

Article

Cost-Aware Query Routing in RAG: Empirical Analysis of Retrieval Depth Tradeoffs

Sanjay Mishra ^{1,*}  and Ganesh R. Naik ² 

¹ Independent Researcher, Raleigh, NC 27526, USA

² School of Technology, Torrens University Australia, Adelaide, SA 5000, Australia

* Correspondence: sanmish4@icloud.com

Abstract

When a large language model (LLM) answers a question using retrieved documents, retrieval-augmented generation (RAG) is the standard approach. Retrieving more documents improves answer accuracy but increases cost and response time; retrieving fewer documents saves resources but may miss critical information. Most existing RAG systems sidestep this dilemma by applying the same retrieval setting to every query, regardless of how simple or complex the question is. This wastes budget allocation on easy questions and under-serves hard ones. This paper introduces Cost-Aware RAG (CA-RAG), a routing framework that solves this problem by treating each query individually. For every incoming question, CA-RAG selects the most suitable retrieval strategy from a fixed menu of four options, ranging from no retrieval to fetching the top $k = 10$ most-relevant documents. The selection is driven by a scoring formula that balances expected answer quality against predicted cost and response time. The weights in this formula act as dials: adjusting them shifts the system toward speed, savings, or quality without any retraining. CA-RAG is built on Facebook AI Similarity Search (FAISS) for document retrieval, OpenAI gpt-4o-mini for generation, and text-embedding-3-small for dense retrieval embeddings. We evaluate CA-RAG on a benchmark of 28 queries. The router assigns different strategies to different queries, achieving 26% fewer billed tokens compared to always using heavy retrieval and 34% lower response time compared to always answering without retrieval, while maintaining answer-quality parity in both cases. Further analysis shows that most savings come from simpler queries, where heavy retrieval was unnecessary. All results are reproducible from logged comma-separated value (CSV) files. CA-RAG demonstrates that a small but well-designed set of retrieval strategies combined with lightweight per-query routing can meaningfully reduce the cost and latency of LLM deployments without compromising answer quality.

Keywords: retrieval-augmented generation; cost-aware inference; utility maximization; discrete routing; hybrid retrieval; large language models; token efficiency; production systems



Academic Editor: Peican Zhu

Received: 10 April 2026

Revised: 24 June 2026

Accepted: 2 July 2026

Published: 6 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Large language models (LLMs) deliver strong generative fluency but remain prone to hallucination on fact-intensive or time-sensitive queries [1,2]. Retrieval-augmented generation (RAG) addresses this by conditioning generation on retrieved external passages, substantially improving factual grounding [1,3]. RAG deployments must simultaneously honor latency service-level objectives (SLOs) and token budgets driven by commercial API

pricing [4]. A single static configuration, i.e., fixed top- k , one index, or one decoding budget is rarely optimal across a heterogeneous query workload. Definitional queries (i.e., what is RAG?) waste expensive context tokens on unnecessary retrieved passages, while multi-hop or long-form analytical prompts are not well supported by limited retrieval [5,6].

Running a RAG pipeline costs money at every step. Each document retrieved and added to the prompt is billed as an input token by the API provider. Generating embeddings at query time adds a small but real extra charge. Longer prompts also take more time to process, increasing how long users wait for a response. When a system handles thousands of queries per minute, even a small difference, say, fetching three documents instead of ten, adds up quickly in both cost and response time. This creates a situation where if we set retrieval too low, the complex questions get poor answers. If we set it too high and simple questions waste money unnecessarily. Neither option is ideal, and neither adjusts to the specific needs of each query.

This mismatch motivates *per-query routing*: rather than applying a single-retrieval depth to all queries, each query should independently select the strategy best suited to its complexity and the user's cost-quality objective. An effective routing mechanism must satisfy three requirements. It must be **fast** so that the routing decision does not introduce latency. It must be **transparent** so that operators can audit why a given query was assigned to a given strategy. And it must be **adjustable** so that the same mechanism can serve both latency-sensitive interactive workloads and cost-sensitive batch workloads through simple parameter adjustment rather than re-engineering. While adaptive retrieval has been explored conceptually in prior work, rigorous empirical studies that jointly account for token costs, latency, and answer quality with reproducible run logs, remain limited in the literature. This paper addresses that gap directly.

This paper introduces **CA-RAG** (Cost-Aware RAG), a framework that addresses this gap by routing each query to one of several predefined *strategy bundles*. For each incoming query, CA-RAG selects the bundle that maximizes a weighted score computed from estimated answer quality, predicted latency, and token cost. Unlike approaches that learn a continuous routing policy or train a separate classifier, CA-RAG treats routing as a straightforward optimization over a fixed set of options. This design is intentional: a fixed catalog with clearly defined options is easy to interpret, easy to debug, and ready to deploy without additional training. It also makes the cost latency–quality tradeoff explicit and operator-controllable, rather than burying it inside a learned model. The framework is evaluated around four research questions, each targeting a property that any practical routing system must demonstrate before it can be trusted in production.

RQ1 (Routing Behavior): Does per-query routing genuinely exercise the full bundle catalog, or does it collapse to a near-fixed policy? This question matters because a router that always selects the same bundle is operationally indistinguishable from a static configuration, demonstrating genuine bundle diversity is a necessary precondition for routing to add any value at all.

RQ2 (Cost–Quality Tradeoff): How much token cost and latency does routing save relative to fixed baselines, and at what quality cost? This is the core economic question for any deployment: routing is only worth its added complexity if the measured savings against the natural alternatives (always-light, always-heavy, always-direct) are non-trivial and the quality cost is within acceptable bounds.

RQ3 (Per-Query Variance): Are routing benefits uniform across queries or concentrated in specific query types? Aggregate means can mask important per-query behavior—a router that saves on average but catastrophically underserves a subset of queries is unsuitable for production. Understanding where savings come from informs both bundle design and the placement of guardrails.

RQ4 (Weight Sensitivity): Can the same bundle catalog support multiple operating points on the cost–latency–quality surface through weight adjustment alone? Production priorities shift over time—a system tuned for an interactive chatbot today may be repurposed for an overnight batch pipeline tomorrow—and a routing framework that requires re-engineering for each such shift is a liability rather than an asset.

The contributions of this paper are as follows:

- A concrete utility formulation and discrete bundle catalog suited to operational RAG routing, with full token billing accounting.
- An open reference implementation with retrieval backed by FAISS (Facebook AI Similarity Search, an open-source library for efficient approximate nearest-neighbor search over dense vector embeddings [7]), BM25-ready tokenization, OpenAI gpt-4o-mini/text-embedding-3-small integration (model choice justified in Section 6.2), and a CLI experiment harness.
- A 28-query empirical study with eighteen quantitative figures and six auto-generated tables, all produced directly from measured CSV logs, covering strategy frequency, token decomposition, latency distributions, utility–cost structure, correlation analysis, and router vs. fixed-baseline comparisons.
- Operational guidance for bundle design, weight calibration, monitoring, and failure-mode triage in deployed RAG systems.

2. Related Work

2.1. Retrieval for Open-Domain QA

Dense passage retrieval [5] and hybrid sparse–dense fusion are standard components in open-domain question answering. Retrieval-augmented pretraining (REALM) [8], fusion-in-decoder models [3], and few-shot retrieval-augmented language models (Atlas) [9] demonstrated the value of tightly coupling retrieval with generation at scale. RETRO [10] and nearest-neighbor language models [11] further showed that retrieval can be integrated at pretraining time to expand effective knowledge capacity. REPLUG [12] and in-context retrieval-augmented LMs [13] extended retrieval conditioning to black-box and in-context settings without architectural retraining. RA-DIT [14] jointly fine-tunes retrievers and generators but still assumes a *fixed* retrieval depth per query. CA-RAG builds on this foundation but targets a distinct axis: *operational routing* over a discrete bundle set under explicit cost and latency terms, rather than architectural improvements to retrievers or generators.

2.2. Classical and Hybrid Retrieval

BM25 [15] remains a competitive probabilistic baseline and is complementary to dense retrieval in hybrid pipelines. Learned sparse retrieval (SPLADE) [16], late-interaction models (ColBERT [6], ColBERTv2 [17]), and query expansion via hypothetical document embeddings (HyDE) [18] improve recall and ranking quality but do not address *how much* context to inject at inference time. FAISS [7], HNSW [19], and product quantization [20] enable scalable approximate nearest neighbor search over large embedding indices. Ni et al. [21] survey dense retrieval methods; Thakur et al. [22] provide a heterogeneous zero-shot retrieval benchmark. CA-RAG uses FAISS as the dense backend while preserving BM25-compatible tokenization for future hybrid fusion; its contribution is not retriever architecture but rather *when and how deeply* to invoke retrieval given a fixed backend.

2.3. Adaptive and Active Retrieval

A growing body of work recognizes that indiscriminate retrieval wastes compute on simple queries and under-serves complex ones. Self-RAG [23] trains reflection tokens so

the model can adaptively decide whether to retrieve and how to critique retrieved passages, but it requires specialized model training and does not expose per-query token billing or latency SLOs. FLARE [24] and DRAGIN [25] perform *active* retrieval during generation, triggering lookups when the model exhibits low confidence or unmet information needs. IRCOT [26] interleaves chain-of-thought reasoning with iterative retrieval for multi-hop questions. Corrective RAG (CRAG) [27] adds a lightweight retrieval evaluator that triggers corrective actions when retrieved documents are low quality. Adaptive-RAG [28] is the closest prior work to CA-RAG: it trains a classifier to route queries among no-retrieval, single-step, and multi-step retrieval strategies based on estimated complexity. However, Adaptive-RAG optimizes primarily for *answer accuracy* on academic QA benchmarks. It does not formulate routing as a multi-objective utility maximization over token cost and latency; does not provide operator-controllable weight knobs; and does not publish reproducible per-query logs decomposing embedding, prompt, and completion tokens. Mallen et al. [29] show that parametric knowledge suffices for popular entities while rare facts require retrieval, motivating per-query depth selection but stopping short of an operational routing framework.

2.4. Multi-Hop QA and Query Heterogeneity

Benchmarks such as HotpotQA [30], Natural Questions [31], and MS MARCO [32] establish that query workloads are heterogeneous: definitional lookups, single-hop factoid questions, and multi-hop reasoning prompts coexist in production traffic. Press et al. [33] document a compositionality gap in which LLMs fail on questions requiring combining multiple facts, precisely the regime where deeper retrieval is warranted. These datasets drive retrieval-quality research but are rarely used to evaluate *operational* tradeoffs. CA-RAG's benchmark is intentionally small-scale to enable controlled routing analysis; the gap it addresses is the absence of studies that jointly measure token cost, latency, and answer quality with full billing decomposition on a per-query basis.

2.5. RAG Surveys and Production Considerations

Gao et al. [2] and Wang et al. [34] survey the RAG landscape and identify retrieval granularity, evaluation methodology, and deployment cost as open research challenges. Both surveys note that most systems apply a single retrieval configuration uniformly and call for adaptive strategies, but cite few works that quantify the cost–latency–quality Pareto surface with reproducible telemetry. CA-RAG offers a concrete, measurable response to the cost challenge through per-query routing with full token accounting, operator-adjustable weights, and CSV-logged experiment artifacts from which all figures and tables in this paper are generated.

2.6. Cost-Aware LLM Inference and Model Routing

A parallel line of work optimizes inference cost by routing queries across *LLM providers* rather than retrieval depths. FrugalGPT [35] learns an LLM cascade that escalates from cheap to expensive models based on a post-generation quality scorer, achieving up to 98% cost reduction on classification and QA tasks. RouteLLM [36] trains routers on human preference data (Chatbot Arena [37]) to select between strong and weak LLMs at inference time. LLMLingua [38] compresses prompts post hoc to reduce billed input tokens. These methods address complementary axes of the cost problem—model selection and prompt compression—but none jointly routes *retrieval depth* (top-*k*, skip-retrieval) alongside latency and quality under a single transparent utility function. In a production RAG stack, model routing and retrieval-depth routing are orthogonal decisions; CA-RAG fills the latter gap while remaining compatible with cross-provider normalization (Section 5.8).

2.7. Contextual Bandits and Adaptive Decision-Making

CA-RAG is conceptually related to contextual bandit formulations in which queries are contexts and bundles are actions [39,40]. The benchmark in this paper disables exploration and uses hand-specified priors rather than learned reward models, enabling controlled analysis of the routing mechanism in isolation. Online bandit-based recalibration is identified as a key direction for future work. Relative to learned routers (Adaptive-RAG, RouteLLM, FrugalGPT), CA-RAG’s fixed catalog trades some adaptivity for *deployability*: no training data, no fine-tuning pipeline, and auditable routing decisions via explicit utility attribution (Section 5.9).

2.8. Research Gap

Table 1 summarizes how prior work relates to CA-RAG’s design goals. Adaptive retrieval methods improve answer quality on academic benchmarks but seldom report per-query token decomposition, latency distributions, or operator-controllable cost–quality tradeoffs. Cost-optimization methods target model selection or prompt compression, not retrieval-depth routing. RAG surveys identify the problem but lack reproducible empirical studies with logged CSV telemetry. CA-RAG occupies the intersection: a discrete bundle catalog, explicit utility maximization over quality/latency/cost, full token billing, and an open experiment harness designed for production operators rather than model trainers.

Table 1. Positioning of CA-RAG relative to prior work. ✓ = addressed; ✗ = not primary focus; (~) = partial.

Approach	Adapt. Retr.	Token Billing	Latency SLO	Op. Weights	Repro. Logs
Self-RAG [23]	✓	✗	✗	✗	✗
FLARE [24]	✓	✗	✗	✗	✗
Adaptive-RAG [28]	✓	✗	✗	✗	✗
CRAG [27]	✓	✗	✗	✗	✗
FrugalGPT [35]	✗	✓	✗	(~)	✗
RouteLLM [36]	✗	✓	✗	✓	✗
LLMLingua [38]	✗	✓	✓	✗	✗
RAGRouter [41]	✓	✗	✗	✗	✗
RAG surveys [2,34]	(~)	✗	✗	✗	✗
CA-RAG (this work)	✓	✓	✓	✓	✓

Explicit Differentiation from the Four Closest Prior Systems

Four systems in this space deserve direct differentiation because superficial similarities can obscure fundamental architectural differences.

Self-RAG [23] operates at the *token level*: it trains special reflection tokens ([Retrieve], [IsRe1], [IsSup]) into the LLM weights, so the generator itself decides mid-generation whether to trigger retrieval and whether a retrieved passage is relevant. This requires fine-tuning the LLM on a curated dataset annotated with reflection tokens—a non-starter for operators using black-box commercial APIs. CA-RAG treats the generator as an immutable black box and makes all routing decisions externally at query-submission time, adding zero training overhead and remaining compatible with any commercial API. A direct comparison on token-level routing vs. query-level routing would require running Self-RAG’s inference pipeline on the same query set, which is architecturally incompatible with a black-box OpenAI setup; this is explicitly acknowledged as a limitation (Section 9).

