Vespa.ai

Best Practices for Large-Scale RAG Applications

Guided by The RAG Blueprint

We Make AI Work

Vespa.ai

Vespa.ai is an AI Search Platform for developing and operating large-scale applications that combine big data, vector search, machine-learned ranking, and real-time inference. With native tensor support for complex ranking and decisioning, Vespa enables real-time AI applications like RAG, recommendation, and intelligent search—at enterprise scale.

Contents

↗ All contents contain linked titles.

Introduction

As Retrieval-Augmented Generation (RAG) systems scale, the same architectural challenges continue to surface:

- What's the right size for your searchable unit?
- Should you rely on keywords, vectors, or both?
- How do you keep costs under control as the system scales?
- And how do you make ranking accurate *and* efficient?

We've seen these issues firsthand while working with teams like Perplexity, and that's why we created The RAG Blueprint. It's a practical playbook that walks you step-by-step through each stage of system design, helping teams build more accurate, cost-efficient, and production-ready applications.

This ebook distills the highlights from the Blueprint, so you can learn the best practices even if you're not actively building today.

You'll gain a clearer understanding of what it takes to make RAG accurate, scalable, and cost-efficient, including pro tips on how Vespa makes it possible.

Choosing the Right Searchable Unit

One of the first decisions in any RAG system is choosing the right unit of retrieval, because everything else in the pipeline including retrieval accuracy, ranking quality, cost efficiency, and LLM answer quality depends on this choice.

The document unit (sometimes called the searchable unit) means the size and boundaries of the chunk of text that you index and retrieve. It's the atomic piece of content that your search engine retrieves and returns when a user query comes in. It could be a page, a paragraph, or even a sentence.

The RAG Blueprint guides you through this choice because it has a direct impact on retrieval accuracy, context, and efficiency depending on the size you choose:

- **Too small:** Early approaches often chunked documents into very small pieces (e.g., a paragraph per chunk). While this maximized the chance of catching relevant text, it sacrificed context. You'd retrieve fragments that lacked the surrounding information needed for accurate ranking and answer generation.
- **Too large:** On the other hand, indexing entire books or very large PDFs as a single unit pushes all the ranking complexity inside a single document. Cross-document ranking becomes trivial, but intra-document relevance becomes nearly impossible to solve efficiently.

Best Practices from The RAG Blueprint

Aim for a unit size that balances context with retrievability.

For example:

- Larger paragraphs, or a few paragraphs combined, tend to work well for chunks.
- When in doubt, err toward larger units rather than smaller ones. Modern models are getting better at handling larger contexts.

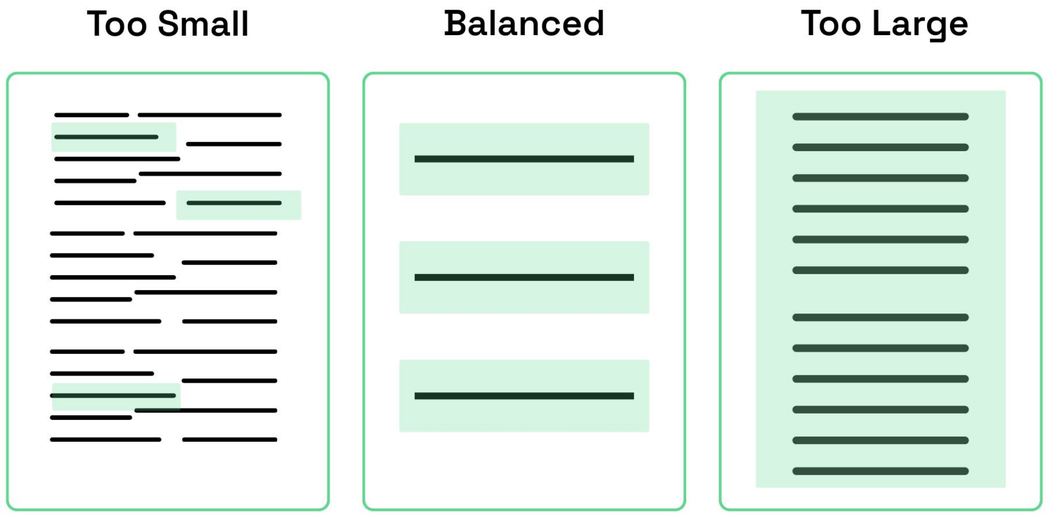

Figure 1: Example of unit sizes

Pro-Tip in Vespa

With Vespa's **chunk selection** mechanism, you can model your entire document as one while still chunking the content. This ensures only the most relevant parts of large documents are surfaced.

