Tuning RAG Systems for Production

A Decision Framework for Eight Optimization Options and One Production-UX Pattern

Evaluated against seven core metrics — with empirical data from

a production RAG deployment over the U.S. Code of Federal Regulations

Brad Hinkel

White Paper · April 2026

Executive Summary

Retrieval-augmented generation (RAG) has a large design space. Chunking strategy, embedding model, retrieval mode, top-k, reranking, query transformation, generation strategy, and model selection are all tunable — and their interactions are not additive. Teams routinely adopt "best practices" from blog posts, find that their own metrics don't move, and lose weeks to tuning that a single well-designed experiment would have settled.

This white paper distills empirical findings from a production RAG deployment over 85,351 sections of U.S. federal regulations (CFR Titles 7, 21, and 42). Eleven controlled experiments were run against a fixed 60-question evaluation set, each varying a single component of the pipeline. We report results for eight optimization options that most teams consider early in a RAG project, plus a production-UX pattern that becomes critical in high-stakes domains:

1. **Sequential Generation** — multi-call instead of single-call generation
2. **Top-k Tuning** — how many retrieved chunks to pass to the generator
3. **LLM Model Selection** — Sonnet vs. Haiku (larger vs. smaller frontier model)
4. **Hybrid Retrieval** — BM25 + vector via reciprocal rank fusion
5. **Query Rewriting** — LLM expands the user query before embedding
6. **HyDE Query Expansion** — LLM generates a hypothetical passage before embedding
7. **Hierarchical Retrieval** — retrieve fine-grained chunks, then assemble coarse-grained context for generation
8. **Fine-Tuned Embeddings** — voyage-law-2, a legal-domain-specialized model
9. **Inference-Time Confidence Signal** (UX pattern) — a per-query reliability indicator surfaced to the user; doesn't improve eval metrics but improves user trust and safety

Headline findings:

- **Only two of eight optimization options produced meaningful, unambiguous wins.** Sequential generation and top-k tuning carried the weight; the rest were neutral, harmful, or confounded.
- **Four options were directly harmful or net-neutral in this domain.** Query rewriting degraded MRR by 21–25%. HyDE degraded MRR marginally. BM25 hybrid moved MRR by +6% with no downstream benefit. Hierarchical retrieval collapsed generation metrics in every variant tested.
- **One "obvious" win was not.** A legal-domain-specialized embedding model (voyage-law-2) underperformed a general-purpose model — because it was paired with a chunking strategy that fragmented regulatory meaning. Domain-specialized components cannot compensate for structural mismatch upstream of them.

- **Upgrading to a larger LLM was not worth the cost.** Sonnet over Haiku produced a faithfulness gain within measurement noise at 3.6× the latency.
- **The confidence signal is the one addition that did not move the eval metrics — and is still worth shipping.** Production-UX patterns that surface per-query reliability to the user are under-invested in most RAG systems. In high-stakes domains — legal, medical, financial — they are arguably more load-bearing than another point of faithfulness on an aggregate score.

The bottom line for practitioners: start with the boring options (honor your document's structure, tune top-k, consider sequential generation), measure on your own corpus, treat every optimization as falsifiable until your own evaluation harness says otherwise — and don't forget that what the user actually sees at inference time is one answer, not an average score.

Context and Methodology

The experiments underlying this paper were run during the development of a public-facing regulatory-query product. The domain — federal regulations — is hallucination-sensitive: incorrect paraphrases of statutory language have legal consequences. That raises the bar for every optimization: a technique is worth adopting only if it improves grounding, not just perceived fluency.

Baseline Configuration

Unless otherwise stated, all experiments varied one component at a time against this baseline:

Component	Baseline Value
Corpus	CFR Titles 7, 21, 42 — 85,351 section-level chunks
Chunking	Section boundary (DIV8 in the CFR's own XML hierarchy)
Embedding model	OpenAI text-embedding-3-small (1536 dims)
Vector store	PostgreSQL 16 + pgvector, cosine similarity
Retrieval	Pure vector, top_k = 10
Generation	Claude Haiku 4.5, sequential two-call strategy
Evaluation set	60 regulatory questions with ground-truth CFR references

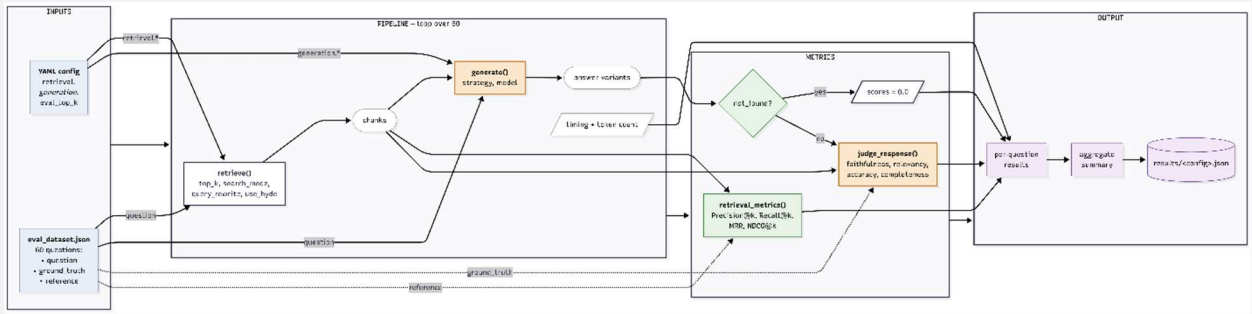
The Seven Core Metrics

Each experiment was scored against seven metrics covering retrieval quality, generation quality, and operational cost. Treating these as a single composite hides the real tradeoffs — an optimization can win on one axis while losing on another.