Adaptive-RAG [28] is the closest prior work on the routing axis. It trains a lightweight classifier to route queries among three tiers (no retrieval, single-step, multi-step) based on

predicted query complexity. The classifier is fine-tuned on labeled complexity data from academic QA benchmarks (Natural Questions, MuSiQue, HotpotQA). Three gaps separate it from CA-RAG: (i) it does not model token cost or latency as explicit objectives—all routing decisions are quality-driven; (ii) the trained classifier is fixed after training and does not recalibrate to production telemetry; and (iii) it does not provide operator-adjustable weight knobs for shifting between cost-sensitive and latency-sensitive operation without retraining. CA-RAG addresses all three gaps through its utility function and EWMA recalibration, while Adaptive-RAG’s trained classifier achieves stronger quality routing than CA-RAG’s heuristic complexity signal—a genuine tradeoff between quality routing and operational flexibility.

CRAG [27] adds a lightweight retrieval evaluator that fires *after* retrieval and decides whether to use, discard, or search to supplement the retrieved passages. The corrective action is triggered on a per-passage basis, making it a *post-retrieval* quality gating mechanism rather than a *pre-retrieval* depth selection mechanism. CRAG always incurs the token cost and latency of at least one retrieval pass; CA-RAG avoids this overhead entirely on queries routed to `direct_llm`. The two approaches are complementary rather than competing: CRAG’s corrective evaluator could be integrated as an optional post-retrieval stage within any of CA-RAG’s retrieval bundles.

RAGRouter [41] uses the LLM itself as an in-context routing agent: it prompts the model to classify whether the query requires retrieval, what type of retrieval is appropriate, and which knowledge source to query. This leverages the model’s inherent reasoning about information needs but adds an extra LLM call per query (raising cost and latency) and depends on the generator’s introspective accuracy. CA-RAG’s routing decision is cost-free in LLM calls: it evaluates a closed-form utility expression over four pre-specified bundles without any model invocation, keeping the routing overhead below 1 ms regardless of query complexity.

In summary, Self-RAG requires model fine-tuning; Adaptive-RAG requires labeled training data and sacrifices operational knobs; CRAG operates post-retrieval and always incurs retrieval cost; RAGRouter adds an LLM call overhead per query. CA-RAG’s design point is the only one among these that is training-free, pre-retrieval, cost-zero at routing time, and operator-adjustable via numeric weights.

3. Problem Formulation

Consider a user submitting a query q to a RAG system. The system has a fixed set of retrieval strategies, which we call strategy bundles, collected in a set $\mathcal{B} = \{b_1, b_2, \dots, b_n\}$. Each bundle specifies how many documents to retrieve and how to construct the generation prompt. The system picks one bundle b^* , retrieves a set of relevant passages C_{b^*} (or nothing, if the bundle skips retrieval entirely), and returns an answer a_{b^*} to the user.

The difficulty is that no single bundle works well for every query. Three practical concerns pull in different directions:

- **Quality:** A factual, multi-step question needs several retrieved passages to be answered correctly. A simple definitional question does not. Applying deep retrieval uniformly wastes resources on easy questions without improving answers on hard ones.
- **Latency:** Each additional passage in the prompt makes the model take longer to respond. For user-facing applications, response time matters, and a retrieval strategy that is too heavy can push response times beyond acceptable limits.
- **Cost:** Commercial language model APIs charge by the token. Each retrieved passage injected into the prompt is billed as an input token. At scale, the difference between retrieving three passages and ten passages per query adds up to a measurable difference in operating cost.

These three concerns cannot all be satisfied by a single fixed configuration. A bundle tuned for maximum quality will overspend on simple queries. A bundle tuned for minimum cost will give poor answers on complex ones. What is needed is a way to let each query choose the bundle that best fits its own demands.

CA-RAG does this by assigning a score to each bundle for every incoming query and selecting the highest-scoring one. The score, which we call the utility, combines estimated answer quality with penalties for predicted latency and token cost:

$$U_b(q) = \hat{Q}_b(q) - \lambda_1 \cdot \hat{L}_b(q) - \lambda_2 \cdot \hat{T}_b(q) \quad (1)$$

Here, $\hat{Q}_b(q)$ estimates how well bundle b is likely to answer query q , $\hat{L}_b(q)$ is the predicted response time normalized across bundles, and $\hat{T}_b(q)$ is the predicted token cost, also normalized. The weights λ_1 and λ_2 let the operator decide how much to penalize latency and cost relative to quality. The operational instantiation with weights (w_Q, w_L, w_C) , reference-ratio normalization, and literature-grounded prior construction are given in Equation (10) and Section 5.3. Setting both weights to zero makes the router pick purely for quality. Increasing λ_2 pushes the router toward cheaper bundles. The routing decision is simply the bundle with the highest utility score:

$$b^* = \arg \max_{b \in \mathcal{B}} U_b(q) \quad (2)$$

(equivalently Equation (11) in the bundle-catalog specification).

Because the weights are explicit numbers in a configuration file rather than parameters inside a neural network, the tradeoff is transparent. An operator can look at any routing decision and understand exactly why a particular bundle was chosen.

One clarification is worth stating directly: CA-RAG does not introduce a new method for estimating answer quality. The quality term $\hat{Q}_b(q)$ is a simple heuristic based on observable properties of the query. The contribution of this paper is the routing framework itself—the idea of scoring bundles per query using an explicit utility function—and the experiments that measure whether this approach delivers real cost and latency savings in practice.

4. System Architecture

CA-RAG cleanly separates *routing* from *retrieval and generation*. The router consumes query signals and a bundle catalog and emits a retrieval–generation specification; the execution layer performs retrieval and generation and returns logged metrics to the telemetry store.

End-to-End Pipeline

For each query q , CA-RAG executes the following pipeline:

1. **Signal extraction:** Compute QuerySignals (length, cue-word counts) and a heuristic complexity score $c \in [0, 1]$.
2. **Utility estimation:** Evaluate U_b for all $b \in \mathcal{B}$ using priors and optional telemetry.
3. **Bundle selection:** Dispatch to $b^* = \arg \max_b U_b$.
4. **Retrieval:** Execute retrieval per the specification of b^* , obtaining context C and retrieval confidence.
5. **Generation:** Construct a prompt from q and C ; call the LLM generator.
6. **Telemetry logging:** Log prompt, completion, and embedding tokens; end-to-end latency; lexical quality proxy; and realized utility.

This pipeline makes routing decisions auditable and reproducible at the query level. Figure 1 provides a block-diagram view of the same pipeline, showing how the router, the bundle catalog, the retrieval/generation execution layer, and the telemetry store are decoupled. The router is the only component that consumes both the query signals and the bundle catalog; the execution layer never sees the catalog directly, which keeps the routing logic in a single auditable place and allows bundles to be added, removed, or re-priced without modifying retrieval or generation code.

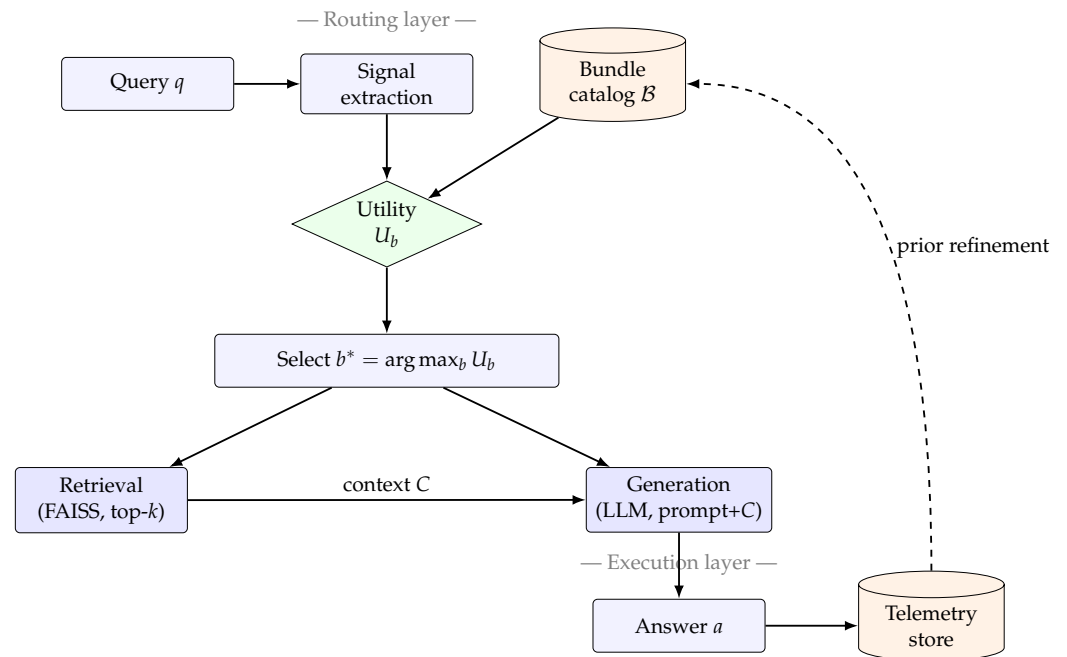


Figure 1. CA-RAG architectural block diagram. The routing layer (top) consumes query signals and the bundle catalog to select b^* via utility maximization. The execution layer (bottom) carries out the chosen retrieval depth and generation and then writes per-query metrics to the telemetry store. The dashed feedback path indicates optional online prior refinement.

5. Methodology

5.1. Query Signals and Complexity

From q , lightweight `QuerySignals` are derived: character length, word count, interrogative cue-word count, and a heuristic *complexity score*

$$c(q) = \text{clip}\left(\alpha \cdot \frac{\text{wordlen}(q)}{L_{\max}} + \beta \cdot \frac{\text{cues}(q)}{K_{\max}}, 0, 1\right),$$

with $\alpha = 0.6$, $\beta = 0.4$, $L_{\max} = 20$, and $K_{\max} = 3$. Complexity modulates quality priors without requiring an additional LLM call.

5.2. Catalog Design and Retrieval Depth Quantization

The four bundles $k \in \{0, 3, 5, 10\}$ represent a discretization of a continuous retrieval depth axis. This section provides an information-theoretic grounding for both the choice of bundle count N and the specific depth intervals. Each design step below is tied to a parameter choice and to prior empirical or decision-analytic literature.

Step 1: Quality saturation model

Empirically, answer quality as a function of retrieval depth k follows an exponential saturation curve:

$$Q(k) = Q_{\text{base}} + Q_{\text{gain}}(1 - e^{-\lambda k}), \quad (3)$$

where Q_{base} is the parametric (zero-retrieval) quality floor, $Q_{\text{gain}} = Q_{\infty} - Q_{\text{base}}$ is the maximum incremental gain from retrieval, and $\lambda > 0$ is the *corpus information-density rate*. The functional form is standard in learning-curve and saturation analyses [42]: the marginal gain $\partial Q / \partial k = Q_{\text{gain}} \lambda e^{-\lambda k}$ is monotonically decreasing, matching the diminishing returns repeatedly reported in open-domain QA when increasing top- k beyond a moderate threshold [3,5,21,34]. Mallen et al. [29] further show that parametric LLM quality is query-dependent—popular entities are answerable without retrieval (Q_{base} is non-trivial), while rare facts require external context (Q_{gain} is large)—motivating a $k = 0$ bundle alongside retrieval depths.

Parameter selection: The anchor $Q_{\text{base}} = 0.52$ is set at the `direct_llm` prior in Table 2 (Section 5.3), reflecting the parametric-only baseline tier. The triplet ($Q(3) = 0.66$, $Q(5) = 0.74$, $Q(10) = 0.82$) is taken from the same quality priors and inverted through Equation (3) to obtain $\lambda \approx 0.21$ and $Q_{\text{gain}} \approx 0.30$. This single-parameter fit is intentionally minimal: three anchor points suffice to identify a one-parameter saturation family, consistent with the sparse top- k ablations in DPR [5] and the coarse retrieval-granularity tiers used in Adaptive-RAG [28]. The upper depth $k_{\text{max}} = 10$ follows common open-domain practice of retrieving on the order of 5–20 passages before reader truncation [3,5]; $k_{\text{max}} = 10$ captures most of the saturation gain ($Q(10) / Q_{\infty} \approx 0.94$ under the fitted curve) without extending into the near-flat tail where marginal token cost dominates [2].

Step 2: Continuous-relaxation optimum

Treating k as continuous and passage cost as linear in k , the unconstrained utility-maximizing depth satisfies $\partial U / \partial k = 0$:

$$k^*(w) = \frac{1}{\lambda} \log \rho, \quad \rho = \frac{w_Q Q_{\text{gain}} \lambda}{w_C c_1}, \quad (4)$$

where c_1 is the billed token increment per additional passage. Equation (4) is the first-order condition for a concave benefit function with linear cost—the same marginal-benefit-equals-marginal-cost logic used in microeconomic dispatch [43] and in cost-aware LLM cascades [35,36]. When $\rho \leq 1$, retrieval never pays at the margin and $k^* = 0$. Because ρ depends on the operator weights (w_Q, w_C), Equation (4) partitions weight space into regions where different bundles are optimal, justifying a *range* of discrete depths rather than a single globally best k [44].

Step 3: Optimal bundle count via quantization theory

Routing over a finite catalog incurs *quantization regret*: The utility loss from selecting the nearest available k_b instead of the continuous optimum $k^*(w)$. Applying the Zador (1982) [45] high-rate bound for scalar quantizers [42] to $U(k)$, the minimax regret for N levels satisfies:

$$\begin{aligned} \epsilon(N) &\lesssim \left(\frac{I}{2N} \right)^2, \\ I &= \int_0^{k_{\text{max}}} |U''(k)|^{1/2} dk \\ &= 2\sqrt{w_Q Q_{\text{gain}}} \left(1 - e^{-\lambda k_{\text{max}}/2} \right). \end{aligned} \quad (5)$$

Solving $\epsilon(N) \leq \epsilon_{\text{tol}}$ yields the minimum sufficient catalog size:

$$N^* = \left\lceil \frac{I}{2\sqrt{\epsilon_{\text{tol}}}} \right\rceil. \quad (6)$$

Parameter selection: We set $\epsilon_{\text{tol}} = 0.02$, i.e., at most 2% utility regret from discretizing k , following the sensitivity-analysis practice in multi-criteria decision making, where sub-

5% objective perturbations are treated as operationally negligible [44,46]. With $\lambda = 0.21$, $Q_{\text{gain}} = 0.30$, $k_{\text{max}} = 10$, and $w_Q = 1.0$ (the default quality weight in Equation (10)), Equation (6) yields $N^* = 3$ non-zero retrieval levels. Adding the $k = 0$ no-retrieval bundle gives a total catalog of four—the smallest discrete set that (i) covers the skip-retrieval regime identified by Mallen et al. [29], (ii) spans the elbow of the saturation curve, and (iii) keeps quantization regret below ϵ_{tol} .