Modeling Document Signals

After selecting your document unit, the next step is to represent it in multiple ways. This ensures your system has the right signals to power accurate retrieval and ranking.

In practice, this means deciding how that piece of content is stored and enriched inside your retrieval system. Each unit should carry different layers of information, so the search engine has everything it needs to surface the best results quickly and accurately:

- **Embeddings (semantic signals):** Create one or more embeddings per document or section to capture semantic meaning and enable similarity-based retrieval.
- **Full Text (lexical signals):** Preserve the raw text for exact keyword matches, phrase queries, and lexical scoring.
- **Structured Metadata (contextual signals):** Add contextual attributes like category, author, or publish date. These not only refine ranking but also allow filtering, faceting, and drill-down exploration.

By combining semantic, lexical, and contextual signals, you give your retrieval system the depth and flexibility it needs. This multi-layered representation ensures that results are not just relevant, but also precise, explainable, and aligned with your business context. It lays the foundation for reliable ranking at scale.

Best Practices from The RAG Blueprint

Model your document units with embeddings, full text, and metadata together. This combination delivers completeness (no signal gaps), flexibility (semantic, lexical, and business-rule use cases), and performance.

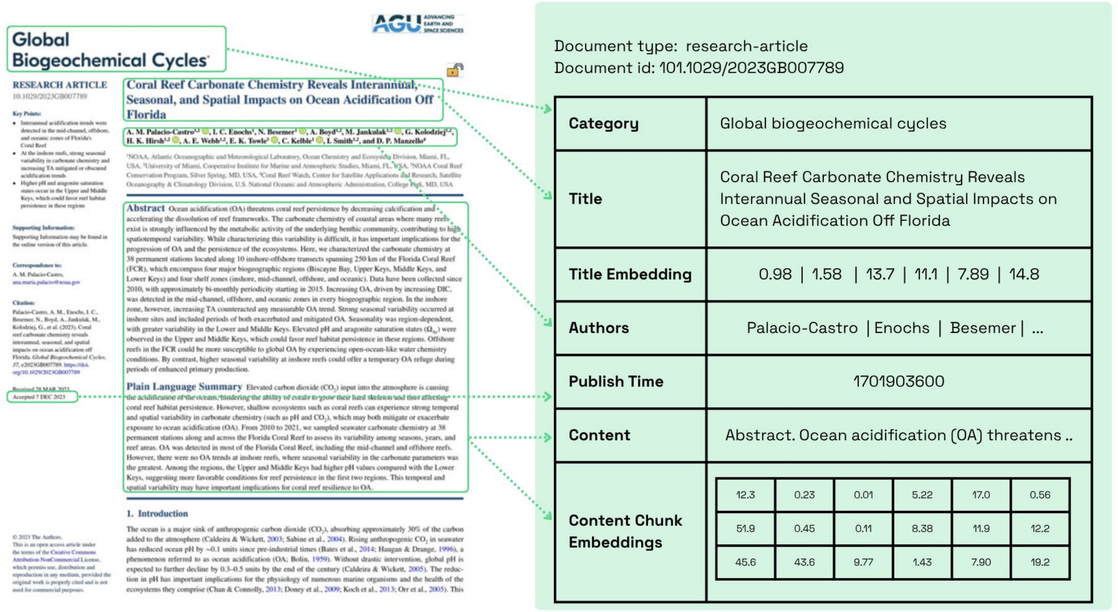

Figure 2: How Vespa stores document signals

Pro-Tip in Vespa

With Vespa, you can store all three: text, embeddings, and metadata, directly in the schema. Vespa then handles retrieval and ranking natively, so you don't need to manage separate embedding pipelines.

