination, debugging and lost velocity.

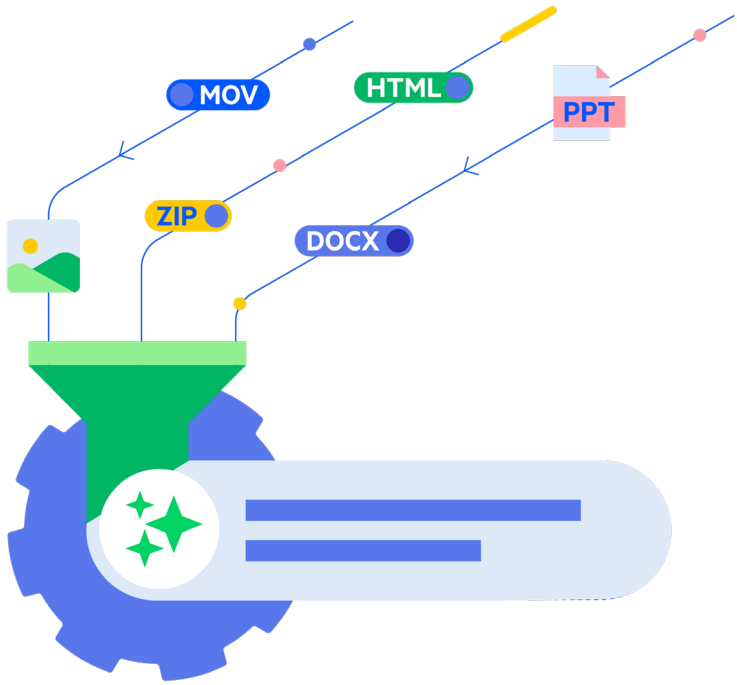
At this stage, risk becomes as significant as cost. Small changes, like upgrading an embedding model or adjusting chunking logic, can have unpredictable downstream effects. Without built-in evaluation and a trust mechanism, teams often detect issues only after users lose confidence. Thus, what started as a flexible DIY solution begins to feel fragile.

Why RAG Has Grown Increasingly Important for Enterprise AI Strategy

As organizations move from experimenting with AI to using it in real business workflows, many discover the same challenge: The model itself is rarely the problem. Modern AI models are powerful and continue to improve, however, enterprise success depends far more on whether those models have access to the right information at the right time. This is where RAG becomes critical.

Over the past few years, leading AI models have become increasingly similar in their overall capabilities. While they may differ in cost, speed or deployment options, they generally perform at a comparable level for common tasks like summarization, question answering and content generation.

What does not improve at the same pace is how well these models understand a company's internal knowledge. Two AI systems using the same model can produce very different results, depending on how effectively they retrieve relevant information. In practice, the quality of answers depends less on which model is used and more on how well the system finds and supplies the right content. As a result, retrieval quality has become the main factor separating useful enterprise AI from unreliable experiences.



Why Fine-Tuning Isn't Enough

Fine-tuning is the process of taking a pretrained AI model and further training it on a specific set of examples, so it behaves more consistently for certain tasks or domains. Instead of giving the model access to live information, fine-tuning adjusts how the model responds based on patterns learning during training.

Fine-tuning is often suggested as a way to make AI models more accurate, but it does not address several core enterprise requirements:

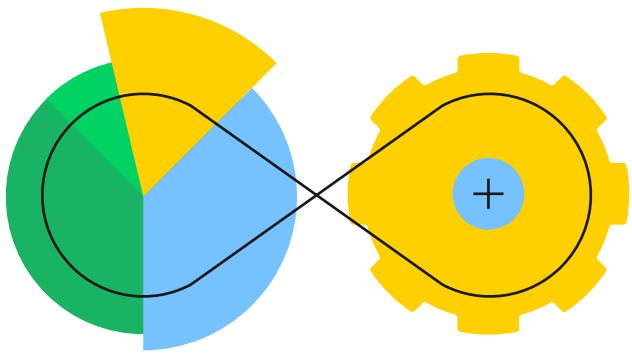
1. Fine-tuned models become outdated quickly. Enterprise information changes constantly—policies are updated, documents evolve and new data is created every day. Updating a model every time this happens is slow, expensive and impractical.