Step 4: Geometric depth spacing

The distortion-minimizing breakpoint placement from the Lloyd–Max optimality conditions [47] allocates levels proportional to $|U''(k)|^{-1/3}$, concentrating quantizer resolution where the utility curvature is highest [42]. For $U''(k) = -w_Q Q_{\text{gain}} \lambda^2 e^{-\lambda k}$, this reduces to geometric spacing:

$$k_i = k_1 \cdot r^{i-1}, \quad r = \left(\frac{k_{\text{max}}}{k_1} \right)^{1/(N-1)}, \quad (7)$$

Parameter selection: The minimum non-zero depth $k_1 = 3$ is the shallowest top- k that yields a measurable retrieval gain in DPR-style pipelines [5] and matches the “light” retrieval tier in adaptive routing systems [28]. With $k_1 = 3$, $k_{\text{max}} = 10$, $N = 3$: $r = \sqrt{10/3} \approx 1.83$, giving continuous levels $\{3, 5.5, 10\}$, rounded to $\{3, 5, 10\}$ —the non-zero depths in Table 2. Arithmetic spacing ($\{3, 6, 9\}$ or $\{2, 5, 8\}$) would over-sample the saturated tail where $|U''(k)|$ is small and under-sample the elbow where each additional passage changes routing outcomes most [45].

Table 2. Strategy bundle catalog: retrieval depth and quality/latency priors with shared generation profile.

Bundle	k	Skip Retr.	Qual. Prior	Lat. Prior (ms)
direct_llm	0	✓	0.52	8
light_rag	3	✗	0.66	45
medium_rag	5	✗	0.74	60
heavy_rag	10	✗	0.82	95

5.3. Strategy Bundles

Table 2 defines the four bundles used in this study. Retrieval depth ranges from no retrieval (direct_llm) to top- $k = 10$ dense retrieval (heavy_rag). The depths and catalog cardinality follow Section 5.2; this subsection documents how the *numeric priors* in Table 2 are chosen, why normalization is applied before utility comparison, and why routing is driven by ordinal structure rather than cardinal calibration.

Step 1: Shared generation specification.

All bundles share `paper_gen`: 256 maximum output tokens and temperature 0. Fixed decoding parameters isolate retrieval-depth effects and align with reproducible LLM evaluation practice [48], where temperature-zero sampling reduces run-to-run variance. The 256-token cap matches the short-form answer regime of the benchmark queries and bounds the $\tau_{\text{completion}}$ term in the standard additive token-billing identity

$$\tau_{\text{billed}} = \tau_{\text{prompt}} + \tau_{\text{completion}} + \tau_{\text{embed}}, \quad (8)$$

which follows commercial LLM API pricing (input, output, and embedding tokens billed separately) [2,4,35] and is used without modification to form $\hat{C}_b$ in Equation (10).

Step 2: Quality priors (first source).

The quality priors (0.52 for `direct_llm` through 0.82 for `heavy_rag`) are hand-specified seed beliefs, not learned posteriors. They are anchored to three literature sources:

- **Parametric vs. retrieved gap:** RAG consistently outperforms parametric-only generation on knowledge-intensive queries [1,3,5,12], establishing the *ordering* $Q_{\text{direct}} < Q_{\text{light}} < Q_{\text{medium}} < Q_{\text{heavy}}$.
- **Diminishing top- k returns:** DPR and follow-on work show that increasing reader context from a handful of passages to ~ 20 yields gains that taper sharply [5,21]; the spacing between 0.66, 0.74, and 0.82 reflects this concavity and is consistent with the fitted saturation curve in Equation (3).
- **Query-heterogeneity floor:** Even without retrieval, LLMs answer a non-trivial fraction of queries from parametric memory [29], justifying $Q_{\text{base}} = 0.52 > 0$ rather than treating `direct_llm` as zero quality.

The priors are mapped to $[0, 1]$ as *single-attribute value functions* $v(Q)$ in the sense of multi-attribute utility theory [49,50]: they encode relative standing on a bounded ratio scale, not interval-calibrated accuracy. What matters for routing is **ordering and relative spacing**—measurable value functions are unique only up to a positive affine transformation $Q' = aQ + b$ with $a > 0$ [49,51], and under a fixed weighted-sum aggregator (Equation (10)), any such transform applied uniformly to all bundles leaves the arg max ranking unchanged because only utility *differences* across alternatives matter [51,52]. Absolute calibration to an external benchmark (e.g., exact F1 on Natural Questions [31]) is therefore unnecessary at routing time; telemetry recalibration (Step 5) replaces cardinal accuracy with deployment-specific estimates. The saturation form in Equation (3) used to space the quality priors is likewise taken from the literature without modification [5,42].

Step 3: Latency priors (second source).

Latency priors (8 ms to 95 ms) come from short pilot runs of each bundle on the benchmark corpus, rounded to order-of-magnitude values. This follows standard systems benchmarking: measure component latencies separately before composing an end-to-end budget [53]. `direct_llm` omits embedding lookup and FAISS search; `light_rag` adds query embedding [5,54] and a single IndexFlatIP search whose wall-clock cost follows the textbook brute-force bound

$$T_{\text{exact}}(N) \propto Nd, \quad (9)$$

for N indexed passages of dimension d [7]; `heavy_rag` adds top-10 retrieval and longer prompt prefill, which increases both retrieval and generation latency. Equation (9) is used as-is from the FAISS complexity literature; CA-RAG does not modify it. The priors need not be precise: they are *ordinal* indicators that deeper retrieval is slower, mirroring the latency–recall trade-off documented for exact vs. approximate nearest-neighbor search [7, 19]. Monotonicity ($L_{\text{direct}} < L_{\text{light}} < L_{\text{medium}} < L_{\text{heavy}}$) is the operative constraint.

Step 4: Normalization before utility comparison (third source).

Quality, latency, and token cost arrive in incomparable units (dimensionless index, milliseconds, billed tokens). Combining them without scaling would let the numerically largest raw term dominate regardless of operator intent [44,46]. CA-RAG therefore adopts three standard forms from the literature without modification.

(i) *Weighted-sum scalarization:* Bundle utility is the linear compensatory aggregate

$$U_b = w_Q \hat{Q}_b(q) - w_L \hat{L}_b^{\text{norm}} - w_C \hat{C}_b^{\text{norm}}, \quad (10)$$

i.e., the classical *weighted-sum model* (WSM) for multi-attribute decision making [44,46,49], in contrast to outranking methods (e.g., ELECTRE) that use pairwise dominance relations instead of a single scalar score [55]. Geoffrion [52] provides the Pareto-optimality conditions

under which such linear scalarizations are admissible. CA-RAG's contribution is not Equation (10) itself but the discrete bundle catalog and operator weights that instantiate it. Bundle selection uses the standard winner-takes-all rule

$$b^* = \arg \max_{b \in \mathcal{B}} U_b, \quad (11)$$

which is a deterministic policy over a finite action set [39,40].

(ii) *Reference-ratio normalization*: Latency and cost are commensurated via division by operator reference points,

$$\hat{L}_b^{\text{norm}} = \frac{\hat{L}_b}{L_{\text{ref}}}, \quad \hat{C}_b^{\text{norm}} = \frac{\hat{C}_b}{C_{\text{ref}}}, \quad (12)$$

and the ratio-normalization variant catalogued by Hwang and Yoon [46] and Keeney and Raiffa [49] for making heterogeneous attributes comparable. Defaults $L_{\text{ref}} = 800$ ms and $C_{\text{ref}} = 3500$ tokens are deployment-specific anchors; the functional form of Equation (12) is unchanged from the MCDA literature. The weights (w_Q, w_L, w_C) then encode marginal rates of substitution on the common scale [43,44].

(iii) *Additive cost from token counts*: The cost numerator $\hat{C}_b$ applies Equation (8) and normalizes by C_{ref} , following the per-token billing model of commercial LLM APIs [4, 35]. Cross-provider extensions use the dimensionless cost factor Φ_p^C (Section 5.8), which is likewise a ratio of billed amounts [35,36]. Because only ratios enter Equation (10), multiplying all latency priors by a common positive constant leaves routing unchanged—a consequence of the ratio-scale construction in Equation (12) [51].

Step 5: Telemetry recalibration.

The priors are *seeds*, not permanent fixtures. After sufficient bundle observations, realized `quality_proxy`, `latency`, and `token` fields replace seed values via the standard exponentially weighted moving-average (EWMA) update [56,57]

$$\mu_t = (1 - \alpha) \mu_{t-1} + \alpha x_t, \quad (13)$$

where x_t is the observed metric on the t -th execution of a bundle, and $\alpha = 0.15$ is the smoothing constant (authors' deployment choice; the recurrence itself is unmodified from the forecasting literature). The implied half-life $n_{1/2} = \ln(0.5) / \ln(1 - \alpha) \approx 4.3$ observations at $\alpha = 0.15$ follows directly from Equation (13) [57]. Importance-weighted variants (Section 5.7) apply the self-normalized importance-sampling correction of Sugiyama et al. [58] without altering the EWMA structure. Conceptually, replacing μ with telemetry estimates is the production analogue of bandit reward estimation [39,40]: priors supply a cold-start policy; Equation (13) supplies the online update rule.

Step 6: Default utility weights.

The weights $(w_Q, w_L, w_C) = (1.0, 0.35, 0.45)$ are *product parameters* elicited from five principled criteria, not fitted hyperparameters tuned for benchmark leaderboard position [44,49]. The criteria and the derivation of each numeric value are stated explicitly below.

(C1) Anchor normalization: The utility function $U = w_Q Q - w_L L_n - w_C C_n$ is positive-homogeneous of degree one in the weight vector: multiplying all three weights by any positive scalar λ yields λU , which leaves $b^* = \arg \max_b U_b$ unchanged [51]. Setting $w_Q = 1$ anchors the scale so that w_L and w_C are directly interpretable as *marginal rates of substitution* (MRS)—the quality units the operator is willing to sacrifice for a unit improvement in normalized latency or cost [43,49].

(C2) Non-dominance at the reference operating point: At the worst-case operating point $L_n = C_n = 1$ (latency and cost both at their reference values), the quality term must still produce positive utility for any viable bundle. This requires $w_Q \cdot Q_{\min} > w_L + w_C$, i.e.,

$$Q_{\min} > \frac{w_L + w_C}{w_Q}. \quad (14)$$

With the default weights, the right-hand side is $0.35 + 0.45 = 0.80$. The heaviest bundle (heavy_rag) has a quality prior of $0.82 > 0.80$, so all four bundles maintain positive utility at reference load, preventing the cost and latency penalties from unconditionally dominating quality [52]. Choosing $w_L + w_C > w_Q$ would violate this constraint and cause high-complexity queries to always prefer direct_llm regardless of retrieval quality.

(C3) SLO-derived latency weight: The MRS between quality and normalized latency is $w_L/w_Q = 0.35$. At $L_{\text{ref}} = 800$ ms, this encodes the operator judgment such that:

$$\Delta Q_{\text{equiv}} = \frac{w_L}{L_{\text{ref}}} \cdot \Delta L = \frac{0.35}{800} \cdot \Delta L \approx 0.044\% \text{ quality per ms.}$$

Concretely, a 100 ms latency reduction is worth a 4.4 percentage-point quality gain, and eliminating the full 800 ms reference latency is worth 35 percentage points. This calibrates to an *interactive assistant* deployment where users tolerate moderate waiting but not large quality degradations. For batch or asynchronous pipelines, w_L should be reduced toward zero; for real-time streaming, increased toward 1.0 (approaching a hard-deadline structure modeled by the logistic penalty in Equation (22)).

(C4) Per-query cost budget alignment: The MRS between quality and normalized cost is $w_C/w_Q = 0.45$. At $C_{\text{ref}} = 3500$ tokens, eliminating all reference-budget tokens is worth 45 quality percentage points:

$$\Delta Q_{\text{equiv}} = \frac{w_C}{C_{\text{ref}}} \cdot \Delta \tau = \frac{0.45}{3500} \cdot \Delta \tau \approx 0.013\% \text{ quality per token.}$$

At gpt-4o-mini pricing (\$0.000150 per 1 k input, \$0.000600 per 1 k output), the 3500-token reference budget costs $\approx \$0.00053$. The operator therefore implicitly values a 1-percentage-point quality improvement at approximately \$0.0012—meaningful cost sensitivity without making cost the primary routing driver. Setting $w_C > w_Q$ would make cost the dominant objective, forcing the router to direct_llm on essentially all queries.

(C5) Bundle-distinguishing power at cold start: The weights must produce utility gaps between adjacent bundles that are large enough for routing to be non-arbitrary. Computing prior utilities under the default weights at zero query complexity (cold-start priors only) yields Table 3.

Table 3. Cold-start prior utilities under the default weights ($c = 0$), showing normalized quality $\hat{Q}$, normalized latency $\hat{L}_n$, normalized cost $\hat{C}_n$, and the resulting utility U_b .

Bundle	$\hat{Q}$	$\hat{L}_n$	$\hat{C}_n$	U_b
direct_llm	0.637	0.485	0.130	0.409
light_rag	0.714	0.531	0.330	0.379
medium_rag	0.758	0.550	0.445	0.365
heavy_rag	0.802	0.594	0.702	0.278

The minimum adjacent gap is $\Delta U(\text{light}, \text{medium}) = 0.014$, which is below the $\epsilon_{\text{tol}} = 0.02$ catalog design threshold. This is intentional: at cold start, the router *conservatively* prefers direct_llm for all queries until telemetry accumulates evidence that heavier bundles deliver superior quality on this workload [39,40]. The prior-based order-

ing (direct_llm $\succ$ light $\succ$ medium $\succ$ heavy) implements the risk-averse initialization of Thompson sampling; the 57% medium_rag selection rate in the benchmark reflects the posterior after telemetry recalibration, not the prior. The ϵ_{tol} criterion applies to the *continuous* k^* quantization loss (Equation (6)), not to the prior routing margin; once EWMA quality estimates replace priors after ≈ 5 bundle observations, the inter-bundle utility gaps become workload-specific.

Sensitivity to weight perturbations: The decision margin $\Delta = U(b^*) - U(b_2) = 0.030$ (direct vs. light at cold start) is small. The weight sensitivity analysis (Equations (37)–(40)) gives $\Delta w_L^{\text{flip}} = -0.21$, i.e., reducing w_L by 0.21 from 0.35 to 0.14 would flip the cold-start preference from direct_llm to light_rag. This confirms that the cold-start routing is mildly sensitive to w_L but robust to w_C perturbations ($\Delta w_C^{\text{flip}} = +0.55$). Operators with lower latency sensitivity should decrease w_L below 0.20 to allow retrieval bundles to win at cold start. Sensitivity analysis in Section 7 confirms that the same catalog supports multiple operating points under varied weights, consistent with the weight-space partition induced by Equation (4).

5.4. Causal Framework: Retrieval Confidence and Answer Quality

Retrieval confidence $s \in [0, 1]$ (the maximum cosine similarity between the query embedding and any retrieved passage) is an observable proxy for corpus coverage. The central question—*does injecting retrieved context causally improve answer quality?*—requires a causal, not merely correlational, analysis, because s is confounded: hard queries simultaneously attract low-confidence retrieval *and* low parametric quality, making the naive correlation $\text{Cor}(s, Q_{\text{RAG}})$ uninformative about the treatment effect of retrieval.