Defining a Retrieval Strategy

One of the most important design choices in a RAG system addressed by The RAG Blueprint is deciding how queries will retrieve content: lexical search, vector search, or a hybrid of both.

- **Lexical only:** Matches exact words and phrases, making it excellent for precision on IDs, names, or technical terms. But it fails when relevant passages use different wording or synonyms.
- **Vector only:** Uses embeddings to capture meaning, so it can retrieve semantically similar passages even if no keywords overlap. However, vectors often stumble on domain-specific jargon or product names, as these are often not in the training set.
- **Hybrid (recommended):** Combines both approaches, ensuring high recall from vectors and high precision from lexical matches. This balance is especially important for enterprise use cases where domain-specific language and structured attributes matter.

This decision determines whether your RAG system retrieves the right evidence for the LLM to reason over. Without the right balance, you risk omitting essential details or missing nuanced connections.

Best Practices from The RAG Blueprint

Always include lexical signals in ranking. Even when embeddings are strong, keyword matches remain essential for grounding responses.

Tune your balance. Use vectors to maximize recall, then rely on lexical features and ranking expressions to filter noise.

Adapt by domain. In highly specialized fields (life sciences, legal, e-commerce), hybrid is essential. In broad consumer domains, you may lean more on vectors, but don't drop lexical entirely.

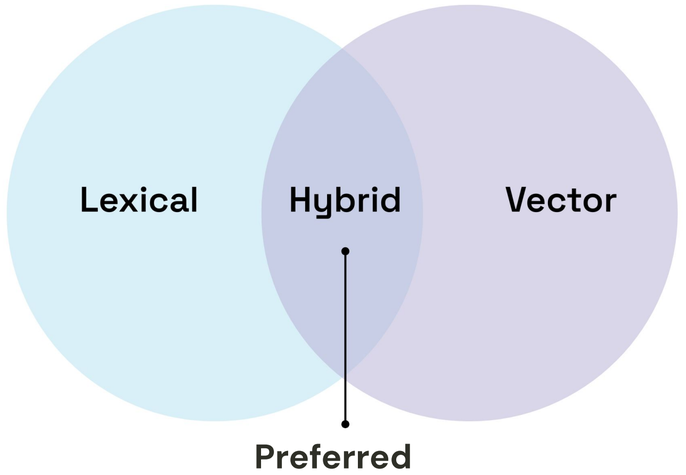

Figure 3: Hybrid Venn Diagram

Pro-Tip in Vespa

Vespa natively unifies hybrid retrieval in a single engine. No external re-rankers or duplicated indexes. Handle billions of documents and petabytes of data with Vespa's millisecond-latency hybrid search

Organizing Queries for Scale

As applications grow, retrieval and ranking needs diversify. Different use cases require different parameters, filters, and ranking models.

If you try to manage all of this with raw parameters in every request, complexity quickly spirals out of control. You end up duplicating logic, introducing inconsistencies, and making experimentation painful.

Query profiles solve this by acting as named, reusable configurations. Instead of hardcoding dozens of parameters for each request, you define a profile once and reference it wherever it's needed.

A query profile bundles together:

- Query parameters (how many results to fetch, what filters to apply)
- Ranking expressions (relevance scoring functions, learned models)
- Filters and constraints (time range, access control, business rules)

Query profiles allow your RAG application to grow in scale and sophistication without becoming unmanageable. They make retrieval strategies modular, maintainable, and adaptable so you can support dozens of use cases while still evolving your ranking models and business rules with minimal friction.

Best Practices from The RAG Blueprint

Define Profiles for Each Use Case: Examples include "semantic search," "hybrid search," or "recommendations."

Leverage Multi-Phase Ranking Inside Profiles: Each profile can specify first-phase retrieval, second-phase reranking, and global ranking separately.

Iterate Safely: You can test new ranking models by switching profiles in controlled rollouts, rather than changing client code.