Metric	What it measures	Why it matters
Precision@k	Fraction of top-k retrieved chunks that are relevant to the query	Noise in the context window — every irrelevant chunk is tokens the generator has to filter out
Recall@k	Fraction of known-relevant chunks that appear in top-k	Coverage — are we giving the generator enough to answer the full question
MRR	Mean reciprocal rank of the first relevant result	Rewards finding the answer early; key signal when generator attention decays with position
Faithfulness	Does the generated answer stay grounded in the retrieved context (LLM-judge)	Direct proxy for hallucination risk in a grounded-answer product
Answer Relevancy	Does the generated answer actually respond to the user's question (LLM-judge)	Catches off-topic drift even when grounding is clean
Tokens / Query	Input + output tokens per user query	Cost line item and context-window utilization — both matter
E2E Latency	Wall-clock time from user submit to final answer	User-perceived speed; a 20s answer is a different product than a 5s answer

A note on the evaluation harness itself: before trusting any experiment result, audit the metrics code. In this project, three subtle bugs in the retrieval evaluator had been under-reporting MRR by roughly 2.5× for weeks. Several optimizations had been tuned against an artificial ceiling that did not exist. This is the single highest-leverage discipline in RAG tuning work: construct cases where the expected answer is obvious, confirm the metrics reward them, and don't accept a tuning result until you've adversarially tried to break the metric that scored it.

Evaluation Harness Flow



At a Glance: Eight Options × Seven Metrics, Plus a UX Pattern

The full experiment matrix in a single table. Directional symbols indicate the sign and rough magnitude of each metric change versus the baseline configuration. Read across a row to see an option's full cost/benefit shape; read down a column to see which options move a given metric. The Confidence Signal row is shown separately because it is an inference-time UX pattern, not an optimization — it does not move test-time metrics by design.

Legend: ▲▲ large gain · ▲ modest gain · — negligible / within noise · ▼ modest loss · ▼▼ large loss · n/t = not tested

Option	P@k	R@k	MRR	Faith.	Rel.	Tokens	Latency	Verdict
Sequential Generation	—	—	—	▲	▲	▲	▼	ADOPT
Top-k = 10 (vs. 6)	▲	▲▲	—	▲	▲	▼	—	ADOPT
Sonnet over Haiku	—	—	—	—	—	▼	▼▼	SKIP
Hybrid (BM25 + vector)	—	—	▲	—	—	—	—	NEUTRAL
Query Rewriting	▼	▼	▼▼	▼	▼	▼	▼	AVOID
HyDE Expansion	—	—	▼	—	—	▼	▼	SKIP
Hierarchical Retrieval	—	▲	—	▼▼	▼▼	▼▼	▼	AVOID
voyage-law-2 (fine-tuned)	▼	▲	▼	▼▼	▼	▲	—	CAUTIOUS
Confidence Signal (UX)	—	—	—	—	—	—	—	ADOPT (UX)

The shape of this matrix is the key takeaway: most of the sophistication on offer in the RAG literature didn't help in this domain, and several options hurt. The "boring" options — let the document's structure guide chunking, tune top-k, separate plain-English from legal-language generation into two calls — did the heavy lifting. Teams that invest in evaluation infrastructure early discover this quickly; teams that don't tend to adopt sophistication on faith.

The Confidence Signal row is the other important takeaway. It shows all neutral marks on test-time metrics — and an ADOPT verdict. That is not a contradiction; it is a reminder that the eval matrix is not the product. In a legal-grade RAG application, what matters to a user is not the average faithfulness

score across the whole eval set; it is whether they can tell, for their one query right now, whether to act on the answer. The confidence signal is the primitive that makes that transparency possible, and its cost to run is negligible.

Option 1: Sequential Generation

What It Is

Instead of emitting all generation outputs from a single LLM call (typically as structured JSON), the pipeline makes multiple sequential LLM calls, each producing one output and optionally conditioning on the prior output. In this project the legal-language call was conditioned on both the retrieved context and the plain-English summary from call 1.

Why You Might Try It

Structured-JSON generation forces the model to juggle several output styles in one pass. For multi-format products where different outputs have different style constraints (e.g., plain-English summary vs. legal-register synthesis with verbatim quotes), per-call specialization may improve each output independently.

Empirical Results

Metric	Single-Call	Sequential (2-call)	Δ
Faithfulness	Baseline	+4.5 points	gain
Legal accuracy	Baseline	+6.7 points	gain
Citation accuracy	Baseline	+12.9 points	large gain
E2E latency	2.5 s	4.9 s	~2×
Tokens / query	Baseline	+modest	noise-level

When to Use It

- When the product emits multiple distinctly-styled outputs from one retrieval (summaries, quotes, citations)
- When faithfulness is product-critical and users will accept ~5-second responses
- When you can reuse the retrieved context across calls (pay the retrieval cost once)

When to Skip

- Conversational products where sub-second response matters more than last-mile grounding
- Single-output products where JSON-mode already produces clean results

VERDICT: ADOPT when output quality matters more than the extra 2–3 seconds of latency.