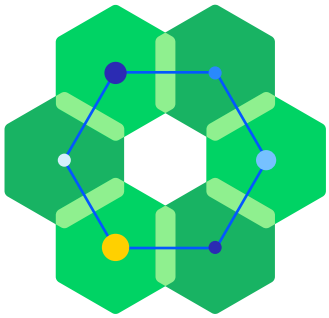
2. Fine-tuning reduces transparency. When knowledge is baked into a model, it becomes difficult to explain where an answer came from or whether the information is still valid. This lack of visibility makes it hard for organizations to trust AI outputs, especially in regulated or customer-facing environments.

3. Fine-tuning cannot enforce access controls. AI models do not understand which users are allowed to see which information. Without retrieval from governed data sources, there is no reliable way to affirm that responses respect permissions, privacy or compliance rules.

RAG solves these problems by keeping knowledge outside the model and retrieving it in real time. Instead of relying on what a model was trained on weeks or months ago, RAG pulls the most relevant and up-to-date information at the moment a question is asked. This produces answers that reflect current knowledge and respect existing access controls.

As AI becomes more deeply embedded in enterprise systems, the ability to manage and govern knowledge matters more than incremental improvements in model performance. RAG allows organizations to treat knowledge as a shared, controllable asset, rather than an invisible part of a model.





The Technical GAP Between Consumer AI and Enterprise AI

Consumer AI	Enterprise AI
Stateless chat	Stateful systems
No permissions	Enforced access control
Opaque answers	Observable & cited
Low risk	High accountability

Meanwhile, the demand for AI experiences in organizations is growing. Consumer AI tools (like ChatGPT) have reshaped expectations around how easy it is to interact with AI. Asking a question and receiving a fluent response feels intuitive and instantaneous. However, the technical foundations that make consumer AI effective for individual use do not translate directly to enterprise environments. The gap between these two becomes clear as soon as AI is expected to operate inside real business systems.

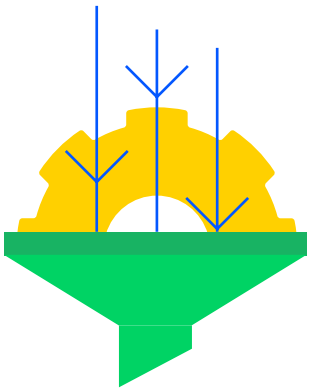
To meet these demands, enterprise AI must rely on three core capabilities: deterministic retrieval, observable answers and source attribution. Deterministic retrieval paths facilitate answers that are grounded in approved sources and produced consistently. Observability into answers allows teams to understand how responses were generated and diagnose issues when something goes wrong. Explicit source attribution provides transparency and trust by showing where information came from and a bolstered ability to respect permissions and compliance requirements.

RAG addresses this gap by grounding AI responses in governed, observable knowledge enabling AI that organizations can trust and operate at scale.

What Production-Grade RAG Means

In an enterprise setting, production-grade RAG is not defined by whether a system can generate fluent answers, but by whether it can do so reliably, repeatedly and transparently as data, users and use cases scale. A RAG system becomes production-grade when it operates as durable infrastructure, able to absorb change without requiring continuous reengineering.

This requires several capabilities that are often missing from early-stage or DIY implementations.



Hybrid Retrieval

No single retrieval method works for all queries. Vector search excels at understanding meaning and knowledge graphs add structure and relationships, while keyword search provides precision and metadata filtering enforces context and access rules. Production-grade RAG combines these approaches, selecting and blending retrieval methods to facilitate both recall and relevance across diverse question types.