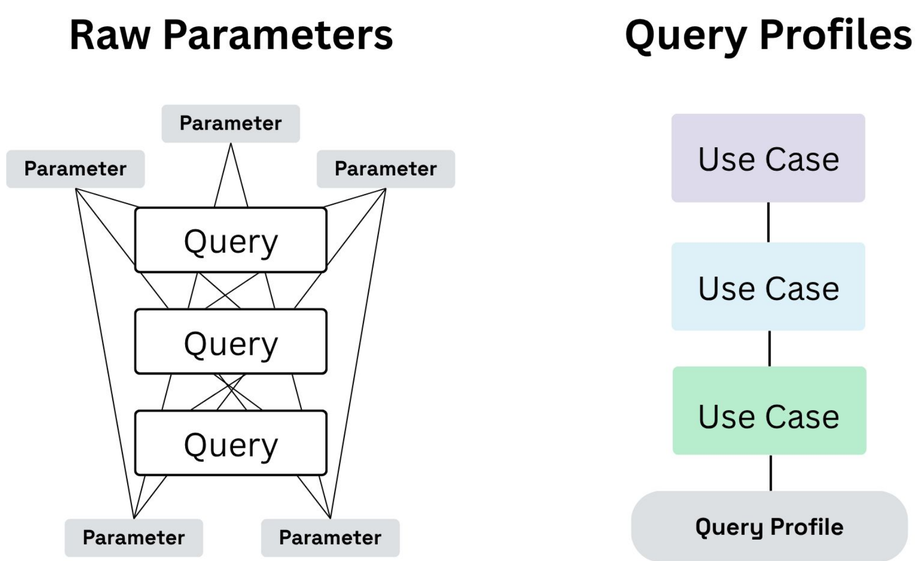

Figure 4: Raw parameters vs. query profiles

Pro-Tip in Vespa

In Vespa, query profiles are built into the engine. They can inherit from each other to reduce duplication and bundle parameters, ranking expressions, filters, and models into one reusable configuration. Unlike other platforms that need duplicated configs or custom middleware, Vespa handles it natively.

Optimizing Recall with Binary Vectors

When a query runs, the system's first job isn't to find the single best answer. It's to gather a candidate set of potentially relevant documents. This is recall: how effectively the system casts a wide enough net to ensure that the truly relevant items are included in the pool.

In RAG, recall is critical because an LLM can only generate answers from the context it receives. If relevant documents are missed in the first-stage recall, no amount of reranking or LLM reasoning can bring them back. Strong recall improves answer completeness and directly reduces hallucinations.

Optimizing recall means gathering good-enough coverage at low cost, then letting later phases (re-rankers, ranking models, and the LLM) refine for precision. The goal isn't perfect accuracy up front, but efficient breadth: high coverage without paying for expensive full-precision vector math across billions of documents.

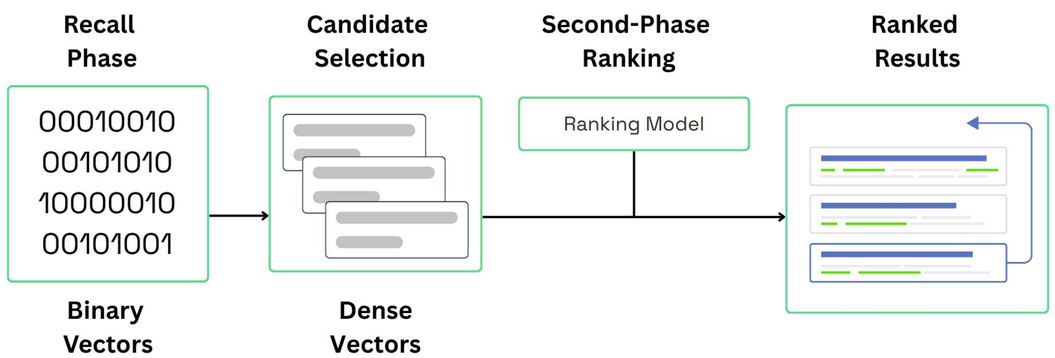

Figure 5: Recall with binary vectors

Best Practices from The RAG Blueprint

Use binary vectors for recall. They're fast, memory-light, and cheap, making them ideal for broad candidate gathering.