5.4.1. Causal DAG

The generative structure is captured by the following structural causal model [59]:

$$\begin{aligned} s &\leftarrow h(q, \text{corpus}) && \text{corpus coverage of } q, \\ Q_d &\leftarrow f_d(q) && \text{quality under direct LLM,} \\ Q_r &\leftarrow f_r(q, \text{context}(s)) && \text{RAG quality.} \end{aligned} \quad (15)$$

Query q confounds both s and both potential outcomes (Q_d, Q_r). The *conditional average treatment effect* (CATE) of retrieval at confidence level s is:

$$\tau(s) = \mathbb{E}[Q_r - Q_d \mid S = s]. \quad (16)$$

For high-coverage queries ($s \rightarrow 1$), retrieved passages are on-topic and $\tau(s) > 0$. For low-coverage queries ($s \rightarrow 0$), the corpus contains only tangentially related passages; injecting them adds *noise* to the prompt, so $\tau(s) < 0$ —retrieval hurts.

5.4.2. Mediation Decomposition

The total effect of retrieval on Q can be decomposed via front-door adjustment [59] into a *direct path* (query $\rightarrow$ answer, bypassing context) and a *mediated path* (query $\rightarrow$ retrieval $\rightarrow$ context $\rightarrow$ answer). The mediated path carries positive signal only when s exceeds the break-even threshold s^* defined by:

$$\tau(s^*) = 0 \iff \mathbb{E}[Q_r \mid S = s^*] = \mathbb{E}[Q_d]. \quad (17)$$

Below s^* , the direct path dominates and the system should fall back to parametric generation; above s^* , the mediated path contributes positive utility and retrieval should proceed.

5.4.3. Parametric Threshold Estimate

Under the quality saturation model (Equation (3)), context quality is approximately $s \cdot Q_{\text{gain}}$ when the retrieved passage is linearly scored. A linear-in- s model for the CATE gives:

$$\tau(s) \approx Q_{\text{gain}} \cdot s - \delta_{\text{noise}}(1 - s)^2, \quad (18)$$

where δ_{noise} is the prompt-pollution penalty for irrelevant context. Setting $\tau(s^*) = 0$ yields:

$$s^* = \frac{\delta_{\text{noise}}}{\delta_{\text{noise}} + Q_{\text{gain}} + \delta_{\text{noise}}/Q_{\text{gain}}}. \quad (19)$$

For $Q_{\text{gain}} = 0.30$ and a conservative $\delta_{\text{noise}} = 0.10$, Equation (19) gives $s^* \approx 0.70$. The bimodal retrieval confidence distribution observed in Section 7.3 places the low-confidence cluster in $[0.55, 0.70]$ —exactly the predicted harm regime.

5.4.4. Confidence-Based Fallback Mechanism

The orchestrator implements Equation (17) as a two-threshold policy controlled by `confidence_fallback_threshold` s_{fall} and `confidence_blend_threshold` s_{blend} :

- $s < s_{\text{fall}}$: **hard fallback**—discard all retrieved context; regenerate using the direct-LLM path. The CATE is estimated negative; retrieval is harmful.
- $s_{\text{fall}} \leq s < s_{\text{blend}}$: **soft blend**—retain only the single highest-scoring passage, limiting context noise while preserving some grounding signal.
- $s \geq s_{\text{blend}}$: **full context**—inject all top- k passages as normal.

Both thresholds default to 0 (disabled) for backward compatibility. The Boolean `fallback_triggered` field on every `CARAGResult` enables offline causal analysis: pairing responses with and without fallback on matched queries estimates $\hat{\tau}(s)$ empirically, enabling threshold recalibration from production telemetry.

5.5. Utility Function and Bundle Selection

Let $\hat{Q}_b(q)$ be the estimated quality for bundle b on query q , $\hat{L}_b$ the expected latency, and $\hat{C}_b$ the expected total token cost. The *base* selection utility is Equation (10), with normalization per Equation (12) and cost accounting per Equation (8); (w_Q, w_L, w_C) are operator-specified weights with defaults (1.0, 0.35, 0.45). The orchestrator selects b^* via Equation (11).

5.5.1. Theoretical Adequacy for Finite Bundle Sets

Equation (10) is a linear scalarization of three objectives. For a *continuous* feasible set, linear scalarization cannot recover non-convex regions of the Pareto front [52]. However, the bundle set $\mathcal{B}$ is finite; every Pareto-optimal bundle is reachable by some weight vector w . Therefore, linear scalarization is *representationally sufficient* for this system—the relevant question is whether a *single fixed* weight vector is adequate across the query distribution, which it is not when query types exhibit heterogeneous operational constraints. This motivates the three extensions described below.

5.5.2. Context-Conditional Weights

Rather than using fixed (w_Q, w_L, w_C) , the weights are scaled by the query complexity score $c \in [0, 1]$:

$$\begin{aligned} w_Q(c) &= w_Q(1 + 0.4c), \\ w_L(c) &= w_L(1 - 0.3c), \\ w_C(c) &= w_C(1 - 0.25c). \end{aligned} \quad (20)$$

Complex queries increase quality sensitivity and reduce latency/cost sensitivity—a user posing a multi-step analytical question accepts a longer wait for a more thorough answer; a simple definitional query demands a fast response. Context-conditional weights are enabled by the `use_context_weights` configuration flag (default off for backward compatibility).

5.5.3. Concave Quality Transform

The linear quality term treats a gain of ΔQ identically regardless of the operating point. For complex queries, this undervalues quality at the high end, where the difference between a partially correct and a fully correct answer is operationally significant. When `use_nonlinear_quality` is enabled, quality is replaced by the normalized concave transform:

$$\phi(\hat{Q}_b, c) = \frac{\log(1 + k(c) \hat{Q}_b)}{\log(1 + k(c))}, \quad k(c) = 1 + \kappa c, \quad (21)$$

where $\kappa = 2.0$ by default. At $c = 0$, $k = 1$, and ϕ is a mild log-curve; at $c = 1$, $k = 3$, and the transform strongly amplifies marginal quality gains near the high end.

5.5.4. Hard Latency Deadline

For interactive deployments where any response exceeding a wall-clock deadline L_D is unacceptable regardless of quality, a logistic step penalty is added:

$$U_b = w_Q(c) \phi(\hat{Q}_b, c) - w_L(c) \hat{L}_b^{\text{norm}} - w_C(c) \hat{C}_b^{\text{norm}} - \lambda_D \sigma(\alpha(L_b - L_D)), \quad (22)$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is the logistic function, $\alpha = 0.02$ controls the steepness of the step, and $\lambda_D = 10$ ensures that the deadline penalty dominates quality and cost terms near L_D . When `latency_deadline_ms` is `None` (the default), the penalty term vanishes and Equation (22) reduces to Equation (10).

5.5.5. Non-Compensatory Quality Floor

A `quality_floor` parameter (default 0.0, i.e., disabled) enforces a lexicographic quality constraint: any bundle whose estimated quality falls below the floor receives utility -10^9 , making it unselectable regardless of latency or cost savings. This prevents the linear formulation from routing to a factually inadequate bundle purely on efficiency grounds, which is important for high-stakes query classes.

After execution, a *realized utility* $\tilde{U}_b$ is computed by substituting observed latency and token counts into Equation (22) with the same configuration, enabling post hoc telemetry analysis.

5.6. Quality Proxy: NLU-Theoretic Framework

The quality proxy $Q_{\text{obs}}(q, C, a)$ maps a query, retrieved context, and generated answer to a scalar in $[0, 1]$ used for telemetry and realized-utility computation. The baseline implementation uses unigram query–answer Jaccard overlap—a bag-of-words metric that measures surface co-occurrence with no compositional interpretation. From natural language understanding theory, this proxy fails systematically on three query classes where lexical similarity and semantic correctness decouple.

5.6.1. Failure Taxonomy

Under distributional semantics [60], word co-occurrence statistics capture *topical* similarity but not *logical* or *referential* correctness. Montague (1974) [61] shows that correct semantic interpretation requires compositional, model-theoretic evaluation; surface-form statistics are insufficient proxies. The three critical failure modes are:

1. **Numerical reasoning:** Correctness is a discontinuous function on $\mathbb{R}$: “12%” and “21%” share the same character-level and token-level similarity yet are semantically opposite in many contexts. Lexical overlap assigns near-perfect similarity to answers that differ only in a digit.
2. **Logical deduction:** Under formal semantics, logical validity requires model-theoretic interpretation. Two answers with identical surface tokens can be a valid syllogism or a blatant contradiction depending on quantifier scope and negation placement. The Jaccard metric is insensitive to negation polarity.
3. **Factual verification:** Entity-level accuracy matters more than word co-occurrence: “Napoleon was born in 1769” and “Napoleon was born in 1679” differ in one digit, but the lexical overlap is near-perfect while the factual content is entirely different.

5.6.2. Query-Type-Aware Composite Proxy

To address these failure modes without requiring a reference answer or an external LLM judge (which would add latency and cost), the system classifies each query into one of four types using lightweight regular-expression heuristics and dispatches to a type-appropriate scorer:

- **NUMERICAL** queries: checks numeric presence (if the query demands a quantity and the answer contains none, a -0.20 penalty applies) and context grounding (answer quantities should be traceable to retrieved passages within 1% relative tolerance; ungrounded numbers suggest hallucination).
- **LOGICAL** queries: detects self-contradiction (sentences that assert X and $\neg X$ sharing ≥ 2 substantive words receive a penalty of up to -0.30) and rewards conclusion completeness (fraction of answer words novel relative to the query).
- **FACTUAL** queries: computes entity-level precision and recall against the retrieved context using a lightweight title-case NER heuristic, approximating the SQuAD entity F1 metric [62] without a reference answer.
- **DESCRIPTIVE** queries: falls back to the lexical baseline, which is adequate when topic keywords in the question should reappear in the answer.

The composite score blends the type-specific scorer ($\alpha = 0.70$) with the lexical baseline ($1 - \alpha = 0.30$):

$$Q_{\text{obs}} = \alpha \cdot Q_{\text{type}}(q, C, a) + (1 - \alpha) \cdot Q_{\text{lex}}(q, a), \quad (23)$$

ensuring scores remain monotonically related to the lexical baseline when type-specific signals are absent and allowing existing telemetry comparisons to remain valid. The query type is logged in every experiment CSV row under `query_type`, enabling per-type quality analysis in downstream studies.

5.6.3. Extended Metrics (Q4 Supplement)

Two additional metrics are implemented in `ca_rag/quality.py` and evaluated in Section 8.8: (i) **ROUGE-L recall** [63] against the retrieved context, which uses the longest-common-subsequence match to reward structural coverage beyond unigram overlap and (ii) **Semantic similarity** via Sentence-BERT (all-MiniLM-L6-v2) [54], which captures paraphrase and lexical variation invisible to surface metrics. Both are reference-free: the retrieved context serves as the comparison target rather than a gold answer. The evaluation script (`scripts/evaluate_quality_metrics.py`) computes all three metrics and a per-query inter-metric correlation report; Section 8.8 shows that Jaccard and ROUGE-L are negatively correlated ($r = -0.21$) and that Jaccard is completely blind to hallucination of ungrounded numbers ($\Delta = 0$ where ROUGE-L registers $\Delta = +0.14$).

5.7. Workload Distribution and Concept Drift

The routing and telemetry mechanisms implicitly assume that queries arrive i.i.d. from a fixed, light-tailed complexity distribution. Production environments violate all three assumptions: query streams exhibit long-tail frequency distributions, non-stationarity (concept drift), and query-type heterogeneity. This section provides a probabilistic framework for each failure mode.

5.7.1. Long-Tail Workload Model

Empirically, query-type frequency in information retrieval systems follows a *Zipf distribution* [64]:

$$p_r = \frac{r^{-s}}{H_n(s)}, \quad H_n(s) = \sum_{j=1}^n j^{-s}, \quad (24)$$

where r is the rank of the query type by frequency, and $s > 0$ is the exponent. For $s = 1$ (standard Zipf) and $n = 4$ bundles, $p_1 \approx 0.48$, $p_4 \approx 0.12$: the least-selected bundle receives only one-quarter the traffic of the most-selected. For $s > 1$ (heavy tail), the ratio grows without bound as n increases.

The EMA telemetry update $\mu_t = (1 - \alpha)\mu_{t-1} + \alpha x_t$ converges to $\mathbb{E}_{\hat{p}}[X]$ under the empirical distribution $\hat{P}$, not the value-weighted distribution $V[X] = \mathbb{E}[X \cdot v(x)] / \mathbb{E}[v(x)]$. For a Zipf workload with exponent s , the estimation error for the least-frequently selected bundle (rank n) scales as $O(n^{s-1})$ —the EMA is asymptotically unbiased for $s < 1$ but diverges for $s > 1$.

5.7.2. Importance-Weighted EMA Correction

The self-normalized importance sampling estimator [58] corrects EMA bias for rare query types by inflating the update step:

$$\alpha_{\text{eff}}(b) = \text{clip}\left(\alpha \cdot \frac{p_{\text{ref}}}{\hat{p}(b)}, 0, \alpha_{\text{max}}\right), \quad (25)$$

where $p_{\text{ref}} = 1/|\mathcal{B}|$ is the target (uniform) reference probability, and $\hat{p}(b)$ is the empirically observed selection fraction. When bundle b is rarely selected ($\hat{p}(b) \ll p_{\text{ref}}$), $\alpha_{\text{eff}}(b) > \alpha$ so that each observation of b contributes proportionally more to its telemetry estimate. The clip at $\alpha_{\text{max}} = 0.80$ prevents divergence when $\hat{p}(b) \rightarrow 0$. This is enabled via `use_importance_weighting = True` in `CARAGConfig`.

5.7.3. Concept Drift: CUSUM Detection

A stationary EMA with fixed momentum α tracks a drifting mean with lag $\approx 1/\alpha$ observations. For drift rate δ (shift in mean complexity per query), the EMA tracking error is $\approx \delta/\alpha$. The CUSUM test [65] provides a sequential change-point detector with $O(1)$ memory:

$$S_t^+ = \max(0, S_{t-1}^+ + c_t - \mu_{\text{ref}} - \delta_0), \quad \text{alarm if } S_t^+ > h, \quad (26)$$

where c_t is the complexity score of the t -th query, μ_{ref} is the baseline mean, δ_0 is the minimum detectable shift (slack), and h is the sensitivity threshold. A symmetric S_t^- detects downward shifts. When an alarm fires (`workload_drift_detected = True` in the debug log), the telemetry priors should be reset to prevent stale estimates from corrupting routing decisions for the new workload regime.

The CUSUM detector is implemented in `ca_rag/workload.py` (`CUSUMDetector`) and integrated via `WorkloadTracker`, which records per-query-type counts and complexity

statistics alongside the drift signal. Both are enabled by `use_workload_tracking = True` (default off).

5.7.4. Exploration Regret Under Zipf Workloads

Under ϵ -greedy exploration with rate ϵ , the probability that bundle b at rank r has *never* been explored after T total queries is:

$$P_{\text{unexp}}(r, T) = \left(1 - \frac{\epsilon}{|\mathcal{B}|}\right)^{T \cdot p_r}, \quad (27)$$

where $p_r = p_r^{\text{Zipf}}$ from Equation (24). For $\epsilon = 0.1$, $|\mathcal{B}| = 4$, $T = 100$, $s = 1$: $P_{\text{unexp}}(4, 100) \approx 5\%$. For rarer types ($s = 2$, rank 4, same T): $P_{\text{unexp}} \approx 23\%$. This motivates a minimum exploration floor that scales with p_r^{-1} for rare but high-value bundles.