Citation Traceability

Enterprise AI must be able to explain itself. Production-grade RAG requires clear traceability from generated answers back to the exact passage that informed them. This enables users to verify information and support compliance and auditing practices, while allowing teams to understand and improve system behavior over time.

Separating Retrieval from Models

Production-grade RAG cannot be locked into a single LLM. An LLM-agnostic design separates retrieval, evaluation and governance from generation, allowing models to be swapped or upgraded without rebuilding the system. This flexibility enables enterprises to adapt to changes in model performance, cost and availability, while keeping their knowledge infrastructure stable

Built-In Evaluation

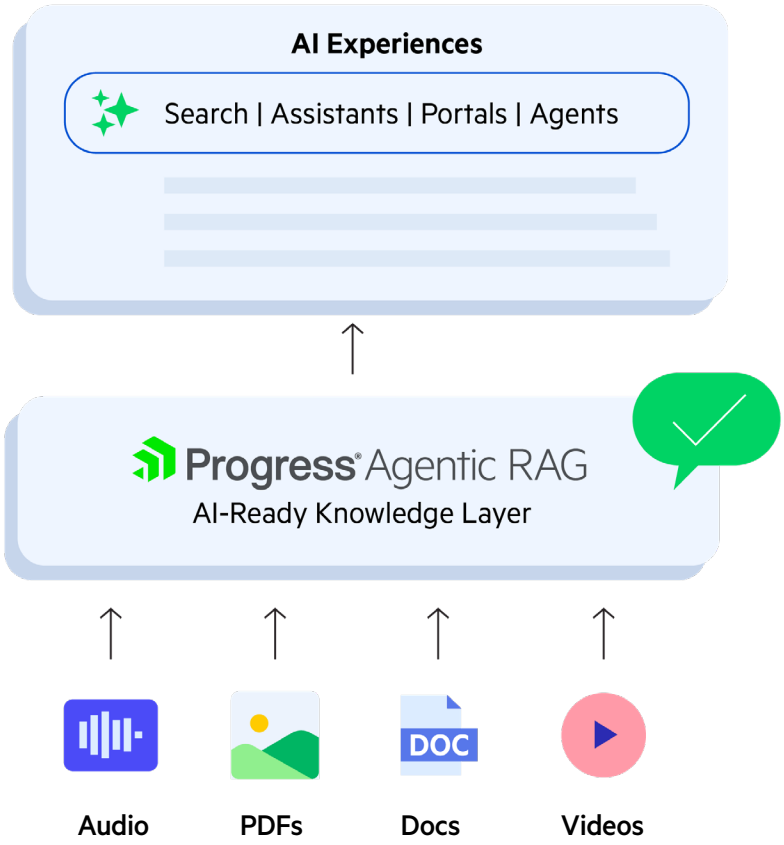
In production, RAG systems must be measurable, not assumed to be correct. RAG evaluation metrics evaluate how well retrieved content supports generated answers by scoring relevance, coverage and grounding. This allows teams to detect degradation, validate changes and improve retrieval quality over time, without relying on manual review or user feedback alone.

Security, Governance and Compliance

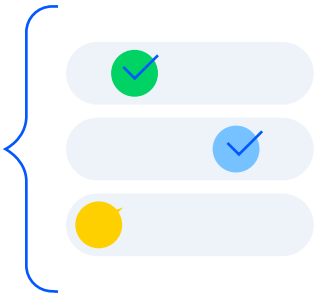
Security and governance must be enforced by RAG systems at the point of retrieval, not after an answer is generated. This includes role-based access control (RBAC), tenant isolation and consistent permission checks across all data sources. Governance and compliance requirements, such as auditability, data residency and source traceability, must be built into the core RAG workflow to generate AI outputs that are accurate, authorized and defensible. Without these controls embedded in the system, enterprises are forced to rely on manual reviews or downstream filtering, increasing risk as AI use expands

One Knowledge Foundation, Many AI Experiences

RAG must support more than a single use case to become production-grade. Search, assistants, copilots and autonomous agents all rely on the same underlying knowledge, but require different retrieval strategies, context assembly and response formats. A modular RAG architecture allows these experiences to be built on a shared knowledge foundation, without duplicating pipelines or hard-coding logic for each application.