Re-rank with full precision. Apply dense vectors, hybrid lexical + semantic features, and business logic to the smaller candidate set for accuracy.

Balance recall depth and cost. Too shallow, and you miss context; too deep, and you waste compute. Vespa lets you tune this tradeoff directly in your schema and ranking expressions.

Pro-Tip in Vespa

Vespa lets you store both binary and dense embeddings in the same schema and use them in different ranking phases.

You can run cheap recall with binary vectors, then seamlessly pass candidates into a dense vector reranker or a hybrid ranking expression, all inside Vespa, no external systems needed.

Designing Fields: Titles, Bodies, & Embeddings

Field design is about structuring your content into meaningful parts so the retrieval system can treat them differently. It's about deciding what counts as a separate "field" (titles, bodies, metadata, and embeddings) in your schema and how those fields are weighted or modeled for search and ranking.

In domains where a title conveys key meaning (scientific papers, product listings, legal cases), field design strongly boosts retrieval quality.

While multiple fields improve control, they also increase complexity and resource usage. Done right, it improves relevance without inflating cost.

Best Practices from The RAG Blueprint

For textual fields: it's almost always worth splitting data into meaningful fields (e.g., title vs. body). Matches in titles should carry more weight than matches buried deep in the body.

For embeddings: the trade-off is trickier. Splitting embeddings by field (title embedding + body embedding) doubles cost and benefits are smaller compared to text fields. Use separate embeddings only if the field-level semantic differences are critical.

Pro-Tip in Vespa

Leverage Vespa's schema flexibility: Vespa lets you store both lexical fields and embeddings side by side, apply different weights, and combine them in ranking without external preprocessing.

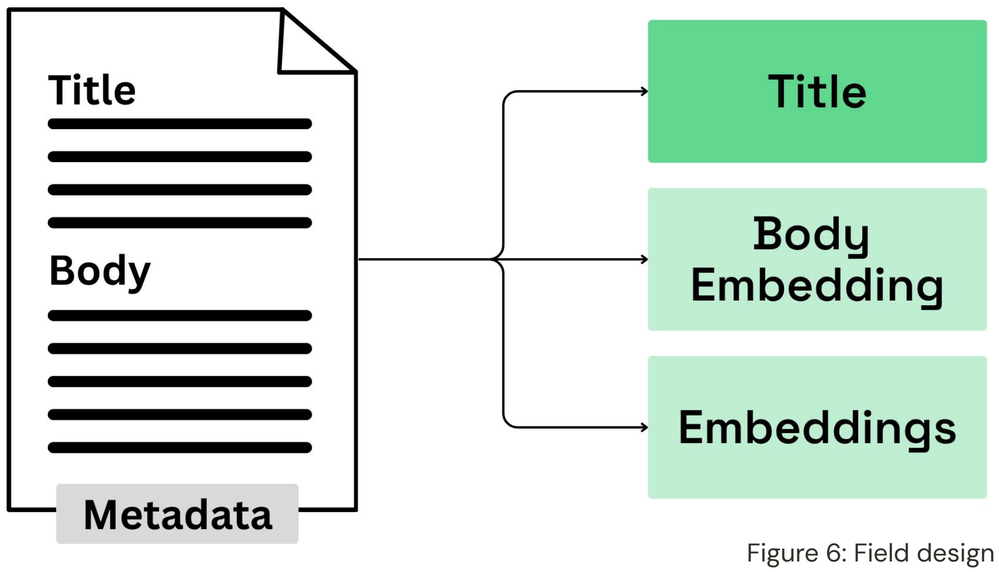

Figure 6: Field design

Combining Signals

Effective ranking requires more than just embeddings or keyword matches. It depends on combining multiple signals that reflect both semantic meaning and real-world context.

Metadata and additional signals play a critical role in ensuring that the most relevant, reliable, and useful results rise to the top. Attributes such as author, publish date, category, or region help the system distinguish between documents that might look similar in content but serve very different purposes. It also enables precise filtering and faceting, allowing users to drill down into exactly the subset of results they need.