Option 2: Top-k Tuning

What It Is

The number of retrieved chunks passed to the generator as context. Larger k means more context (higher recall, more tokens); smaller k means focused context (higher precision risk, fewer tokens).

Why You Might Try It

Top-k is the cheapest tuning knob in the RAG stack — no re-embedding, no model swap, no prompt changes. It's also under-tuned in practice: most teams pick a value once and never revisit it. In domains where answers span multiple source sections (definitions + rule + exceptions), under-sampling the retrieval set starves the generator of the grounding it needs.

Empirical Results (k=6 → k=10)

Metric	k = 6	k = 10	Δ
Recall@k	0.683	0.833	+22 pp
NDCG@k	Baseline	+10%	gain
MRR	Baseline	~flat	no cost
Faithfulness	Baseline	improved	gain
Legal accuracy	Baseline	improved	gain
Tokens / query	Baseline	+13%	modest cost
E2E latency	Baseline	~flat	no cost

Why it worked: regulatory questions typically touch multiple sections (a labeling-requirements query simultaneously needs § 205.300, § 205.301, and § 205.303). A top-6 retrieval set was consistently missing 1–2 of those sections. At top-10, coverage was clean without blowing out the context window.

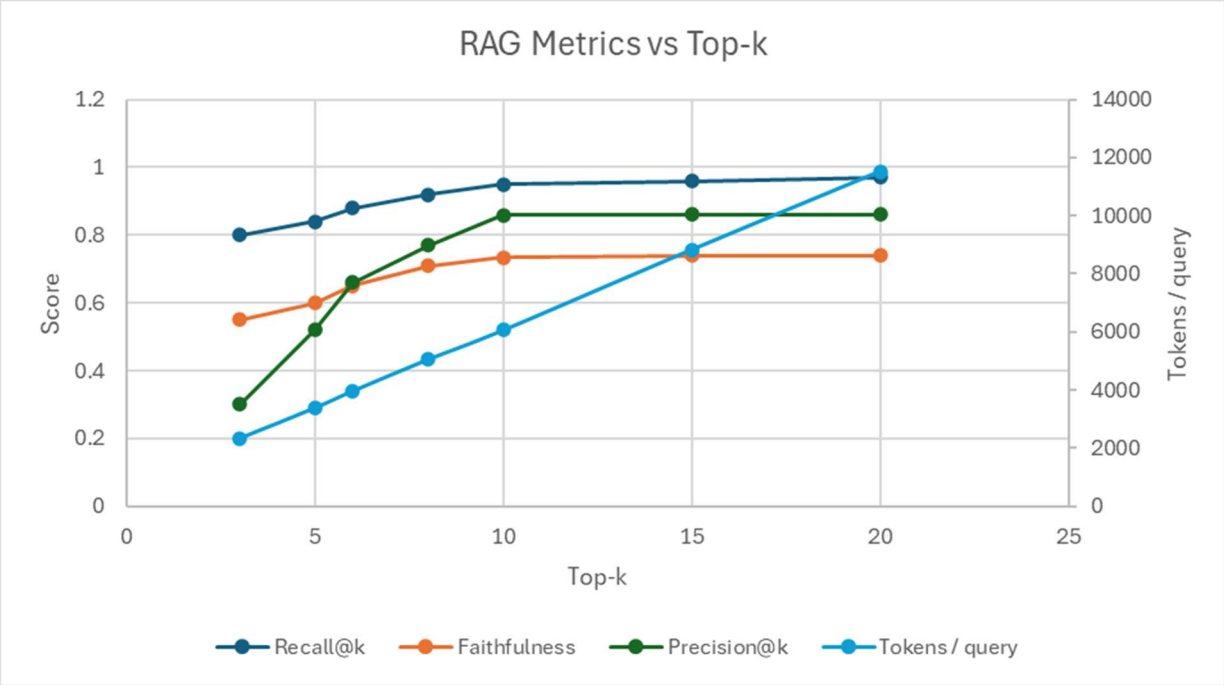
When to Use It

- Always — it's the first thing to tune on any RAG system
- Especially when answers plausibly span multiple source documents or sections
- Sweep k over {3, 5, 8, 10, 15, 20} early in the project; lock in the winner; re-sweep when the corpus or generator changes

When to Skip

- Never skip tuning k. The question is which value, not whether to look.

VERDICT: ADOPT — highest-ROI tuning knob in the stack.



The primary metrics increased until Top-k = 10. The Cost (Tokens/query) increased continuously. Top-k = 10 is the clear winner.

Option 3: LLM Model Selection (Sonnet vs. Haiku)

What It Is

Substituting a larger, more expensive frontier model (Claude Sonnet 4.5) for a smaller, cheaper, faster one (Claude Haiku 4.5) at the generation step.

Why You Might Try It

Larger models tend to hallucinate less, synthesize better across long contexts, and produce more stylistically consistent output. The assumption is that spending more on the generator should buy better-grounded answers — especially in hallucination-sensitive domains.