This modularity is a requirement for scale. By separating ingestion, retrieval, tuning and evaluation into configurable components, teams can introduce new AI experiences, adapt retrieval behavior and evolve use cases without reingesting data or rebuilding systems. The result is faster innovation, lower operational risk and platform growth for an enterprise's needs, rather than fragmenting under them.



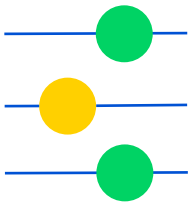
DIY RAG: Assembly Required

Once you define what production-grade RAG requires, the DIY reality becomes clearer: You're not building a single pipeline, you're building a shared infrastructure. Early prototypes hide this complexity because they will work with one dataset, one retrieval method and a narrow set of questions. In production, each layer becomes a subsystem with its own failure modes, tuning knobs and operational burden.

DIY RAG Checklist

- Data Connectors + Ingestion Sync
- Document Parsing + Content Extraction
- Chunking + Segmentation Utilities
- Name Entity Recognition
- Embedding Model + Embedding Lifecycle Management
- Vector Store (Semantic Index)
- Keyword Search (Lexical) + Hybrid Retrieval (Vector + Keyword + Metadata)
- Ranking, Fusion, Reranking
- Prompt Assembly + Context Budgeting
- Citation Traceability (Passage-Level Grounding)
- Evaluation + Regression Testing
- Observability (Logs, Traces, Metrics)
- Security, Governance and Policy Enforcement
- Model Serving (if self-hosting LLMs)

Getting a DIY RAG system off the ground—even a simple one—requires assembling several software components that work together as a cohesive pipeline. At first glance, many of these pieces seem familiar to modern application stacks, but when combined, they form an integrated system that must be maintained, scaled and debugged over time.



Below are the core software components necessary to build RAG yourself:

1. **Data Connectors and Ingestion Frameworks:** These tools extract content from source systems, such as file repositories, databases, SaaS tools or internal applications, and keep that content synchronized over time. They handle scheduling, incremental updates, retries and error handling when systems change or go offline. Without robust ingestion, RAG systems quickly fall out of date.

2. **Document Parsing and Content Extraction:** These tools convert raw files (PDFs, Word docs, HTML pages, emails, etc.) into structured text and metadata that downstream systems can work with. They attempt to preserve layout, headings, tables and document boundaries. Poor extraction leads to low-quality chunks, broken context and irrelevant retrieval, even if the rest of the RAG pipeline is well designed.

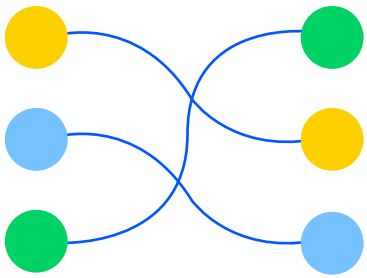
3. **Chunking and Segmentation Utilities:** Chunking tools split documents into retrievable units using fixed-length, semantic or hierarchical strategies. They often enrich chunks with metadata such as section titles, timestamps or document hierarchy. Chunk size and structure directly affect retrieval accuracy. Poor chunking causes either missing context or noisy results that overwhelm the model.

4. **Embedding Models and Lifecycle Management:** Embedding models transform chunks into vector representations for semantic search. Lifecycle tooling manages batch processing, re-embedding when models change and compatibility across versions. Embedding upgrades are inevitable. Without lifecycle management, teams face costly reindexing or inconsistent search behavior.

5. **Vector and Keyword Indexing Engines:** These systems store embeddings and keyword indexes, support similarity search, filtering and ranking, and scale retrieval across large datasets. Index performance affects latency, recall and system reliability. Production systems often require multiple index types working together.