Additional signals further refine ranking by capturing real-world usage and quality indicators. These might include user engagement, click-through rates, freshness, or popularity.

Signals help disambiguate when multiple results seem equally relevant in text or vector space, ensuring the system surfaces the items that are not only similar but also useful.

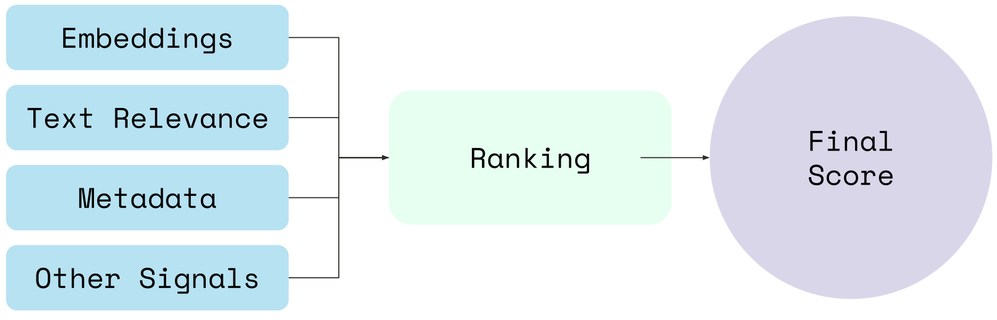

Figure 7: Ranking Signals

Best Practices from The RAG Blueprint

Write signals directly into your documents: Many systems use a separate "feature store" and only apply signals at the final re-ranking stage, but this restricts their influence to the top-k documents.

Model Metadata as First-Class Fields: Always include structured attributes like category, author, date, product type, or jurisdiction in your schema.

Boost by Field Importance: Weight fields differently based on how users interpret them. For example, a match in the title should carry more influence than one buried deep in the body, while publish date can be used to boost recency.

Use metadata to encode domain-specific priorities: In commerce, promote in-stock or higher-margin products; in life sciences, prioritize peer-reviewed research.

Include signals in your ranking formula: These real-world signals help disambiguate between equally relevant results and align ranking with user value.

Don't rely on metadata or signals in isolation: The best results come from blending embeddings, text relevance, and metadata features into a unified scoring model.

Pro-Tip in Vespa

In Vespa, all data types, including text, vectors, metadata, and behavioral signals, can be stored, retrieved, and ranked together inside the same engine.

Ranking with Machine Learning

Strong ranking depends on combining multiple signals into a scoring function that can separate the truly relevant results from the noise. These signals include vector similarity, lexical matching scores, metadata features, behavioral signals, and sometimes hand-crafted rules such as recency or authority boosts. On their own, each signal has limitations, but together, they provide a rich, multi-dimensional picture of relevance.

At the core of this process are ranking models: algorithms that learn or encode how to weight and combine different features.

Ranking is not static and different use cases require different trade-offs. For this reason, teams often deploy multiple ranking models simultaneously, switching between them based on use case, audience, or A/B testing requirements.