Empirical Results

Metric	Haiku 4.5	Sonnet 4.5	Δ
Faithfulness	Baseline	+0.9 points	within noise
Legal accuracy	Baseline	~flat	within noise
Citation accuracy	Baseline	~flat	within noise
Tokens / query	Baseline	+15%	modest cost
E2E latency	5.6 s	20.1 s	3.6×

Why it didn't help: once the retrieval context is clean and coherent (via section-level chunking and top-k=10), the generation task is essentially "synthesize what's already in the prompt." That's a task the smaller model can handle. The ceiling on faithfulness is being set by retrieval quality, not by model capacity.

When to Use It

- When the baseline model is measurably struggling with synthesis (e.g., logical contradictions, misattributed quotes)
- For long-context synthesis beyond the smaller model's effective context window
- As a last resort after chunking, top-k, and prompt engineering have been exhausted

When to Skip

- As a first optimization — it's expensive, slow, and often doesn't move faithfulness when retrieval is the bottleneck

VERDICT: SKIP as a default optimization. Start with the cheap/fast model; escalate only with evidence.

Option 4: Hybrid Retrieval (BM25 + Vector via RRF)

What It Is

Running both a lexical BM25 retriever and a dense vector retriever in parallel, then combining their ranked result lists via reciprocal rank fusion (RRF). The hypothesis is that each retriever catches different failure modes: vector is good for semantic matches, BM25 is good for exact-term matches and rare vocabulary.

Empirical Results

Metric	Vector Only	Hybrid (RRF)	Δ
MRR	Baseline	+6%	modest gain
Recall@k	Baseline	~flat	no change
Precision@k	Baseline	~flat	no change
Faithfulness	Baseline	~flat	no change
Tokens / query	Baseline	~flat	no change
E2E latency	Baseline	~flat	no change

Why it was neutral: CFR text is vocabulary-dense and highly self-consistent — regulations reuse the same controlled terminology across sections. That means a well-tuned vector retriever is already catching most of what a BM25 system would add. MRR rose modestly but downstream generation metrics didn't move, suggesting the additional "catches" weren't load-bearing for the final answer.

When to Try It

- Heterogeneous corpora where rare technical terms matter (code, product IDs, drug names, part numbers)
- Queries that contain exact strings users expect to match literally
- When the retriever is missing obvious keyword hits that a grep would catch

When to Skip

- Controlled-vocabulary corpora where semantic matching already covers the space
- Small corpora where adding BM25 meaningfully increases retrieval latency per query

VERDICT: NEUTRAL. Cheap to test, unlikely to move the needle in controlled-vocabulary domains.

Option 5: Query Rewriting

What It Is

Before embedding, pass the user's natural-language question through an LLM that rewrites or expands it — typically replacing colloquial phrasing with domain vocabulary, adding synonyms, or restructuring as a more retrieval-friendly query. The embedded rewrite is used for retrieval; the original query is typically preserved for generation.

Why You Might Try It

Natural-language questions often don't use the vocabulary of the target corpus. "Can I sell raw milk?" doesn't match "unpasteurized fluid milk products." An LLM rewriter should bridge that gap.

Empirical Results

Metric	Raw Query	Rewritten (Haiku)	Rewritten (Sonnet)
MRR	Baseline	−25%	−21%
Recall@k	Baseline	degraded	degraded
Precision@k	Baseline	degraded	degraded
Faithfulness	Baseline	degraded	degraded
Tokens / query	Baseline	+rewrite call cost	+rewrite call cost
E2E latency	Baseline	+LLM call	+LLM call

Why it backfired: regulatory text uses a precise, carefully-chosen controlled vocabulary. When the LLM expanded "raw milk" into broader related terms, the rewritten embedding drifted toward neighborhoods that contained plausible-but-wrong chunks (dairy processing, food safety more broadly). The original query had been tighter than the rewriter gave it credit for. In a domain where the source vocabulary is already carefully chosen, expansion is drift.

When It Might Still Help

- Open-domain corpora with inconsistent terminology (news archives, forum content)
- Multi-lingual corpora where translation/normalization is a real need
- Extremely short or ambiguous queries ("latest," "refund") that need contextual grounding

When to Skip

- Any corpus written in a controlled vocabulary (regulations, technical documentation, API references)
- Domains where a precise term and its "synonym" mean materially different things (legal, medical, financial)

VERDICT: AVOID in controlled-vocabulary domains. Test carefully before adopting elsewhere.

Option 6: HyDE Query Expansion

What It Is

HyDE (Hypothetical Document Embeddings): before retrieval, an LLM generates a hypothetical passage that would plausibly answer the query in the target corpus's style. The hypothetical passage — not the query — is embedded and used for retrieval. The claim is that a synthetic document is closer in embedding space to real documents than a user's question is.

Empirical Results

Metric	Raw Query	HyDE	Δ
MRR	0.858	0.842	−0.016
Recall@k	Baseline	~flat	no change
Faithfulness	Baseline	~flat	no change
Tokens / query	Baseline	+HyDE gen cost	modest cost
E2E latency	Baseline	+1 LLM call	added latency

Why it didn't help: HyDE is most useful when there's a substantial stylistic gap between how users phrase questions and how the corpus is written. Section-level chunking at the CFR's natural boundary had already closed most of that gap: each chunk is a self-contained regulatory unit whose embedding is well-aligned to regulatory-style queries. The hypothetical-document step added LLM-generation latency for a marginal MRR loss.