6. **Retrieval Strategy and Orchestration:** Retrieval logic decides how to search, whether that be semantic, keyword, hybrid, filtered, etc., and when to use each approach based on the query, metadata and permissions. It's important to note that no single retrieval strategy works for all queries. Orchestration logic becomes increasingly complex as use cases expand.

7. **Ranking, Fusion and Reranking:** These components combine results from multiple retrieval paths and reorder them to surface the most relevant evidence. They may apply business rules, relevance scores or learned rerankers. Raw retrieval results are rarely optimal. Ranking quality often determines whether RAG answers feel accurate or misleading.



8. **Prompt Assembly and Context Budgeting:** This layer selects which retrieved passages are passed to the LLM, formats prompts and informs responses that stay within token limits while preserving context. Poor context assembly leads to hallucinations, truncations or irrelevant answers, even when retrieval is correct.

9. **Language Model Integration:** This layer connects the system to one or more LLMs, handling authentication, retries, rate limits, failover and optional model routing. Model APIs change frequently, thus tight coupling here makes the entire system fragile and hard to evolve.

10. **Citation Tracking and Traceability:** Citation systems track which chunks or passages were used to generate an answer and attach references to the output for transparency and auditability. Without traceability, enterprises cannot verify accuracy, enforce compliance or explain AI behavior.

11. **Evaluation and Regression Testing:** Evaluation tooling measures retrieval relevance, grounding and consistency over time. Regression tests make it so changes don't degrade performance. RAG systems degrade silently without measurement; evaluation is the only way to operate RAG safely and at scale.

12. **Observability, Monitoring and Logging:** This layer collects metrics, logs and traces across ingestion, retrieval and generation to diagnose failures and performance issues. Without observability, teams cannot debug relevance problems or explain outages.

13. **Security, Governance and Policy Enforcement:** Security layers enforce access control, tenant isolation, auditing and compliance rules at query and retrieval time. Enterprise AI must respect permissions and regulations. Governance cannot be added after answers are generated.

14. **Deployment, Scaling and Rollback:** Deployment tooling packages services, manages releases, scales workloads and supports safe rollback when changes introduce issues. RAG systems evolve continuously; without controlled deployment, every update becomes a risk.

Each of these layers may start small, but together they form a complex, interdependent system. In DIY RAG, teams are responsible not only for building these components, but for operating them indefinitely—monitoring changes, fixing breakages and bolstering reliability as usage grows.

The Operational Cost Curve of DIY RAG

DIY RAG often begin as efficient, flexible solutions, where early use cases are supported by a small number of pipelines, limited data sources and a narrow set of retrieval assumptions. At this stage, changes are relatively easy, and the perceived cost of ownership remains low. The challenge is that as RAG expands beyond its initial scope, operational cost and complexity grow faster than usage or value.

The first scaling issue is coupling. In most DIY implementations, ingestion logic, chunking strategies, retrieval methods, ranking rules and prompt construction are tightly intertwined. This coupling makes the system sensitive to change. A modification intended to improve one use case, such as adjusting chunk size or tuning a retrieval parameter, can unintentionally degrade another. As a result, teams become cautious about making changes, which slows iteration and innovation.

The second issue is duplication. To avoid breaking existing workflows, teams often create separate pipelines or indexes for new use cases. Over time, this leads to multiple versions of the same data, inconsistent retrieval behavior and increased maintenance costs. What started as a single RAG system becomes a collection of loosely related systems that must be maintained in parallel.

Increasingly Costly New Use Cases

In a DIY RAG system, new use cases rarely reuse existing logic cleanly. Supporting a different audience, data source or interaction pattern often requires changes across ingestion, chunking, retrieval, ranking and prompting. To avoid breaking existing workflows, teams frequently duplicate pipelines or indexes, increasing operational overhead and long-term maintenance cost with each additional use case.