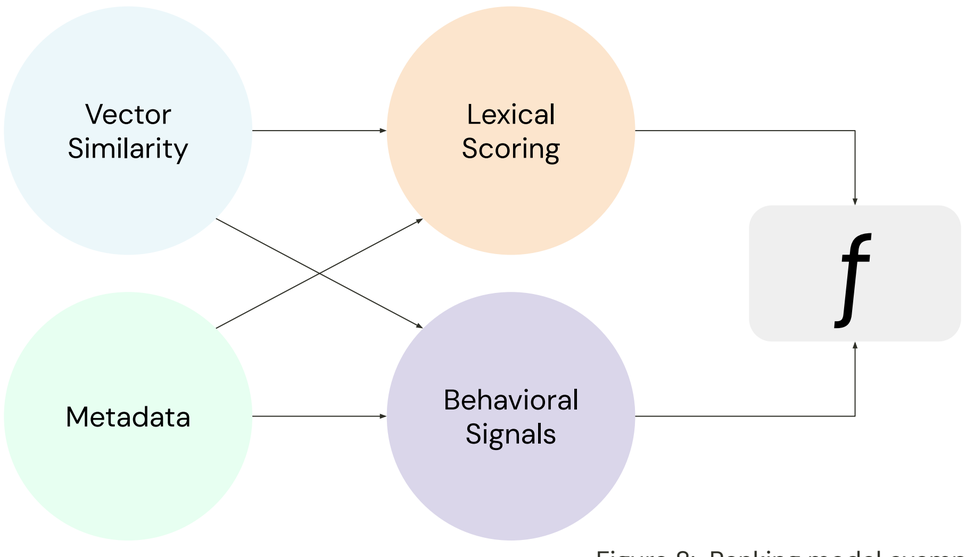

Figure 8: Ranking model example

Best Practices from The RAG Blueprint

Don't rely on vectors alone. Blend multiple features: combine lexical, semantic, metadata, and behavioral signals.

A reliable default is Gradient Boosted Decision Trees (GBDTs). They provide a fast, reliable baseline that balances accuracy and cost.

Layer your ranking. Use cheap models for recall, then apply heavier models (transformers, neural re-rankers) in later stages where precision matters most.

Run multiple models in parallel. Different use cases benefit from different ranking strategies. Keep them side by side for flexibility.

Leverage ranking profiles. In Vespa, profiles let you define, tune, and switch between models natively, simplifying experimentation and deployment.

Pro-Tip in Vespa

Vespa's ranking framework is fully extensible. You can embed GBDTs, neural models, hand-crafted rules, and hybrid signals into the same query pipeline so you can iterate on relevance continuously, without bottlenecks from external systems.

In practice, you'll often deploy multiple models simultaneously, for different use cases or to A/B test generations. Vespa lets you switch between ranking profiles seamlessly at query time.

Ranking in Multiple Phases

In modern retrieval systems, true ranking is machine-learned: models that combine multiple signals to generate a rich, context-aware measure of relevance. This is what makes results feel precise, personalized, and aligned with business needs.

But there's a catch: applying these ranking models directly to billions of documents simply doesn't scale. Running full inference across every possible candidate would blow through compute budgets and create unacceptable latency for end users. Even with efficient hardware, the cost of evaluating deep ranking models at web or enterprise scale is prohibitive.

This tension creates the classic ranking bottleneck: the most accurate models are also the most computationally expensive. Left unchecked, they slow down systems, inflate costs, and limit the size of candidate pools you can reasonably consider.

That's why retrieval architectures should rely on multi-phase ranking: using lightweight functions to narrow down the candidate set, and only then applying more expensive, machine-learned models on the top-k results.

By layering ranking in this way, systems achieve the best of both worlds: scalable retrieval across billions of items, combined with high-precision results where it matters most.

Best Practices from The RAG Blueprint

Use a multi-phase ranking strategy:

- First phase: Use a lightweight function (lexical score, approximate vector similarity, or both) to reduce the candidate pool.
- Second phase: Apply machine-learned ranking on the top-k results (e.g., 100–1000).
- Chunk selection: For large documents, add a third layer: rank chunks within retrieved documents to pass only the most relevant spans to the LLM.

