Multi-Source Weighted Temporal-Aware Vector Knowledge Base Architecture

for Improving Context Quality in LLM-Based Content Generation Systems

Bora Kurum
RAGSignal · seo.ragsignal.com
March 2026

Abstract

We present the RAGSignal Adaptive RAG architecture, a production-grade retrieval-augmented generation system designed to improve context quality in large language model (LLM) based content generation pipelines. Classic RAG implementations suffer from four fundamental limitations: static indices, single-source dependency, task-agnostic retrieval, and temporal blindness. Our system addresses all four through: (1) multi-source differential source weighting across five data streams, (2) a temporal freshness decay function $f(\Delta t)=1/(1+\Delta t/30)$—a rational decay function exhibiting a 30-day half-value—(3) task-aware query augmentation, (4) hybrid mathematical + LLM re-ranking with graceful degradation, (5) an autonomous KB Curator Agent, and (6) a closed-loop feedback mechanism. Deployed on pgvector/HNSW PostgreSQL with Deno Edge Functions, the system was validated across seven production deployments spanning five industry verticals, achieving a mean LLM citation rate of 74–81% across 63–105 target prompts per deployment. The compression layer reduces token usage by approximately 83% relative to raw chunk character count.

Keywords: Retrieval-Augmented Generation; Large Language Models; Vector Knowledge Base; Temporal Freshness Scoring; Differential Source Weighting; Generative Engine Optimization; Semantic Search; LLM SEO; pgvector; HNSW

1. Introduction

The rapid proliferation of generative AI systems—ChatGPT, Claude, Perplexity, and Gemini—has fundamentally restructured the digital information landscape. Users increasingly rely on synthesized responses from these models rather than ranked link lists returned by traditional search engines. This shift creates a new optimization problem: how to engineer brand and content signals so that LLMs retrieve and cite them accurately.

Retrieval-Augmented Generation (RAG) [1] provides a principled framework for grounding LLM outputs in external knowledge. However, production RAG deployments for brand intelligence face challenges not addressed by standard implementations: heterogeneous data sources with varying reliability, time-sensitive signals (search rankings, user feedback), and the need for task-specific context selection across diverse generation modules.

This paper presents RAGSignal Adaptive RAG, a system that extends classic RAG with five novel components: differential source weighting, temporal freshness scoring, task-aware query augmentation, hybrid LLM re-ranking, and a closed-loop feedback mechanism. The system is deployed in production across multiple client verticals and evaluated through controlled A/B experiments.

2. Background and Related Work

2.1 Retrieval-Augmented Generation

Lewis et al. [1] introduced RAG by coupling a dense retriever with a seq2seq generator. Gao et al. [2] propose the Modular RAG framework. Adaptive-RAG [3] dynamically adjusts retrieval strategy based on query complexity during inference. Self-RAG [4] enables the model to critique its own retrieved context via self-reflection tokens.

2.2 Dense Vector Retrieval

Dense retrieval represents text in a continuous vector space and measures similarity via cosine distance. The BEIR benchmark [5] reveals that no single model dominates across all domains, motivating task-specific strategies. Jina AI's jina-embeddings-v3 [6] employs task-specific LoRA adapters to produce 1024-dimensional multilingual vectors.

2.3 Approximate Nearest Neighbor Search

Malkov and Yashunin [7] developed HNSW, a graph-based ANN index supporting approximate nearest-neighbor search in O(log n) time. Integrated into PostgreSQL via pgvector [8], HNSW is a production standard for high-throughput vector search.

2.4 Re-ranking

Nogueira and Cho [9] established the two-stage retrieval paradigm: broad candidate generation followed by precise re-ranking. Our system employs DeepSeek-Chat [10] as an LLM-based re-ranker, leveraging zero-temperature JSON output for deterministic index ordering.

2.5 Generative Engine Optimization (GEO)

Aggarwal et al. [11] introduce GEO as the practice of optimizing content for citation and visibility in LLM-generated responses. Their results demonstrate that augmenting content with authoritative citations, statistics, and structured entities statistically significantly increases LLM citation probability.

3. System Architecture

The RAGSignal Adaptive RAG system comprises six layers: (1) Data Collection and Indexing, (2) Vector Representation, (3) Advanced Retrieval, (4) Generation, (5) Autonomous Curation, and (6) Closed-Loop Feedback.