When It Might Help

- Corpora where the writing style is very different from natural questions (legal opinions, academic papers)
- When chunking is coarse and context mismatch is large
- Zero-shot retrieval against a corpus you've had no chance to tune chunking on

When to Skip

- When chunking already honors the corpus's natural structure
- Cost-sensitive applications — the extra LLM call doubles retrieval latency and adds tokens for often-negligible gain

VERDICT: SKIP when chunking already bridges the vocabulary gap. Revisit if chunking changes.

Option 7: Hierarchical Retrieval

What It Is

A two-stage retrieval pattern. First, retrieve fine-grained chunks (paragraphs, sentences) to identify which broader units (sections, chapters) are relevant. Then, assemble the full text of those broader units and pass it to the generator as context. The fine-grained pass finds the answer location; the coarse-grained assembly preserves the surrounding semantic dependencies.

Why You Might Try It

Paragraph-level retrieval is precise — it tells you exactly where the answer is. But paragraphs often lack the surrounding definitions, exceptions, and references needed to actually answer the question. Hierarchical retrieval promises both: the precision of fine-grained retrieval and the coherence of section-level context. In theory, it's the best of both chunking strategies.

Empirical Results

Two variants tested, both against a paragraph-chunked corpus:

Variant	Behavior	Result
Unbounded assembly	Retrieve paragraphs, assemble the full containing section for each unique hit	Context tokens ballooned 6.4× over flat retrieval. Every generation metric collapsed.
Bounded assembly (6 sections cap, 4,000 chars each)	Truncate assembled sections to fit a budget	Stayed in budget but truncated answers mid-thought. Citation accuracy rose marginally (0.330 → 0.429); faithfulness never recovered to flat-paragraph levels.

Metric	Flat Paragraph	Hierarchical (Bounded)	Δ
Citation accuracy	0.330	0.429	+0.10 (marginal)
Faithfulness	Baseline	below flat-paragraph	did not recover
Tokens / query	Baseline	+substantial	cost increase
E2E latency	Baseline	+assembly step	added cost

Why It Failed

This is a textbook "lost in the middle" problem from the long-context literature, amplified by a specific regulatory pattern. Unbounded assembly drowned the generator in 6× the context it could usefully attend to. Bounded assembly fragmented the exact semantic connections the assembly step was supposed to preserve — if a section's definitions live in paragraph (a), its rule in (b), and its exceptions in (c), truncating any one of them breaks the meaning the hierarchical pass was built to recover. There was no section-cap value that satisfied both constraints simultaneously.

The deeper lesson: hierarchical retrieval was a workaround for having chunked at the wrong granularity in the first place. When the structural chunking was fixed — by moving to section-level chunks that match the corpus authors' own meaning boundaries — the "problem" hierarchical retrieval solved no longer existed. The flat, section-level retrieval was already delivering the coherent context the hierarchical pass was trying to reconstruct.

When It Might Still Help

- Corpora where fine-grained units are genuinely self-sufficient (FAQs, glossary entries, atomic Q&A pairs)
- Extremely long natural units that don't fit in a reasonable context window even at coarse chunking
- When fine-grained retrieval precision is a real bottleneck *and* the assembled coarse context fits comfortably in the generator's effective context window

When to Skip

- Structured long-form documents (regulations, contracts, technical specs, policy manuals) where the natural unit is already close to the ideal context window
- Anytime chunking at the corpus's natural structural boundary is a viable alternative — that path is shorter, cheaper, and (in this project) more effective

VERDICT: AVOID for structured long-form corpora. It's a workaround for bad chunking; fixing the chunking is the better move.

Option 8: Fine-Tuned Embeddings (voyage-law-2)

What It Is

Replacing a general-purpose embedding model (OpenAI text-embedding-3-small) with a domain-specialized one (Voyage AI's voyage-law-2, a 1024-dimensional model fine-tuned on legal text) to improve semantic alignment between queries and the legal corpus.

Why You Might Try It

Domain-specialized embeddings are intuitively the right tool for a domain-specific RAG: they are trained to place the corpus's vocabulary in a more discriminating geometry. For a federal-regulation corpus, a law-tuned embedding model should align better than a general-purpose one.

The Critical Confound

In this project's experiment, the embedding swap was paired with a simultaneous chunking change: from section-level chunks (85K units) to paragraph-level chunks (224K units). Those two variables — embedding model and chunking granularity — were changed together. That is a methodological error worth calling out explicitly, because it means the results below reflect the joint effect, not the embedding model in isolation.

Empirical Results (Joint Effect)

Metric	text-3-small + Section	voyage-law-2 + Paragraph	Δ
Recall@k	0.833	0.850	+0.017
MRR	0.858	0.710 (post-audit)	−0.15
NDCG@k	0.880	0.720	−0.16
Faithfulness	0.729	0.533	−0.196
Legal accuracy	0.734	0.569	−0.165
Citation accuracy	0.602	0.354	−0.248
Embed cost	Baseline	~3× corpus size; re-embed fee	added cost

What Actually Happened

Recall rose modestly — the domain embeddings did find more relevant content. But MRR and NDCG fell, because the retriever returned three or four paragraph chunks from the same section ahead of the single paragraph that actually answered the question. At generation time, the model saw near-duplicate context and the downstream quality metrics collapsed. The failure was not the embedding model; it was paragraph chunks fragmenting regulatory meaning that lives at the section level. A paragraph (a) defining a rule, (b) listing exceptions, and (c) defining terms are semantically interdependent. Separating them into independent retrieval targets broke the meaning.