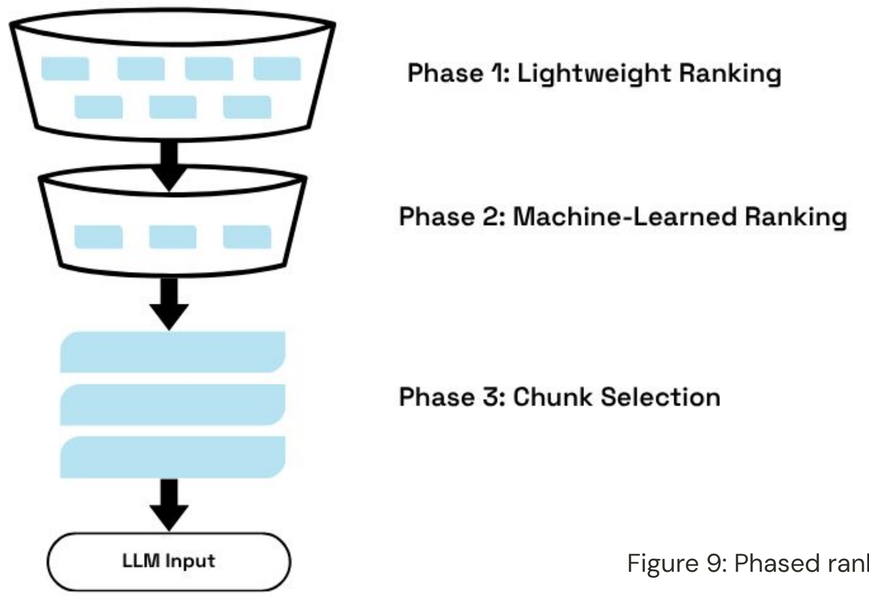

Figure 9: Phased ranking

Pro-Tip in Vespa

Vespa integrates all three phases natively. Lightweight retrieval, machine-learned ranking, and chunk-level scoring can be defined in a single query pipeline and executed at scale

Conclusion

Building scalable, high-quality RAG applications requires more than just “vector search.” It’s about:

- Choosing the right retrieval unit
- Combining lexical and vector signals
- Using binary vectors to control costs
- Organizing queries for maintainability
- Embedding metadata and signals directly in documents
- Applying multiphase ranking with chunk selection

Vespa provides a unified platform for all of these capabilities, retrieval, ranking, chunking, and serving models at scale, without the brittle handoffs between external systems.

If you’re moving beyond prototypes and need RAG that actually scales, these practices can help you build applications that are both cost-efficient and accurate.

Go from theory to production with The RAG Blueprint.

Customer Case Studies

Perplexity

Perplexity, one of the fastest-growing AI-powered answer engines, relies on Vespa.ai to power retrieval at massive scale.

Vespa serves as the backbone of Perplexity's RAG pipeline, handling billions of documents and over 100 million weekly queries with low latency.

By combining vector similarity, lexical matching, metadata, and custom ranking functions in one platform, Vespa ensures that Perplexity retrieves the most accurate, context-rich passages for its large language models. This allows Perplexity to deliver fast, precise, and trustworthy answers to millions of users worldwide—all without compromising on scale, speed, or cost efficiency.

Learn More

Onyx

Onyx.app, the open-source enterprise search and knowledge assistant, uses Vespa.ai to deliver scalable, cost-efficient retrieval for corporate data.

Onyx leverages Vespa's hybrid retrieval, combining vectors, text, and structured metadata, along with Vespa Cloud's autoscaling and resource suggestions to keep operations efficient.

The result is a robust enterprise RAG platform that can index millions of internal documents, respect access controls, and surface the most relevant information instantly for business users.

Learn More

Bigdata.com

BigData.com, a leading market intelligence platform, relies on Vespa.ai to deliver real-time search and insights across vast collections of financial and business data.

Vespa serves as the foundation of their retrieval layer, combining vector similarity, lexical matching, and structured filters to surface the most relevant results instantly.

By running retrieval and ranking in-cluster at scale, Vespa ensures that BigData.com provides analysts and decision-makers with fast, accurate, and cost-efficient access to billions of data points.

Learn More

About Vespa.ai

Vespa.ai is a platform for building and running real-time AI-driven applications for search, recommendation, personalization, and RAG. It enables enterprise-wide AI deployment by efficiently managing data, inference, and logic, handling large data volumes and over 100K queries per second. Vespa supports precise hybrid search across vectors, text, and structured metadata. Available as both a managed service and open source, it's trusted by organizations like Spotify, Vinted, Wix, and Yahoo. The platform offers robust APIs, SDKs for integration, comprehensive monitoring metrics, and customizable features for optimized performance.

Interested to learn more? We have many different resources and information available through our social platforms

GitHub Twitter LinkedIn YouTube

Vespa.ai