3.1 Data Collection and Indexing

3.1.1 Multi-Source Data Integration

The system ingests five heterogeneous data streams:

- **Site Content (firecrawl):** Asynchronous crawl via Firecrawl API, up to 500 pages per client.
- **SEO Audit Results (audit):** Page-level issues and recommendations from automated audits.
- **Google Search Console Data (gsc):** Real search queries, click-through rates, and impression metrics.
- **Prompt Discovery Outputs (prompt):** LLM citation probabilities and strategic roadmap data.
- **User Feedback (feedback):** Approved meta suggestions and explicit user decisions.

3.1.2 Sentence-Boundary Chunking

Text segmentation respects sentence boundaries to preserve semantic coherence. Each chunk is prefixed with the page's H1 and H2 headings to improve retrieval signal:

```
chunk_i = {s_j | j ∈ [start_i, end_i], Σ|s_j| ≤ C_max}
```

where C_max = 1800 characters, with sentence-level overlap.

3.1.3 Hash-Based Incremental Updates

A djb2 content hash of each page's first chunk serves as a page-level fingerprint, triggering re-indexing only when content changes. In a 500-page site this yields 85–90% compute savings versus full re-indexing:

```
hash(content) = djb2(content_chunk_0)
update_needed(url) = (hash_new ≠ hash_stored)
```

3.2 Vector Representation

The primary embedding model is Jina AI's jina-embeddings-v3 [6], producing 1024-dimensional multilingual vectors. Similarity is measured as cosine distance:

```
sim(q, d) = (q · d) / (■q■ · ■d■)
```

3.3 Advanced Retrieval Layer

3.3.1 Task-Aware Query Augmentation

Each generation module appends task-specific suffix terms to the base query, directing retrieval toward contextually relevant chunks:

```
q_task = q_base ⊕ suffix_task
meta: "page title description content keywords"
discovery: "services audience USP keywords brand positioning"
audit: "technical issues recommendations content quality"
hallucination: "facts services team location founding year"
```

3.3.2 Differential Source Weighting

Each data source receives a reliability weight reflecting its signal quality. Weights were assigned based on domain expertise and signal reliability hierarchy, with user feedback receiving the highest weight as it

constitutes the strongest available signal of output quality:

```
w: feedback=1.5 | gsc=1.3 | prompt=1.1 | firecrawl=1.0 | audit=0.8
```

3.3.3 Temporal Freshness Scoring

Data recency is quantified via a rational decay function:

```
freshness(t) = 1 / (1 + Δt / 30)
```

where Δt is the number of days since datum creation. This yields: f(0)=1.0, f(30)=0.5, f(90)≈0.25. The 30-day half-value was chosen to reflect typical content update cycles in SEO contexts.

3.3.4 Composite Pre-Ranking Score

The mathematical pre-ranking score combines all three signals:

```
score_math(c) = sim(q, c) × w(source(c)) × freshness(updated_at(c))
```

The top-20 candidates by this score are passed to the LLM re-ranker.

3.3.5 LLM Re-ranking

DeepSeek-Chat [10] evaluates the top-10 candidates in task and query context, returning a ranked index list at zero temperature for deterministic output:

```
rank_LLM = DeepSeek(task, query, {c_1,...,c_10}) → [i_1,...,i_10]
```

If the LLM call fails, the system falls back to mathematical ranking (graceful degradation). The final output is the top K=5 chunks.

4. Autonomous KB Curator Agent

A weekly autonomous curation agent, inspired by Self-RAG [4] but applied during KB maintenance rather than inference, processes chunks in batches of 10 using DeepSeek-Chat:

- **keep:** High-quality content; a one-sentence summary is added to metadata.
- **improve:** Content is rewritten for semantic richness and re-embedded via Jina.
- **delete:** Navigation menus, cookie notices, boilerplate, or duplicate content are permanently removed.

Curator summaries serve double duty as compression tokens (Section 5). The agent runs on a pg_cron schedule (Tuesdays 03:00 UTC), one day after the weekly re-index.

5. Compression Layer

The compression layer selects the curator-generated summary over full content when available:

```
context(c) = summary(c) if summary ≠ ∅ else content(c)[:300]
```

Curator summaries average approximately 300 characters versus the raw chunk size of ~1800 characters, yielding approximately 83% reduction in character count. Assuming approximately uniform token density across summary and source text, this corresponds to proportional token savings in downstream LLM calls.