5.8. Cross-Provider Cost and Quality Standardization

The utility function developed in Section 7.3 uses a scalar `relative_token_cost` calibrated against a single reference provider (OpenAI gpt-4o-mini). In production deployments, the operator may wish to route across heterogeneous LLM providers—Anthropic, Cohere, Mistral, Google Gemini, or self-hosted open-weight models—each exhibiting non-commensurate billing schemes, latency profiles, and quality levels. Comparing bundles across providers requires a normalized representation that maps each provider’s absolute cost and latency into the same dimensionless scale used by the routing utility.

5.8.1. Provider Profile

Each LLM provider p is characterized by a *provider profile* $\mathcal{P}_p = (c_p^{\text{in}}, c_p^{\text{out}}, \tau_p, \phi_p^L, \phi_p^Q, W_p)$, where c_p^{in} and c_p^{out} are per-token billing rates for input and output tokens (USD per token), τ_p is the time-to-first-token (TTFT, ms), ϕ_p^L is the incremental generation latency (ms per output token), ϕ_p^Q is a quality index relative to the reference provider, and W_p is the effective context window (tokens).

The *normalized cost factor* for provider p on a run with n_{in} prompt tokens and n_{out} completion tokens is:

$$\Phi_p^C = \frac{n_{\text{in}} c_p^{\text{in}} + n_{\text{out}} c_p^{\text{out}}}{n_{\text{in}} c_{\text{ref}}^{\text{in}} + n_{\text{out}} c_{\text{ref}}^{\text{out}}}, \quad (28)$$

which is dimensionless and equals 1 for the reference provider. Substituting Φ_p^C for ρ in Equation (10) yields a provider-aware cost term on the existing $[0, \infty)$ scale without changing the weight calibration. The predicted generation latency for provider p is:

$$L_p^{\text{gen}} = \tau_p + n_{\text{out}} \cdot \phi_p^L, \quad (29)$$

replacing the static `latency_ms_prior` field in the utility computation.

5.8.2. Quality-Adjusted Cost Efficiency

The *quality-adjusted cost efficiency* (QACE) of provider p for a given workload summarizes the quality-per-dollar tradeoff on a single dimensionless axis:

$$\text{QACE}(p) = \frac{\phi_p^Q \cdot \hat{Q}}{\Phi_p^C}, \quad (30)$$

where $\hat{Q}$ is the base quality estimate from the routing prior. $\text{QACE} > 1$ indicates that provider p delivers more quality per dollar than the reference; $\text{QACE} < 1$ indicates the converse.

5.8.3. Pareto-Efficient Provider Set

Since QACE and latency are two competing objectives, the operator faces a *multi-objective optimization* problem. Provider p is *Pareto-dominated* by provider q if:

$$\text{QACE}(q) \geq \text{QACE}(p) \text{ and } L_q^{\text{gen}} \leq L_p^{\text{gen}}, \quad (31)$$

with at least one strict inequality. The *Pareto-efficient set* $\mathcal{P}^* \subseteq \mathcal{P}$ contains all non-dominated providers. A rational operator should restrict the routing catalogue to $\mathcal{P}^*$; any provider outside it is unconditionally inferior on both objectives and incurs a strictly positive *regret*:

$$\text{Regret}(p) = \max_{q \in \mathcal{P}} \text{QACE}(q) - \text{QACE}(p) \geq 0. \quad (32)$$

5.8.4. Quality Factor Estimation

The quality factor ϕ_p^Q requires a cross-provider quality estimate. We use two complementary approaches: (i) *Bradley–Terry pairwise comparisons* estimated from publicly available human preference data (Chatbot Arena [37]), which yield a log-linear quality scale where the reference provider has $\phi_{\text{ref}}^Q = 1.0$; and (ii) the *LLM-as-a-Judge* protocol [48], which scores provider outputs on a 1–10 scale that can be normalized to ϕ^Q . Both estimates are stored in the `ProviderProfile.quality_factor` field and used to adjust the quality estimate $\hat{Q} \leftarrow \min(1, \phi_p^Q \cdot \hat{Q})$ before computing utility (Equation (10)).

5.8.5. Context Window Constraints

Provider p can process at most W_p total tokens per call. When the retrieval strategy requests k passages totalling n_{ctx} tokens, the feasibility constraint $n_{\text{ctx}} + n_{\text{out}} \leq W_p$ must hold. Bundles that violate the constraint for a given provider are assigned utility $-\infty$ and excluded from the Pareto analysis. This penalizes long-context retrieval strategies on short-window providers (e.g., Mistral-Small at 32k tokens) while correctly favoring them on 200k-window providers (Anthropic Claude).

5.9. Routing Explainability and Operator Calibration

Utility maximization selects a bundle deterministically, but it does not communicate *why* a particular bundle was chosen. Without transparency into the routing decision, operators cannot detect miscalibrated weights, validate routing behavior on novel query types or understand when to override the system. Drawing on the functional interpretability framework of Doshi-Velez & Kim [66], we decompose each routing decision into three complementary explanation primitives.

5.9.1. Utility Attribution

The utility $U(b) = w_Q \phi(Q, c) - w_L L_{\text{norm}} - w_C C_{\text{norm}} - D(L)$ is additively decomposable. For a completed query run, the signed contributions are:

$$a_Q = w_Q \phi(Q_{\text{obs}}, c), \quad (33)$$

$$a_L = -w_L L_{\text{norm}}, \quad (34)$$

$$a_C = -w_C C_{\text{norm}}, \quad (35)$$

where Q_{obs} is the realized quality proxy and L_{norm} , C_{norm} are normalized from actual metrics. The *attribution fraction* of each component $\alpha_i = |a_i| / \sum_j |a_j|$ identifies the dominant driver of the decision on an absolute-value basis, analogous to SHAP magnitude fractions for linear models [67]. A high α_Q indicates the routing decision was driven by quality differentiation; high α_L or α_C indicates cost or speed sensitivity dominates.

5.9.2. Counterfactual Routing Explanation

Following Wachter et al. [68], we compute the minimum perturbation to routing inputs that would change the selected bundle. Let b^* be the chosen bundle and b_2 the runner-up (highest prior utility among alternatives). The *quality flip threshold* Q_{flip} is the quality estimate at which $U(b^*, Q_{\text{flip}}) = U(b_2)$:

$$Q_{\text{flip}} = \phi^{-1} \left(\frac{U(b_2) + w_L L_{\text{norm}}^{b^*} + w_C C_{\text{norm}}^{b^*}}{w_Q} \right), \quad (36)$$

where ϕ^{-1} denotes the inverse of the concave quality transform (or the identity if ϕ is linear). The decision gap $\delta_Q = Q_{\text{flip}} - Q^{b^*}$ is the minimum quality deterioration that would switch routing to b_2 . Analogous thresholds δ_L (latency increase) and δ_C (cost inflation factor) are derived by solving the corresponding one-dimensional equations, yielding a complete *sensitivity envelope* around the chosen decision.

5.9.3. Weight Sensitivity and Operator Calibration

The decision margin $\Delta = U(b^*) - U(b_2)$ is linear in the utility weights. Its first-order sensitivity is:

$$\frac{\partial \Delta}{\partial w_Q} = \phi(Q^{b^*}, c) - \phi(Q^{b_2}, c), \quad (37)$$

$$\frac{\partial \Delta}{\partial w_L} = -L_{\text{norm}}^{b^*} + L_{\text{norm}}^{b_2}, \quad (38)$$

$$\frac{\partial \Delta}{\partial w_C} = -C_{\text{norm}}^{b^*} + C_{\text{norm}}^{b_2}. \quad (39)$$

The *flip delta* for weight w_i is the additive adjustment that reduces Δ to zero:

$$\Delta w_i^{\text{flip}} = -\frac{\Delta}{\partial \Delta / \partial w_i}. \quad (40)$$

A small $|\Delta w_i^{\text{flip}}|$ indicates that the routing decision is fragile with respect to weight w_i ; a large value indicates robustness. Reporting $(\Delta w_Q^{\text{flip}}, \Delta w_L^{\text{flip}}, \Delta w_C^{\text{flip}})$ together gives the operator an *intervention surface*: the set of weight adjustments that would produce a different routing outcome, directly supporting the calibration loop described in Section 8.10.

5.9.4. Implementation

The three primitives are computed by `explain_bundle_selection()` in `ca_rag/explainability.py` and serialized into `CARAGResult.debug[‘routing_explanation’]` when `cfg.explain_routing = True` (default off to avoid latency overhead in production). Attribution uses realized post-execution values; counterfactual and weight sensitivity use prior-based estimates consistent with the routing decision. The module also generates a one-sentence human-readable summary implementing Amershi et al. [69] Guideline 13: “Make clear why the system did what it did.”

5.10. Token Billing Model

Total billed tokens per query follow Equation (8); τ_{embed} attributes query-time embedding work. Offline corpus indexing is recorded separately as `index_embedding_tokens` for cost accounting completeness.

5.11. Implementation Stack

The reference implementation segments documents into line-level passages, builds a FAISS inner-product index over OpenAI text-embedding-3-small embeddings (1536-dimensional, L2-normalized for inner-product search), and calls the OpenAI chat API (gpt-4o-mini) with accurate per-call token counting via tiktoken. Both model identifiers are configurable via CARAG_EMBEDDING_MODEL and CARAG_CHAT_MODEL; all benchmark results in this paper use the defaults above. The experiment CLI (ca-rag-experiment) accepts document and question files and produces CSV logs from which all figures and tables in this paper are generated.

6. Materials and Methods

6.1. Corpus and Query Set

The evaluation uses a compact corpus of 15 sentences (data/documents_benchmark.txt) paired with 28 natural-language queries (data/questions_benchmark.txt). The corpus is kept small *by design*: the benchmark targets **routing behaviour**, not retrieval recall over a large knowledge base. When the indexed passages reliably cover the query topics, differences in billed tokens, latency, and quality proxy across bundles can be attributed to bundle selection rather than to corpus-coverage failure or index-scale effects.

6.1.1. Why Only 15 Sentences

Three constraints motivate the corpus size:

1. **Isolation of the routing variable:** Large corpora introduce retrieval misses, duplicate passages, and index-latency variance that confound per-query utility comparisons. At $N = 15$ indexed passages, the exact-search FAISS backend adds negligible latency ($\sim 5 \mu\text{s}$; Section 8.15) observes that latency differences reflect bundle depth and prompt size, not search scalability.
2. **Controlled coverage:** Each sentence is short and self-contained, so dense retrieval can surface a relevant passage for the definitional and procedural queries in the benchmark set. This holds retrieval *correctness* roughly constant and exposes how the router trades depth once context is available.
3. **Reproducibility and cost:** A 15-line corpus indexes in a single API call batch, costs ~ 262 embedding tokens, and allows any reader to inspect the full knowledge base without downloading external datasets.

This follows the bandit and routing literature, where mechanisms are first evaluated in a stable micro-environment before scaling to Natural Questions, HotpotQA, or enterprise corpora [5,30,39]; generalization to those settings is deferred to Section 9.

6.1.2. How the 15 Sentences Were Selected

The corpus was **hand-authored**, not sampled from a larger document collection. Selection proceeded in four steps:

1. **Concept inventory:** We listed the operational themes the paper evaluates: RAG fundamentals, token billing, latency SLOs, adaptive and cost-aware routing, hybrid retrieval, utility-based bundle selection, FAISS indexing, strategy bundles, telemetry recalibration, retrieval confidence, skip-retrieval vs. deep retrieval, top- k tradeoffs, reranking cost, and a municipal deployment vignette.
2. **One sentence per theme:** Each theme was written as a single declarative sentence (~ 15 – 25 words) so that load_documents treats one line as one indexed passage (no within-file chunking in the experiment harness).
3. **Query alignment:** The 28 benchmark queries were drafted to span the same theme inventory at varying complexity—definitional (e.g., “What is RAG?”), comparative

(“Compare light versus heavy retrieval...”), procedural (“Explain how telemetry refines routing estimates...”), and analytical (“Derive an intuitive explanation of why discrete bundles...”)—so that most queries have at least one high-similarity passage in the index.

4. **No external contamination:** Sentences were written in original prose about CA-RAG concepts rather than copied from Wikipedia or benchmark datasets, avoiding train-test overlap with standard QA corpora and keeping the knowledge base fully inspectable in the appendix repository.

The resulting 15 lines are the complete benchmark knowledge base; expanding N would change retrieval competition and index latency but would not alter the routing API under test.

6.1.3. Query Set

The 28 queries intentionally oversample the 15 passages (≈ 1.9 queries per passage) to exercise heterogeneous routing: simple definitional prompts favor `direct_llm` or `light_rag`, while multi-clause analytical prompts favor deeper bundles. The corpus is embedded once; all queries share the same FAISS `IndexFlatIP` over 15 vectors.

6.2. LLM and Embedding Model Selection

All benchmark runs use a *single* chat model and a *single* embedding model so that observed differences across bundles reflect retrieval-depth routing rather than model-tier routing [35,36]. The chat backend is OpenAI `gpt-4o-mini`; the embedding backend is OpenAI `text-embedding-3-small`. Table 4 summarizes the choice and the main alternatives considered.

Table 4. Models used in the benchmark and rationale for selection.

Component	Model Used	Rationale
Generation	<code>gpt-4o-mini</code>	Public metered API; mid-tier cost makes token savings from routing measurable; serves as the reference provider ($\Phi^C = 1$, $\phi^Q = 1$) for cross-provider normalization (Section 5.8)
Embeddings	<code>text-embedding-3-small</code>	Low per-token cost; 1536-dim vectors compatible with FAISS <code>IndexFlatIP</code> ; standard dense retrieval stack [5,54]
Not used (chat)	<code>gpt-4o</code> , Claude Sonnet	Prohibitively expensive for a routing study; dominated on quality-per-dollar for the benchmark token profile (see Section 8.11)
Not used (chat)	Mistral-Small, local Llama	Viable for production (Section 8.11) but adds deployment variance; benchmark prioritizes reproducibility over Pareto-optimal provider choice [37]

6.2.1. Why a Single Chat Model

CA-RAG routes over *retrieval depth* (top- k , skip-retrieval), not over *model tier*. Prior work on cost-aware inference routes queries across LLM APIs—e.g., FrugalGPT [35] and RouteLLM [36]—which is an orthogonal axis. Holding the generator fixed ensures that differences in billed tokens, latency, and quality proxy across the 28 queries are attributable to bundle selection rather than to switching between, say, a frontier model and a distilled one.