The lesson: a domain-specialized embedding model cannot compensate for a chunking strategy that fights the document's natural structure. When the corpus authors have already decided where the

meaning boundaries are (and regulations, contracts, and technical specifications typically have — via sections, clauses, numbered lists), a general-purpose embedding model operating on those boundaries is hard to beat.

When It Might Help

- When the corpus is in a genuinely specialized vocabulary that general-purpose embeddings handle poorly (medical imaging reports, ICD-10 codes, pharmaceutical chemistry)
- When tested with the *same* chunking strategy as the general-purpose baseline — isolate the variable
- When the general-purpose baseline is already measurably bottlenecked by vocabulary alignment

When to Skip

- When the corpus has clear natural structure and a general-purpose model operating on those boundaries already works
- As a first-line optimization — specialized embeddings are expensive to evaluate (corpus re-embedding cost) and hard to interpret without a clean isolated test

VERDICT: CAUTIOUS. Not a first optimization. Test only with matched chunking; expect the corpus's structure to matter more than the embedding's specialization.

Beyond Metrics: An Inference-Time Confidence Signal

The eight options above are evaluated by their effect on test-time metrics — an aggregate score over a fixed evaluation set. But users don't see test-time metrics. They see one answer to one query, and they need a signal for whether to trust it. ***This is especially load-bearing in domains where the cost of acting on a wrong answer is high: legal, medical, financial, compliance, safety.***

The Problem Test-Time Metrics Don't Solve

Suppose a RAG system scores 0.73 faithfulness on average across its eval set. That aggregate hides variance. For any given user, the system may produce a 0.95 answer or a 0.20 answer, and the user has no way to distinguish the two. A low-stakes product (recommendations, content generation) can absorb that variance. A high-stakes product cannot — a legal user acting on a 0.20 answer has materially different outcomes than a user acting on a 0.95 answer, and they deserve to know which they got.

LLM-as-judge at inference time would solve this — but it adds another LLM call per user query, doubling latency and cost. The product needs something cheaper: a signal that rides on top of the data the pipeline already has.

What Was Built

A two-component confidence score, computed without any additional LLM call:

- **Retrieval score** = average cosine similarity of the top-3 retrieved chunks. Proxy for "did the retriever find semantically relevant content?"
- **Citation coverage** = fraction of CFR § references mentioned in the generated text that actually appear in the retrieved chunk set. Proxy for "did the generator ground its claims in what the retriever gave it?"

Composite formula: **0.35 × retrieval_score + 0.65 × citation_coverage**

Bucketed into four tiers surfaced to the user as a visible badge on every answer:

- **high** ≥ 0.75 — strong retrieval and grounded citations; treat as reliable
- **medium** 0.50–0.74 — useful but verify for high-stakes decisions
- **low** < 0.50 — weak retrieval or ungrounded citations; treat with caution
- **not_found** — no relevant content retrieved; show "no relevant regulations found" instead of confabulating an answer

Empirical Behavior

Tier distribution on the 60-question evaluation set:

Tier	% of Queries	n	Avg Faithfulness	Product Behavior
high	71.7%	43	0.759	Surface answer normally
medium	16.7%	10	0.855	Surface with "verify for high-stakes" note
low	5.0%	3	0.850	Surface with strong caution indicator
not_found	6.7%	4	0.000	Suppress answer; show "no relevant regulations found"

What the Signal Reliably Does

- **not_found detection — ship-ready.** 100% of not_found flags in the eval set corresponded to answers with faithfulness exactly 0.000. This tier is perfectly calibrated. Users see a clean "no relevant regulations were found for this query" message instead of a plausible-but-hallucinated answer. For a legal-grade product, this single tier is worth deploying on its own.
- **Aggregate monitoring.** Average confidence across rolling windows of production traffic is a reasonable proxy for system health. A sustained drop triggers a page.

What the Signal Doesn't Reliably Do (Yet)

- **Per-answer high / medium / low ordering.** Medium and low tiers scored higher on faithfulness than high in this project — counterintuitive, driven primarily by small-sample effects (n=10, n=3) and a specific pathology of the citation-coverage component. It penalizes faithful paraphrase that doesn't drop inline § references into the generated text. Those answers are grounded; they just don't cite inline. The formula sees few inline citations and under-scores them.
- **Calibrated probability estimates.** The 0.763 composite score is not a probability that the answer is correct. It's a useful ordering signal; don't present it to users as a percentage.

Why This Matters More in High-Stakes Domains

In low-stakes RAG — product recommendations, content brainstorming, consumer Q&A — a wrong answer is annoying. The user tries again, or they notice the error and move on. In high-stakes RAG — legal, medical, financial, compliance, safety — a wrong answer can drive a wrong decision with real consequences: a compliance filing based on an outdated regulation, a treatment plan based on a misremembered drug interaction, an investment based on a hallucinated disclosure.

In these domains, the confidence signal reframes the question. Instead of asking "is the system accurate enough to ship?" (a threshold question with no clean answer), ask "is the system transparent enough

about when it doesn't know?" The second question has a shipping bar: the not_found case is perfectly calibrated, and that alone is enough to ship safely. A system that is sometimes wrong but reliably tells you when it's uncertain is a fundamentally different product than a system that is sometimes wrong silently.