Changing Retrieval Strategies is High Effort and High Risk

Adjusting retrieval strategies in DIY RAG typically impacts multiple layers of the system at once. Introducing hybrid retrieval, new ranking logic or different filtering often requires reindexing data, rewriting retrieval orchestration and retesting downstream behavior. Without built-in evaluation, teams must rely on manual testing, making retrieval changes slow, risky and difficult to validate.

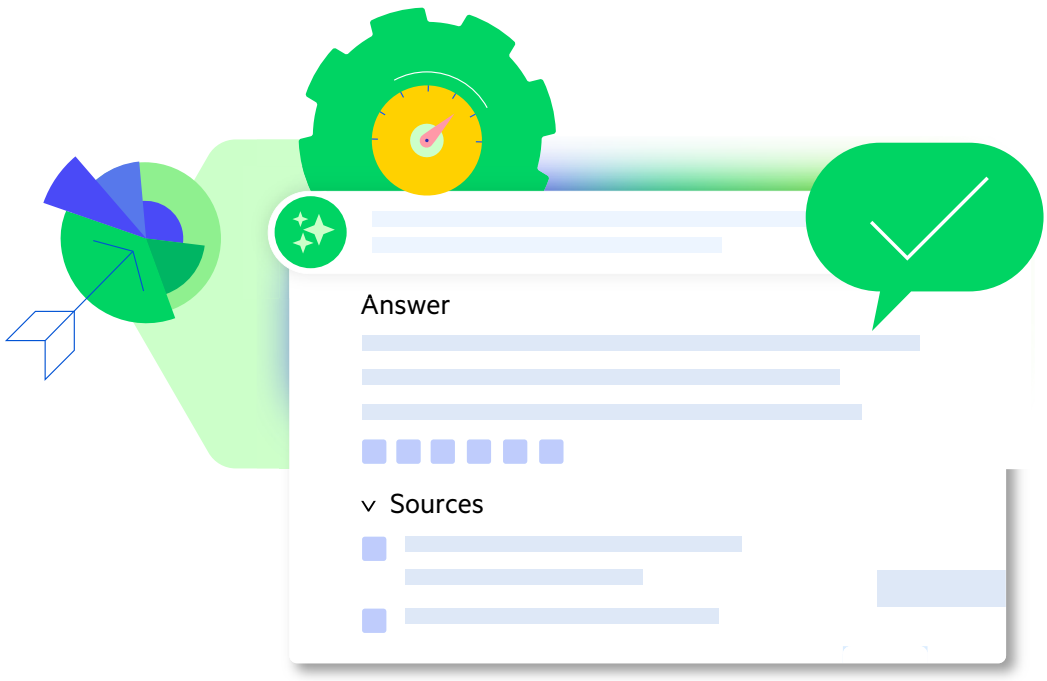
Compounding Engineering Effort

As DIY RAG systems mature, engineering efforts shift from innovation to maintenance. Teams spend increased time managing ingestion failures, re-embedding data, tuning retrieval, monitoring performance and responding to regressions. This ongoing effort grows with system usage, creating a nonlinear cost curve where maintaining RAG infrastructure becomes more expensive than building new AI experiences.

RAG-as-a-Service: A Solution, Not a Shortcut

As RAG systems move into production, many organizations reach the same conclusion: The challenge is no longer building RAG but operating it reliably at scale. DIY approaches offer flexibility early on, but as use cases multiply, the ongoing cost of maintaining ingestion pipelines, tuning retrieval strategies, enforcing governance and measuring quality begins to outweigh the benefits of your customized DIY solution.

RAG-as-a-Service addresses this problem by treating retrieval and knowledge management as shared infrastructure, rather than application-specific code. Instead of each team rebuilding the same components, core capabilities (ingestion, indexing, retrieval, evaluation and governance) are provided as a managed platform that can be reused across multiple AI experiences and as a durable knowledge layer allowing teams to build, iterate and scale AI experiences on top of the same foundation without repeatedly re-engineering the underlying system. This shifts engineering effort away from maintaining RAG systems and instead towards building higher-value applications.