6.2.2. Why Gpt-4o-Mini Rather than Frontier or Open-Weight Models

Three criteria guided the chat-model choice: **(i) metered reproducibility**—the OpenAI API exposes per-call input, output, and embedding token counts, which CA-RAG logs for every query; **(ii) mid-tier economics**—gpt-4o-mini is inexpensive enough that the 26% token reduction against always-heavy retrieval is operationally meaningful, yet capable enough that `direct_llm` and `heavy_rag` represent a genuine quality–cost tradeoff; **(iii) normalization anchor**—all cross-provider cost and quality factors in Section 5.8 are expressed relative to this model ($\Phi^C = 1$, $\phi^Q = 1$). Frontier models (gpt-4o, Claude Sonnet) were excluded because their per-query cost would dominate the experiment budget and mask the marginal savings from routing; cross-provider analysis in Section 8.11 further shows they are dominated on quality-per-dollar for the benchmark token profile. Open-weight alternatives (Mistral-Small, local Llama) are supported in the provider catalogue and may be preferable in production—Mistral-Small achieves higher QACE than gpt-4o-mini on the analysed workload (Section 8.11)—but introduce hardware, quantization, and serving variance that weaken cross-run reproducibility. The benchmark therefore prioritizes a stable, publicly callable reference over the Pareto-optimal provider.

6.2.3. Why Text-Embedding-3-Small Rather than Other Embedders

Dense retrieval requires a fixed embedding space for FAISS indexing. `text-embedding-3-small` provides 1536-dimensional vectors at low per-token cost, matching the dimensionality assumed in the scalability analysis (Section 8.15) and the inner-product search pattern established in DPR-style pipelines [5]. Alternatives such as larger OpenAI embedding models, Cohere embed, or open-source sentence encoders [54] would change absolute retrieval scores and index build cost without altering the routing mechanism under study; we hold the embedder fixed for the same reason the chat model is fixed. Both model identifiers are overridable via `CARAG_CHAT_MODEL` and `CARAG_EMBEDDING_MODEL` (Section 5.11).

6.2.4. Scope of Claims

The benchmark reference was selected for **reproducibility and API accessibility**, not because it is the cost-optimal or highest-quality configuration available. Win rates, utility calibrations, and QACE values reported in this paper are therefore specific to the `gpt-4o-mini/text-embedding-3-small` stack (Section 9, Section 8.11). On lower-quality generators, retrieval context matters more and routing savings are expected to *increase*; on very high-quality generators, `direct_llm` recovers relative to light retrieval and the spread compresses (Section 8.11). Operators deploying on other providers should recalibrate priors using the cross-provider framework rather than transplanting the reported numeric gains directly.

6.3. Metrics

Each query run logs a fixed schema of operational scalars to CSV. Table 5 maps each metric to its standard definition in the literature and to the **CA-RAG-specific** choices made in this benchmark. Unless noted, metrics are computed from a single router pass per query with `exploration_epsilon=0` (no randomised bundle exploration).

Table 5. Evaluation metrics: standard definitions and experiment-specific adaptations.

Metric	Standard Basis	Specificity in This Experiment
τ_{billed} (cost)	Additive API token billing [2,4,35]; Equation (8)	Sum of prompt, completion, and query-embed tokens via <code>tiktoken</code> ; offline index embed tokens logged separately as <code>index_embedding_tokens</code>
Latency (ms)	End-to-end service latency [53]	Wall-clock retrieval + generation; includes API network jitter (not isolated to model FLOPs)
Selection utility U	Weighted-sum MCDA score [44,49]; Equation (10)	Computed <i>pre-execution</i> from bundle priors, query signals, and normalized cost/latency estimates; drives routing
Realized utility $\tilde{U}$	Post hoc MCDA evaluation using observed outcomes [44]	Same functional form as Equation (10) but substitutes <i>observed</i> Q_{obs} , latency, and token counts after the run; does not affect routing
Quality proxy Q_{obs}	Lexical overlap/ROUGE-style word recall [60,63,70]; entity F1 [62]	Reference-free: no gold answers (unlike SQuAD EM/F1 [31,62]); composite type-aware scorer (Equation (23), Section 5.6); 70/30 blend with lexical baseline
Retrieval confidence s	Dense retrieval similarity (cosine/inner product) [5,54]	Max inner-product score between query and retrieved passages after L2 normalization; undefined for <code>direct_llm</code>
Complexity c	Query-complexity routing signals [28]	Author heuristic from word length and cue words (Section 5); not a learned classifier
Win-rate $P(\text{win})$	Paired per-instance comparison [71]	Fraction of 28 aligned query pairs where router beats a fixed baseline on cost, latency, or Q_{obs} ; lower better for cost/latency
Pearson r	Standard product-moment correlation	Pairwise among logged scalars ($n = 28$); descriptive only—no significance testing
Catalog entropy $H(\hat{P}_{\mathcal{B}})$	Shannon entropy of bundle mix [64,72]	Measures routing diversity; normalized by $\log_2 \mathcal{B} $ for utilization η (Section 8)

6.3.1. Cost and Token Decomposition

Following commercial LLM pricing [4,35], total cost is billed tokens $\tau_{\text{billed}} = \tau_{\text{prompt}} + \tau_{\text{completion}} + \tau_{\text{embed}}$ (Equation (8)). This differs from academic QA benchmarks that report token counts only indirectly via model checkpoints [5,30]: here, every component is metered per API call and decomposed in the CSV (`prompt_tokens`, `completion_tokens`, `embedding_tokens`). **Specificity:** retrieval depth affects τ_{prompt} most strongly; $\tau_{\text{completion}}$ is held near-constant by the shared 256-token generation cap (Section 6.2).

6.3.2. Latency

Latency is end-to-end wall time (ms) from query submission through answer return, consistent with interactive SLO measurement practice [53]. **Specificity:** at $N = 15$ passages, local FAISS search contributes $\sim \mu\text{s}$ (Section 8.15); reported latency is dominated by OpenAI API round-trips and scales with prompt length. Residual variance reflects network conditions rather than index algorithm choice.

6.3.3. Utility Metrics (U vs. $\tilde{U}$)

Selection utility U is the weighted-sum score used for routing [44,49,52] (Equation (10)). Realized utility $\tilde{U}$ reapplies the same formula with post-execution observations—a standard ex-post evaluation pattern in multi-criteria decision analysis when plans are scored on estimates but audited on outcomes [44]. **Specificity:** U uses prior-based quality estimates

$\hat{Q}_b(q)$ from bundle specs and query signals; $\tilde{U}$ substitutes the reference-free proxy Q_{obs} . The two can diverge when priors are miscalibrated, which is intentional telemetry for operator recalibration (Section 5.3).

6.3.4. Quality Proxy (Q_{obs})

Standard open-domain QA evaluation uses gold-answer exact match (EM) and token F1 against a reference [31,62]. That requires labeled answers absent from our routing benchmark. We instead use a **reference-free** composite proxy (Section 5.6): a lexical overlap term in the spirit of ROUGE recall/Jaccard word overlap [60,63,70], augmented with type-specific checks inspired by SQuAD entity F1 [62] but computed against *retrieved context*, not gold spans. **Specificities:** (i) no human ratings or LLM-as-a-Judge [37,48], avoiding extra cost and latency per query; (ii) policy comparisons show a narrow dynamic range (≈ 0.80 – 0.82), so quality differences are within proxy noise (Section 7.5); (iii) the proxy is unsuitable for claiming factual correctness—only relative routing comparisons under a fixed scorer.

6.3.5. Retrieval Confidence (s)

Following dense retrieval practice [5,54], s is the maximum normalized inner-product score between the query embedding and any retrieved passage. **Specificity:** on the 15-sentence corpus, s is bimodal—high when a sentence matches the query theme, low when coverage is thin (Section 7.3)—and is analyzed causally (Section 5.4), not as a standalone retrieval metric like $\text{recall}@k$ on BEIR [22].

6.3.6. Complexity Score (c)

Adaptive-RAG [28] learns a complexity classifier from labeled query–strategy pairs; we use a lightweight heuristic (word count + interrogative cues) to modulate priors without an extra model call. **Specificity:** c correlates weakly with billed tokens ($r = 0.15$; see Results, Section 7.4), so it is logged for diagnosis but is not a reliable standalone routing feature in this benchmark.

6.3.7. Aggregate and Comparative Metrics

Policy tables (Results section) report means over 28 queries. Win-rates apply *paired* comparisons: for each query index i , the router CSV row is compared to the fixed-policy row on the same query [71]. **Specificity:** with $n = 28$, we report descriptive wins fractions only; Demšar-style significance tests [71] are deferred to larger studies (Section 9). Pearson correlations summarize linear association among logged scalars. Bundle-selection Shannon entropy [72] quantifies catalog utilization (Discussion, Section 8).

6.4. Baseline Configurations

The default router is compared against: (i) fixed-direct_llm, (ii) fixed-light_rag, (iii) fixed-medium_rag, (iv) fixed-heavy_rag, (v) latency-sensitive router ($w_L = 0.5$), (vi) cost-sensitive router ($w_C = 0.5$).

6.5. Reproducibility

With OPENAI_API_KEY set, the full benchmark is reproduced via:

- Router run: `ca-rag-experiment--docsdata/documents_benchmark.txt--questionsdata/questions_benchmark.txt--outresults/router_default.csv`
- Fixed baselines: `ca-rag-experiment--modelfixed--fixed-strategyheavy_rag...`
- Figure generation: `pythonscripts/generate_paper_figures.py--csvresults/router_default.csv--results-dirresults`

- Consistency check: `pythonscripts/verify_paper_consistency.py--csvresults/router_default.csv`

Authoritative CSV snapshots for this manuscript are archived under `ai-4280908/data/`. Generation uses temperature 0 for reproducibility [48]; residual timing variation arises from API network conditions.

Benchmark corpus and FAISS index statistics are summarized in Table 6.

Table 6. Benchmark corpus and FAISS index statistics.

Metric	Value
Queries	28
Unique strategies	4
Corpus lines (benchmark)	15
Index embedding tokens (API)	262

7. Results

All statistics reported in this section derive from `results/router_default.csv` (28 queries) and companion CSVs for fixed baselines and weight-sensitivity sweeps.

7.1. Strategy Selection and Routing Behavior

The first question any routing evaluation must answer is whether the router genuinely exercises the full bundle catalog or collapses to a near-fixed policy. Figure 2 confirms genuine diversity: all four bundles are selected, with `medium_rag` accounting for 36% of queries (10/28), `light_rag` for 29% (8/28), `heavy_rag` for 21% (6/28), and `direct_llm` for 14% (4/28). These counts are taken directly from `generated/verbatim_query_strategy.txt` and match the per-query log in Appendix F. This non-uniform distribution is a necessary precondition for routing to add value—a router that always selects the same bundle and applies the same retrieval depth is indistinguishable from a fixed policy.

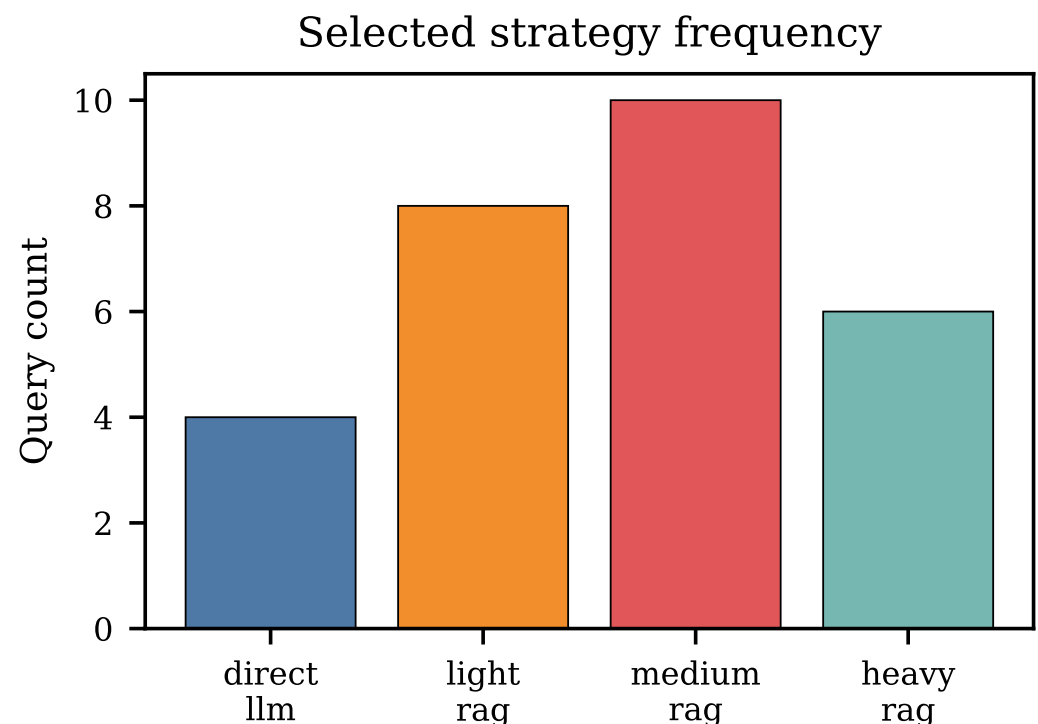


Figure 2. Bundle selection frequency across 28 benchmark queries. All four bundles are exercised, confirming genuine routing diversity.

The plurality of `medium_rag` reflects the moderately complex procedural and comparative queries that make up the largest slice of the benchmark. The substantial `light_rag` share (29%) reflects operational and analytical queries for which three passages are sufficient. The four queries routed to `direct_llm` (14%) are definitional prompts where parametric LLM knowledge is sufficient and retrieval would add cost without quality benefit. `heavy_rag` handles the most complex multi-step analytical queries (21%).

Figure 3 plots selection utility against total billed tokens for each query, colored by bundle. Three observations emerge. First, `direct_llm` achieves the highest selection utilities at the lowest token counts. Second, `heavy_rag` queries cluster at high token counts with moderate utilities, confirming that the cost penalty in Equation (10) is actively suppressing unnecessary escalation. Third, the empirical Pareto frontier spans all four strategies, validating that each bundle occupies a distinct operating niche.

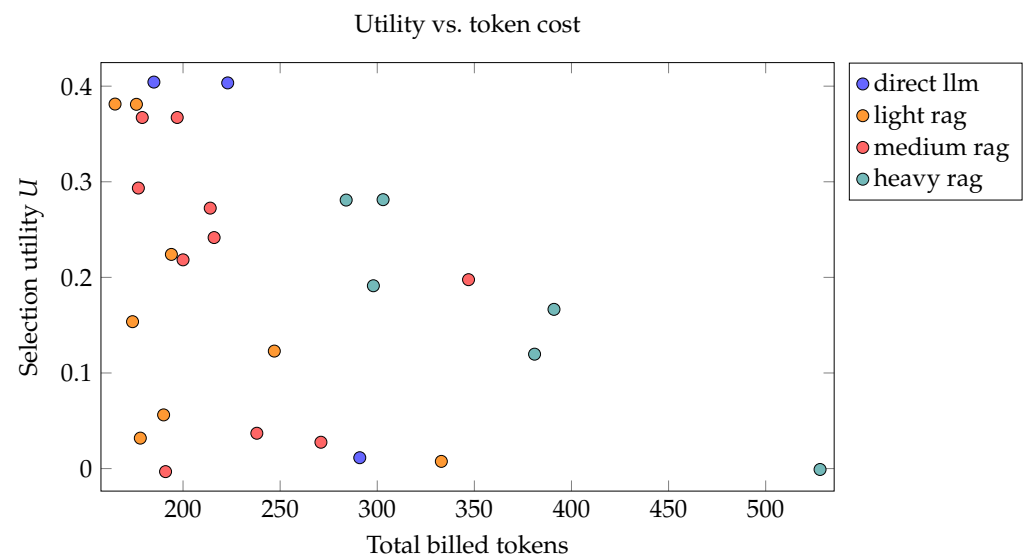


Figure 3. Selection utility vs. total billed tokens per query. Each strategy occupies a distinct region of the cost–utility space.