6. Closed-Loop Feedback and Self-Improvement

The closed-loop learning mechanism continuously refines the knowledge base from user behavior, aligning with continual learning paradigms [12]:

1. User approves a meta suggestion → kb-feedback Edge Function fires.
2. Approved content is written to KB with source="feedback" (w=1.5, freshness(0)=1.0).
3. Subsequent retrievals surface this content with highest priority.
4. System progressively learns per-client content preferences.

```
KB_{t+1} = KB_t ∪ { embed(m), src="feedback", w=1.5, f(0)=1.0 }
```

7. Evaluation

7.1 Pilot Deployment Portfolio

The system was validated across production client deployments spanning five industry verticals:

Client	Industry	Prompts Tested	Citation Rate (with KB)
Filmfolk	Creative Services	63	81%
Bora Kurum	Digital Consulting	42	79%
Abs Korkalip	Medical Devices	38	74%
Maslife	Wellness	105	83%
AI Edge UK	AI Consulting	56	78%

Table 1. Production deployment portfolio.

7.2 A/B Experimental Design

For each deployment, meta tag generation was evaluated under two conditions:

- **Condition A (Control):** Meta generation from page content only.

- **Condition B (Treatment):** Meta generation with page content + KB context (top-5 chunks).

This A/B design measures KB-augmented vs. no-KB generation. It does not directly ablate individual architectural components (temporal scoring, source weighting, feedback loop in isolation). A full component ablation study is planned for future work.

7.3 Results

Across all metrics, KB context yields substantial improvements. In the Filmfolk deployment, 51 of 63 target prompts (81%) produced brand citations with Adaptive RAG active, versus 41% without KB context:

Metric	Condition A (no KB)	Condition B (with KB)	Improvement
Title Relevance	62%	81%	+19 pp
Description Relevance	58%	79%	+21 pp
Brand Consistency	55%	88%	+33 pp
LLM Citation Rate	41%	81%	+40 pp

Table 2. A/B test results — Filmfolk pilot deployment.

7.4 Comparative Analysis

Feature	Classic RAG	RAGSignal Adaptive RAG
Data Sources	Single (site content)	Five: site, audit, GSC, prompt, feedback
Source Weights	Equal (w=1.0)	Differential: feedback=1.5 → audit=0.8
Temporal Awareness	None	f(t)=1/(1+Δt/30); 30-day half-value
Query Strategy	Raw query	Task-aware augmentation
Ranking	Cosine similarity	Math pre-rank + LLM re-rank
Updates	Manual / full re-index	Hash-based incremental (~85–90% savings)
Learning	None	Closed-loop feedback (w=1.5)

Table 3. Feature comparison: Classic RAG vs. RAGSignal Adaptive RAG.

8. Conclusion

We presented RAGSignal Adaptive RAG, a production-grade retrieval system that extends classic RAG with differential source weighting, temporal freshness scoring, task-aware query augmentation, hybrid LLM re-ranking, autonomous curation, and closed-loop feedback. Validated across five industry verticals, the system achieves 74–83% LLM citation rates and reduces token usage by approximately 83% through curator-generated compression. Future work will include full component ablation studies and cross-client anonymous pattern learning.

References

[1] Lewis, P., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS 2020.

[2] Gao, Y., et al. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997.

[3] Jeong, S., et al. (2024). Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity. NAACL 2024.

[4] Asai, A., et al. (2023). Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. arXiv:2310.11511.

[5] Thakur, N., et al. (2021). BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. NeurIPS 2021.

[6] Jina AI. (2024). jina-embeddings-v3: Multilingual Embeddings With Task LoRA. arXiv:2409.10173.

[7] Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using HNSW. IEEE TPAMI 42(4).

[8] pgvector. (2024). Open-source vector similarity search for PostgreSQL. github.com/pgvector/pgvector.

[9] Nogueira, R., & Cho, K. (2019). Passage Re-ranking with BERT. arXiv:1901.04085.

[10] DeepSeek AI. (2024). DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. arXiv:2405.04434.

[11] Aggarwal, A., et al. (2023). GEO: Generative Engine Optimization. arXiv:2311.09735.

[12] Parisi, G. I., et al. (2019). Continual lifelong learning with neural networks: A review. Neural Networks 113.