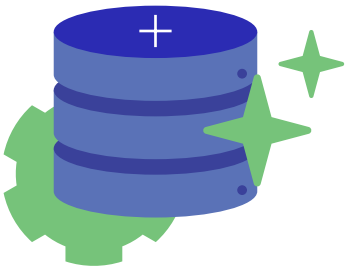


Build vs. Buy: Technical Tradeoffs at a Glance

	DIY RAG (Build)	RAG-as-a-Service (Buy)
Time to initial prototype	Fast for a single use case	Fast and production-ready
Knowledge ingestion	Built and maintained per pipeline	Centralized, persistent ingestion layer
Knowledge reuse across use cases	Limited, often duplicated	Reused across AI experiences
Retrieval strategies	Custom-coded and hard-wired	Configurable, multi-strategy by design
Hybrid retrieval	Added incrementally	Native (vector + keyword + metadata)
Ranking and reranking	Custom implementation per use case	Built in and centrally managed
Citation traceability	Custom logic required	Enforced at passage-level
LLM coupling	Tightly coupled to model choice	LLM-agnostic by design
Effort to change models	High—often disruptive	Low—models can be swapped independently
Security and access controls	Implemented per application	Enforced centrally at retrieval time
Governance and auditability	Added after the fact	Built into core workflow
Operational overhead	Grows nonlinearly with usage	Predictable and centralized
Ability to handle multiple use cases	Limited—typically requires separate pipelines or duplicated logic per use case	Designed to support multiple AI experiences
Readiness for agentic workflows	Requires significant re-architecture	Designed to support agentic patterns

Accelerating AI Strategy with the Progress Agentic RAG Solution

The Progress Agentic RAG solution was designed specifically for this stage of maturity. It provides a production-grade RAG platform with modular ingestion, configurable retrieval strategies, built-in evaluation and enterprise-ready security and governance. By decoupling knowledge infrastructure from individual applications and models, the Agentic RAG solution allows organizations to scale AI use cases with greater reliability, lower operational risk and far less ongoing engineering effort than DIY approaches.



In these DIY approaches, every new AI experience (search, assistants or agents) requires reingesting data, retuning retrieval strategies and revalidating results. With the Agentic RAG solution, knowledge is ingested, indexed, enriched and governed once, then reused across experiences. New applications inherit evaluation metrics and security controls by default, dramatically reducing time to value.

The solution also enables faster iteration by separating knowledge infrastructure from application logic. Teams can evolve retrieval strategies, introduce new ranking or reranking applications approaches and adjust evaluation criteria centrally without redeploying downstream applications. This allows AI experiences to improve continuously as data changes, usage grows and new requirements emerge.

In practice, RAG-as-a-Service with the Progress Agentic RAG solution turns AI from a series of disconnected projects into a reusable platform capability. By providing a persistent knowledge layer with modular ingestion, configurable retrieval, built-in evaluation and enterprise-grade governance, the solution enables organizations to deliver AI experiences more quickly, with far less operational risk than DIY RAG approaches.

Try it today with a 14-day free trial

About Progress Software

Progress Software (Nasdaq: PRGS) empowers organizations to achieve transformational success in the face of disruptive change. Our software enables our customers to develop, deploy and manage responsible AI-powered applications and digital experiences with agility and ease. Customers get a trusted provider in Progress, with the products, expertise and vision they need to succeed. Over 4 million developers and technologists at hundreds of thousands of enterprises depend on Progress. Learn more at www.progress.com

Worldwide Headquarters

Progress Software Corporation
15 Wayside Rd, Suite 400, Burlington, MA 01803, USA
Tel: +1-800-477-6473

facebook.com/progresssw
twitter.com/progresssw
youtube.com/progresssw
linkedin.com/company/progress-software
progress_sw_

Progress