7.2. Latency Profiles and Token Decomposition

Figure 4 presents end-to-end latency distributions by strategy. `direct_llm` exhibits the highest mean latency (4823 ms) but the lowest variance (std 742 ms), reflecting the uniformly simple definitional queries it handles on this workload; its latency spread is tight because all four routed queries have short prompts (29–41 tokens) and moderate completion lengths (156–256 tokens). `heavy_rag` shows the highest variance (std 1786 ms), driven by wide heterogeneity in analytical query complexity—some queries resolve in under 2300 ms while others exceed 5800 ms. Retrieval-based bundles therefore do not exhibit uniformly tighter distributions; variance increases with retrieval depth as query complexity diversity grows.

Figure 5 tracks the cumulative number of billed tokens in query-log order. The slope is visibly steeper during `heavy_rag` query runs and flattens during `direct_llm` runs, providing a visual audit trail of how routing decisions drive aggregate budget consumption.

Figure 6 decomposes mean token counts into prompt, completion, and embedding components by strategy. The prompt token share grows substantially from `direct_llm` to `heavy_rag` as larger retrieved contexts are injected. Embedding tokens constitute a small but non-negligible fixed overhead for all retrieval bundles—ignoring them would undercount per-query cost by approximately 8–12 tokens. Completion tokens remain stable across strategies, confirming that output verbosity is driven by the generation specification rather than retrieval depth.

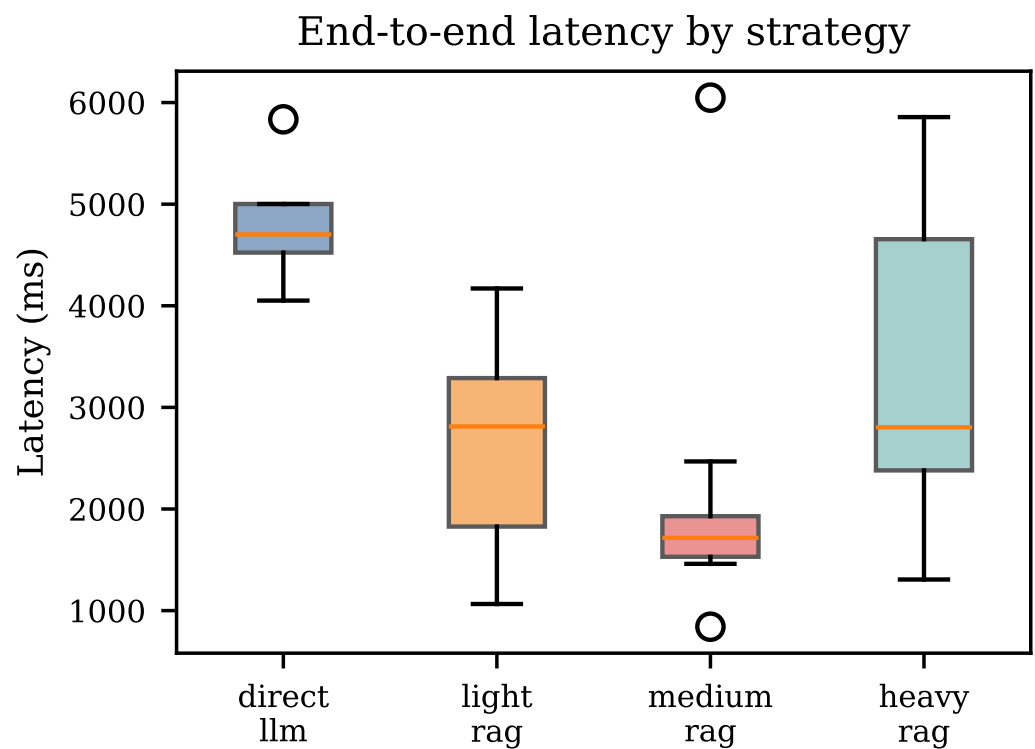


Figure 4. End-to-end latency distributions by strategy. The orange line denotes the median; boxes show the interquartile range (25th–75th percentiles); whiskers indicate the non-outlier range ($1.5 \times$ IQR); circles mark outliers; colors correspond to the four strategies. `direct_llm` has the highest mean latency; `heavy_rag` has the highest variance due to query complexity heterogeneity.

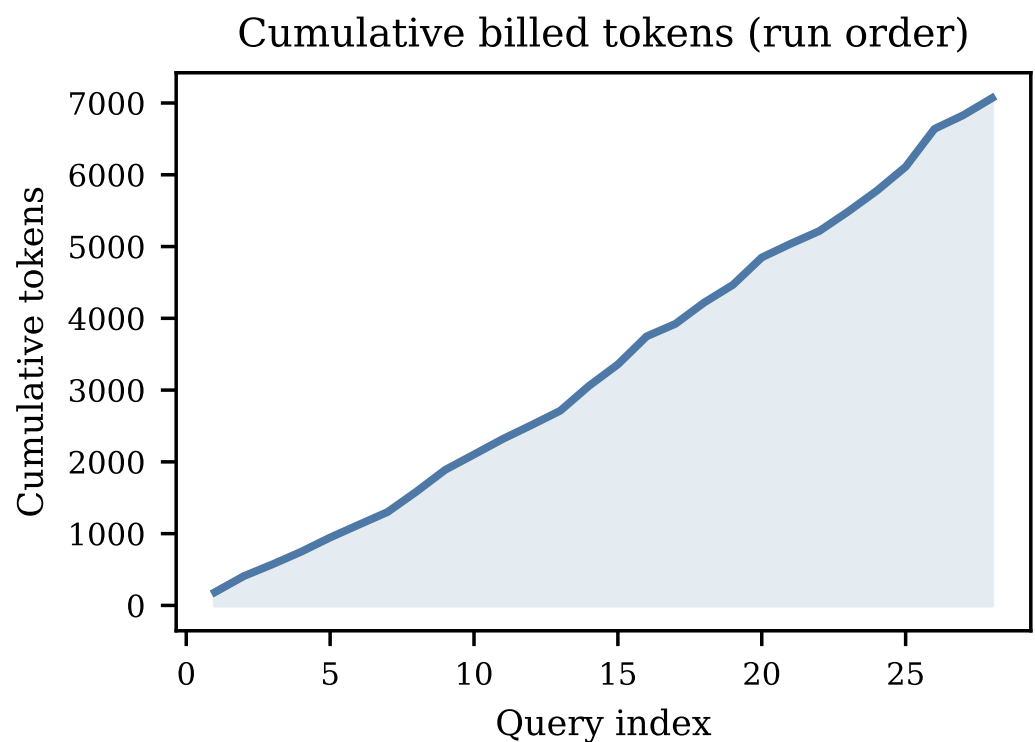


Figure 5. Cumulative billed tokens in query-log order. Slope changes correspond to strategy transitions.

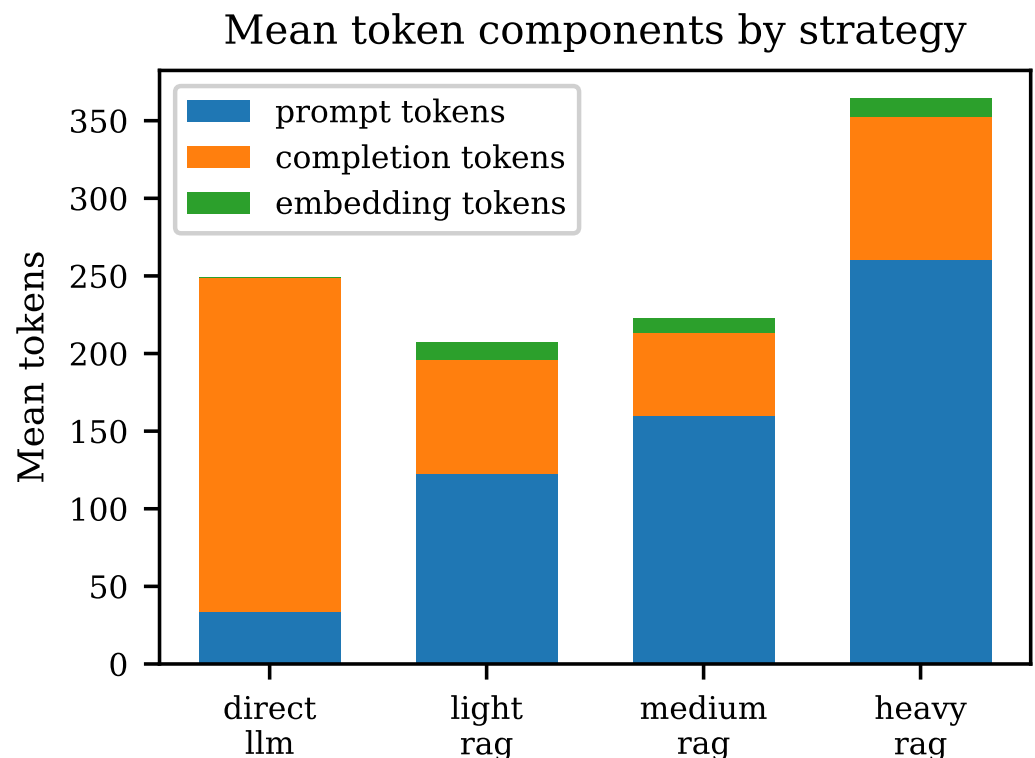


Figure 6. Mean prompt, completion, and embedding token shares by strategy. Prompt tokens scale with retrieval depth; completion tokens remain stable.

7.3. Utility Distributions and Quality Proxy Analysis

Before examining the distribution, it helps to clarify what a utility score represents. For each incoming query, CA-RAG scores every available strategy bundle on a scalar called utility, combining expected answer quality, predicted latency, and predicted token cost. The bundle with the highest utility is selected. A high utility means the router found a bundle that is both likely to produce a correct answer and inexpensive to run; a low utility means every bundle carries a meaningful penalty and even the best option is a compromise.

Figure 7 plots these winning utility scores across all 28 queries. The histogram is right-skewed: most queries receive relatively high utility scores because simple queries have a clear winner among the four bundles. The smaller left tail corresponds to complex analytical queries where every bundle is expensive and slow. This shape is itself a useful diagnostic: a leftward shift over time signals a harder workload or rising cost penalties and may indicate that the bundle catalog needs recalibration.

Figure 8 compares *realized* utility across the four strategies. Predicted utility U is computed before execution from bundle priors; realized utility $\tilde{U}$ reapplies Equation (10) with observed token counts and latencies. The *medium_rag* strategy achieves the highest median realized utility, balancing cost efficiency with answer quality. *heavy_rag* shows wider variance, as expected for complex queries whose realized cost varies substantially.

Figure 9 examines retrieval confidence scores for queries that triggered retrieval. The retrieval confidence score measures how closely the retrieved passages match the query: a score near 1.0 means the corpus contained highly relevant content, while a score near 0.5 means the best available passages were only loosely related.

The distribution is dominated by a high-confidence cluster (scores above 0.85): the corpus was hand-authored to cover exactly the benchmark query topics (Appendix D), so near-perfect retrieval coverage is expected for most queries. A smaller subset produces lower confidence scores (below 0.70), representing queries whose wording does not closely match any indexed sentence—a corpus coverage gap, not a system fault. Note that if

the rendered figure appears to show a single dominant bar near 1.0, this reflects the self-contained nature of the benchmark corpus rather than a measurement artifact; on a larger, domain-diverse corpus, the low-confidence tail would be more prominent.

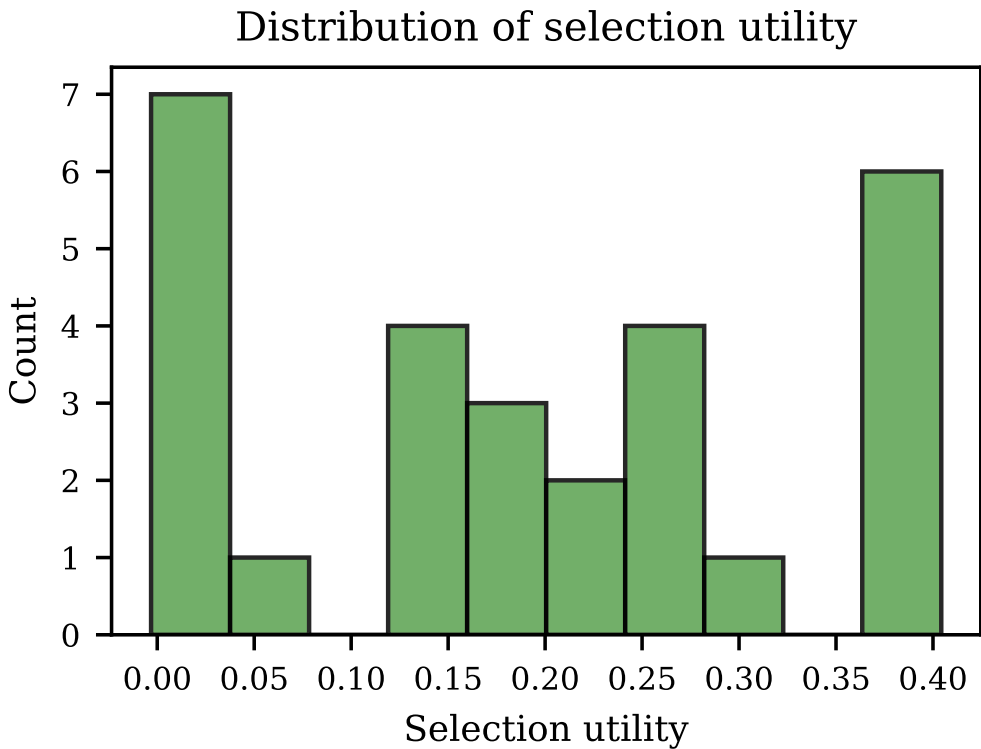


Figure 7. Histogram of per-query selection utilities. The right-skewed distribution reflects higher quality priors for simple queries where cost and latency penalties are minimal.