This is also an ML/AI principle that generalizes well past RAG: calibrated uncertainty is arguably more important than raw accuracy for any model that informs high-stakes decisions. A 90% accurate model that knows when it's in the 10% is safer than a 95% accurate model that is silently overconfident.

Cost

Negligible. Both components are computable from data the pipeline already has — cosine similarities are returned by the retrieval step; citation coverage is a regex pass over the generated text. No additional LLM call. No additional retrieval call. No meaningful latency contribution. The cost is a few dozen lines of code and a visible badge in the UI.

Evaluation Table (For the Matrix)

Metric	Without Signal	With Signal	Effect
Precision@k	Baseline	unchanged	no effect by design
Recall@k	Baseline	unchanged	no effect by design
MRR	Baseline	unchanged	no effect by design
Faithfulness	Baseline	unchanged	no effect by design
Tokens / query	Baseline	unchanged	no added cost
E2E latency	Baseline	unchanged	no added cost
User trust calibration (new)	not measurable without signal	100% of not_found cases correctly suppressed	critical in high-stakes domains

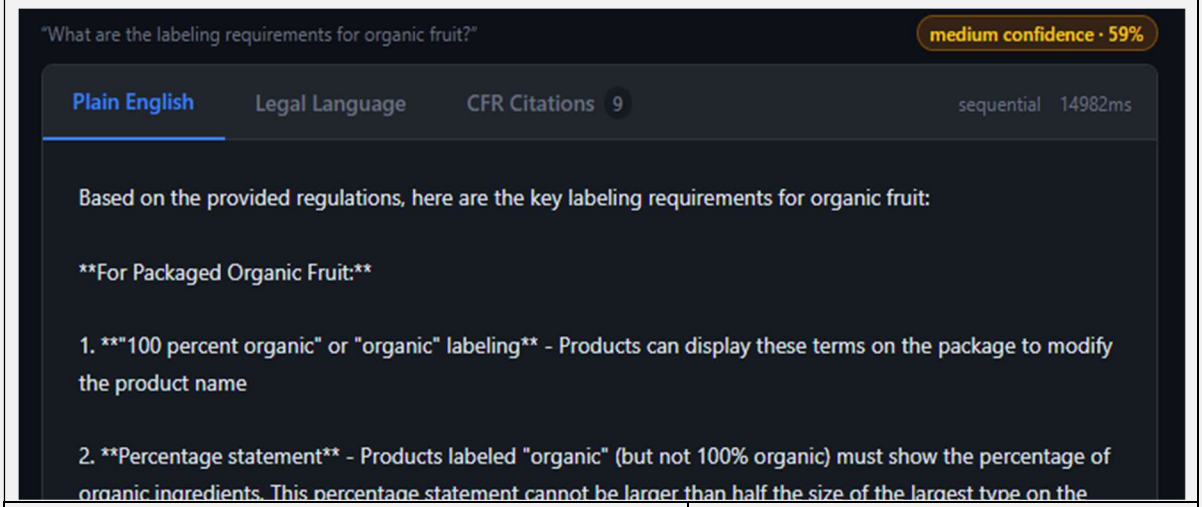
Implementation Roadmap

10. **Ship the not_found tier first.** It's perfectly calibrated, immediately useful, and requires only the existing retrieval similarity scores.
11. **Add tiered badges as a secondary signal, not a guarantee.** Don't show a numeric "72% confidence" — it implies a calibration the formula doesn't support. Show a qualitative tier (high / medium / low / not_found) with guidance appropriate to the stakes.
12. **Expand the eval set to 200+ queries** to tighten the calibration between tiers.

13. **Replace the citation-coverage regex with a semantic grounding check.** A small prompt that verifies whether each factual claim is semantically supported by at least one retrieved chunk. More expensive than the regex, far cheaper than a full judge call.
14. **Reweight the composite with learned weights.** The 0.35 / 0.65 split was an engineering guess. With a larger validation set, grid-search.

VERDICT: ADOPT for any high-stakes RAG system. Ship the not_found case first — it's perfectly calibrated and zero-cost.

Confidence Signal to User – Production UI



Four Confidence Badges:

- High
- Medium
- Low
- No Relevant Content (does not return any info)

Decision Framework: What to Try in What Order

The shape of the empirical results suggests a staged approach rather than a smorgasbord. Each stage is cheap to test, and each stage's outcome shapes what's worth trying next.

Stage 0 — Audit the Harness

Before any optimization, stress-test the evaluation code. Construct cases where the expected answer is obvious and confirm the metrics reward them. This is the single highest-leverage discipline in RAG tuning. In the underlying project, three bugs in the retrieval evaluator had been under-reporting MRR by ~2.5× for several weeks — invalidating the tuning decisions that had been stacked on top.

Stage 1 — Honor the Document Structure

Before reaching for clever embeddings or retrieval techniques, chunk at the document's natural boundaries. Regulations have sections. Contracts have clauses. Technical documentation has function / class / module. The authors have already decided where meaning lives; a chunker that respects those boundaries gives every downstream component a better substrate to work with. In this project, section-level chunking produced a +20-point faithfulness improvement and a +25-point citation-accuracy improvement over paragraph-level chunks — against a domain-specialized embedding model.

Stage 2 — Tune Top-k

The cheapest knob in the stack. Sweep over {3, 5, 8, 10, 15, 20} against your eval set. Expect a sweet spot where recall plateaus and faithfulness peaks. Lock in, re-sweep when generator or corpus changes.

Stage 3 — Consider Sequential Generation

If the product needs multiple distinctly-styled outputs (plain-English + formal + citations, or summary + details + sources), sequential per-output generation typically improves each output at the cost of ~2× latency. Worth it for quality-sensitive products; skip for sub-second conversational UX.

Stage 4 — Cheap Retrieval Experiments

Try hybrid retrieval (BM25 + vector via RRF). It's cheap to add, cheap to remove, and will either unlock exact-term matches your vector retriever is missing or confirm they don't matter. Results in this project were neutral; your mileage may vary by corpus vocabulary heterogeneity.

Stage 5 — Be Cautious About Expensive Optimizations

Query rewriting, HyDE, larger models, and fine-tuned embeddings are the "sophisticated" options most RAG tutorials emphasize. In this project, none of them paid off, and query rewriting was actively

harmful. Each costs meaningfully more per query than the options above. Evaluate them only after Stages 1–4 are locked in, and only with a controlled single-variable experiment.

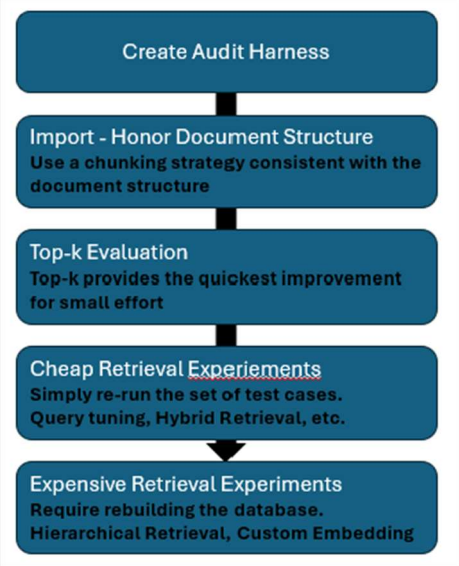
Stage 6 — Ship the Inference-Time Confidence Signal

See the dedicated section above. The not_found tier alone is worth deploying — it was perfectly calibrated in this project's eval set (100% of not_found flags corresponded to answers with faithfulness 0.000) and it costs nothing to compute. For high-stakes domains, this is arguably the single most important production-UX investment a RAG team can make, because it converts a system-reliability problem into a user-transparency feature.

A Common Failure Mode: Hierarchical Retrieval as a Workaround

Teams often arrive at hierarchical retrieval as a rescue for bad chunking. The pattern is recognizable: paragraph chunks are chosen for precision, faithfulness drops because context fragments, and the team adds a hierarchical assembly step to reconstruct what they just tore apart. This was exactly what failed in this project. The cheaper and more effective move is upstream — fix the chunking at the document's natural boundary first, then check whether hierarchical retrieval is still solving a problem you still have. In most cases it won't be.

Staged Decision Framework Flowchart



Scope and Caveats

These findings come from a single corpus (U.S. federal regulations in three CFR titles), a single evaluation set (60 hand-curated regulatory questions), and a specific generation stack (Claude Haiku 4.5, OpenAI embeddings, pgvector). They are not universal laws. Corpus-specific effects that could change the picture:

- **Heterogeneous or multilingual corpora** — hybrid retrieval and query rewriting are likelier to help.
- **Corpora without strong natural structure** — e.g., long narrative documents, chat transcripts, forum posts. Paragraph or semantic chunking may outperform structural chunking, and fine-tuned embeddings may have more room to help.
- **Smaller corpora (<10K chunks)** — every optimization's signal-to-noise ratio changes; small-sample effects dominate.
- **Faster / cheaper frontier models** — the Haiku-vs-Sonnet finding is contingent on the specific Haiku-4.5 ceiling as of April 2026; a weaker base model might make the size upgrade matter.

The replicable discipline, independent of domain, is: a clean evaluation harness, a controlled single-variable experiment, and a willingness to treat every sophistication as falsifiable. The metric matrix in this paper is an output of that discipline, not a substitute for it.

Conclusion

A surprisingly large share of RAG optimization work is the same work done in the same order: audit the metrics, chunk at natural boundaries, tune top-k, consider sequential generation, stop. The more elaborate options — query rewriting, HyDE, hybrid retrieval, bigger models, fine-tuned embeddings — cluster near zero or below for domains with clean structure and controlled vocabulary. This doesn't mean those techniques are wrong; it means they are solutions to specific problems (vocabulary mismatch, stylistic gap, long-tail terminology) that a well-chunked, well-tuned baseline doesn't have.

For practitioners: invest in the evaluation harness first, the structural chunking second, and the knobs in order. For leaders: budget for the unglamorous work — ground truth, single-variable experiments, metric audits — because that is where the compounding gains live. The sophistication every tutorial emphasizes is cheap to add back if the empirical case emerges; the discipline that actually moves the numbers is harder to retrofit.

About This White Paper

All empirical results reported here are drawn from the Government Regulation Query project (April 2026), a production RAG deployment at regs.bradhinkel.com over CFR Titles 7, 21, and 42. Full methodology, individual experiment details, and the inference-time confidence signal are documented in the companion case study.