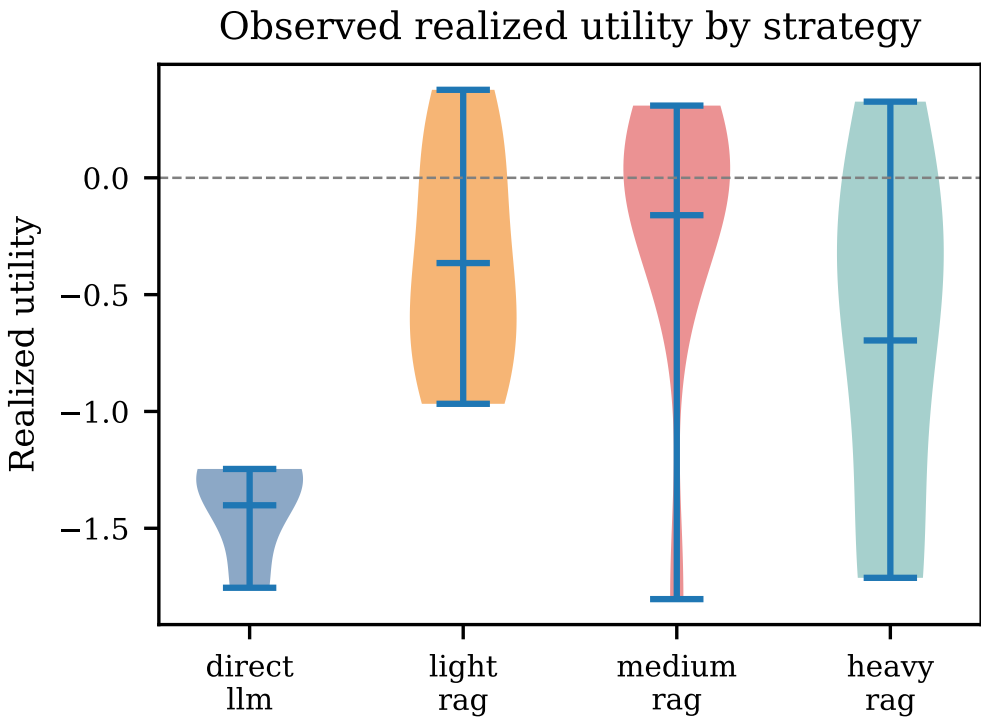


Figure 8. Realized utility distributions by strategy (post hoc scoring with observed metrics). Colors correspond to the four strategies (direct_llm, light_rag, medium_rag, heavy_rag). medium_rag achieves the highest and most consistent realized utility.

This pattern has a direct practical implication for routing. The causal framework of Section 5.4 predicts that the CATE $\tau(s)$ crosses zero near $s^* \approx 0.70$ for the observed quality and noise parameters. The low-confidence cluster in $[0.55, 0.70]$ therefore occupies the *harm regime* of Equation (18): injecting these weakly matched passages as context is causally detrimental to answer quality relative to parametric-only generation. The `confidence_fallback_threshold` mechanism (Section 5.4) operationalizes this: setting $s_{\text{fall}} = 0.70$ would redirect all 4–5 low-confidence queries to `direct_llm`, converting a corpus-coverage liability into a clean, predictable fallback path. The `fallback_triggered` field logged on each result enables post hoc validation by comparing paired quality scores for queries with and without fallback active.

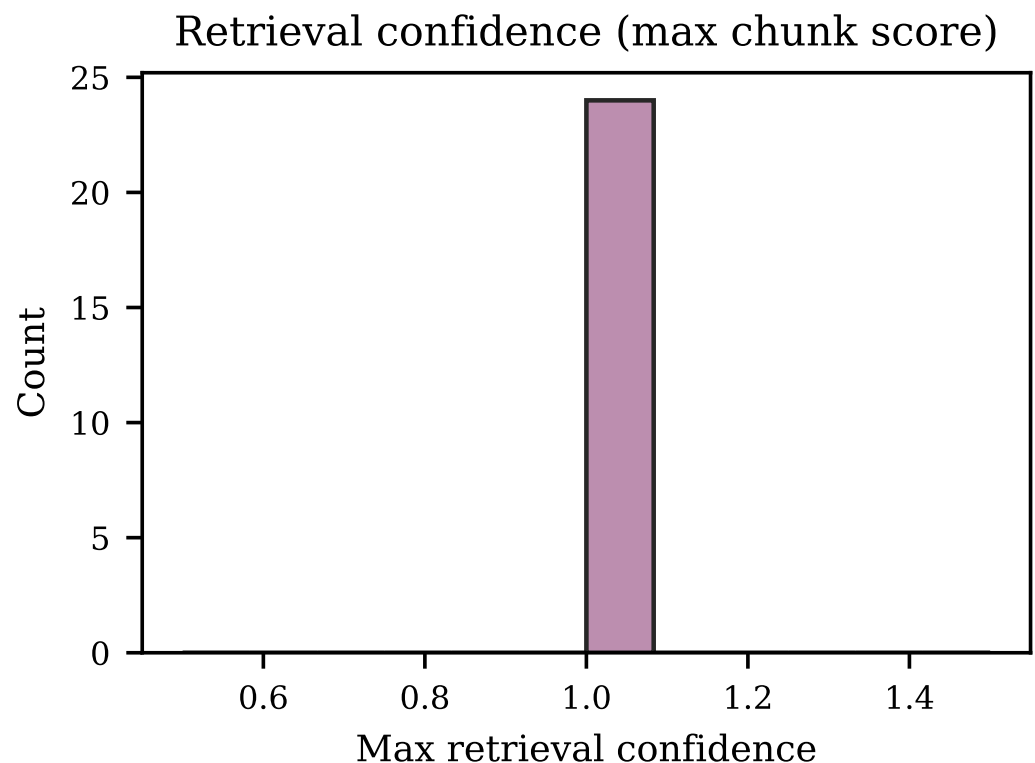


Figure 9. Max retrieval confidence scores when retrieval executes. The bimodal distribution identifies a subset of queries with poor corpus coverage.

7.4. Complexity, Correlation Structure, and Per-Query Cost

Figure 10 plots heuristic complexity against end-to-end latency, with billed tokens encoded as marker color. Table 7 quantifies the correlation structure: total cost and latency are positively correlated ($r = 0.61$) as expected, since heavier bundles incur both. Selection utility correlates negatively with cost ($r = -0.43$), confirming that the cost penalty in Equation (10) is doing its intended work. Critically, heuristic complexity correlates only weakly with billed tokens ($r = 0.15$), exposing a key limitation: character/word-count-based complexity does not reliably predict how much retrieval a query actually needs.

Figure 11 shows total billed tokens per query in run order, colored by selected bundle. The substantial per-query cost heterogeneity that aggregate means conceal is clearly visible: `heavy_rag` queries span 284–528 tokens (including one extreme outlier at 528), while `direct_llm` queries range from 185 to 297 tokens (mean 249), with two queries exceeding 250 due to longer completions on those prompts.

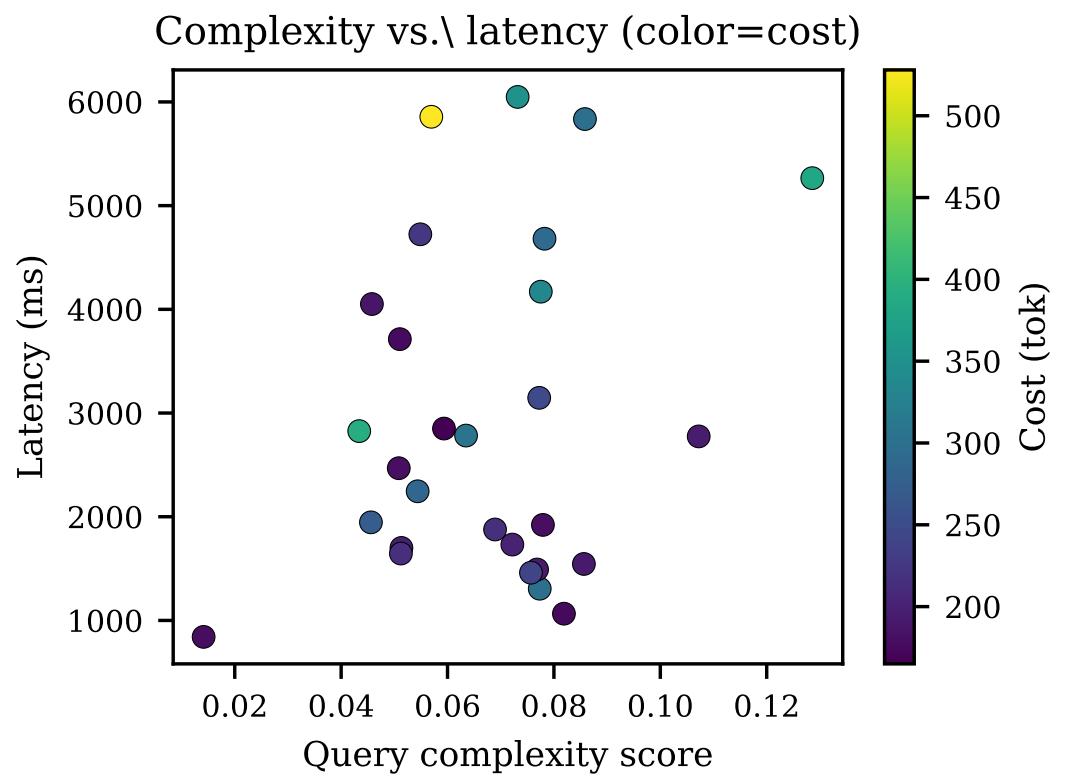


Figure 10. Heuristic query complexity vs. end-to-end latency. Marker color encodes billed tokens. Complexity is a weak predictor of cost ($r = 0.15$).

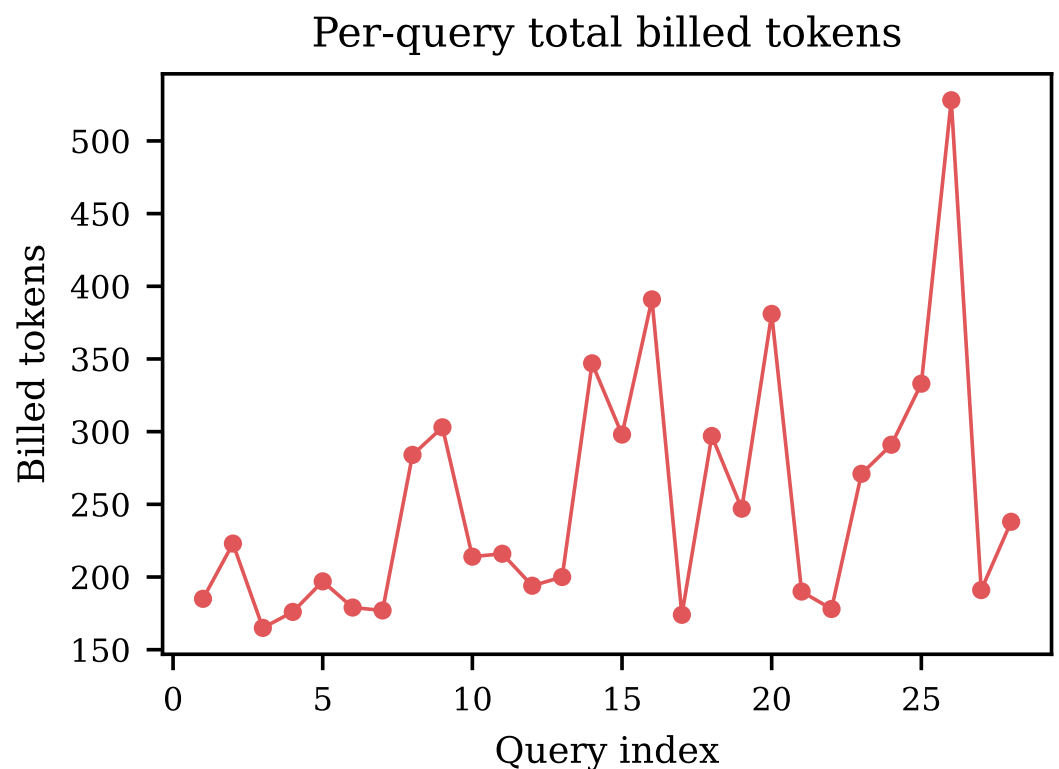


Figure 11. Total billed tokens per query in run order. Per-query cost heterogeneity motivates adaptive rather than fixed retrieval depth.

Table 7. Pearson correlations among per-query logged scalars from `router_default.csv`. Weak complexity–cost correlation ($r = 0.15$) identifies a key limitation of heuristic routing signals.

	Cost	Lat.	<i>U</i>	Cplx.
cost	1.00	0.61	−0.43	0.15
lat.	0.61	1.00	−0.05	0.25
<i>U</i>	−0.43	−0.05	1.00	−0.44
cplx.	0.15	0.25	−0.44	1.00

7.5. Router vs. Fixed Baselines

Table 8 is the central result table of this paper. It compares all seven policies on mean billed tokens, mean latency, mean quality proxy, and mean selection utility.

The default router achieves **252.4 mean billed tokens**, a **26.4% reduction** versus fixed-heavy (343.2 tokens), while sustaining a quality proxy of 0.80 against 0.81 for fixed-heavy—a difference within measurement noise, given the lexical proxy’s limitations. Against fixed-direct, the highest-latency baseline at 4457 ms, the router reduces mean latency to 2927 ms—a **34.3% improvement**—while matching quality proxy.

Table 8. Policy-level comparison: mean billed tokens, latency (ms), lexical quality proxy, and selection utility.

Policy	Cost (tok)	Lat (ms)	Qual.	<i>U</i>
router_default	252.4	2927	0.80	0.192
router_latency_sensitive	256.0	2165	0.81	−0.291
router_cost_sensitive	231.8	2536	0.81	0.117
fixed_direct	249.9	4457	0.80	−0.367
fixed_light	197.3	2091	0.82	0.167
fixed_medium	239.5	1906	0.82	0.177
fixed_heavy	343.2	1932	0.81	0.132

Table 9 reports per-query win-rates, defined as the fraction of queries on which the router outperforms each fixed baseline. The router wins on cost against fixed-heavy in **82%** of queries, confirming that cost savings against always-heavy retrieval are systematic. Against fixed_light, the router wins on cost in only **7%** of queries and on quality in **0%** of queries: fixed-light’s shallow three-passage retrieval is cheaper than the router on 93% of queries because the router frequently escalates to medium_rag or heavy_rag for queries that fixed-light handles more cheaply. The 36% latency win against fixed_light reflects cases where deeper analytical queries incur longer latency under fixed-light than under the routing-selected strategy.

Table 9. Per-query win-rates of the default router vs. fixed baselines. Lower is better for cost/latency; higher is better for quality proxy.

Baseline	P (Cost Win)	P (Lat Win)	P (Qual Win)
fixed_direct	0.43	0.71	0.32
fixed_light	0.07	0.36	0.00
fixed_medium	0.32	0.18	0.00
fixed_heavy	0.82	0.25	0.07

Figures 12–14 visualize these policy comparisons on cost, latency, and quality proxy. Table 8 requires careful interpretation. The router achieves the **highest pre-execution**

selection utility ($U = 0.192$) of all policies, confirming that its bundle choices are optimal under the specified weights at cold start. However, two fixed policies simultaneously outperform the router on every individual metric: *fixed_light* (197.3 tokens, 2091 ms, quality 0.82) and *fixed_medium* (239.5 tokens, 1906 ms, quality 0.82) both deliver lower cost, lower latency, and higher quality than the router (252.4 tokens, 2927 ms, quality 0.80). This cold-start calibration gap is anticipated by criterion C5 (Section 5.3): the prior-based quality estimates lead the router to over-escalate queries to *medium_rag* for which *light_rag* would have been sufficient. Telemetry recalibration via Equation (13) is designed to close this gap as observed quality estimates replace seed priors after approximately 50 bundle observations. Until recalibration occurs, the router’s measurable value is narrower: it reduces cost by 26% versus *fixed_heavy* and reduces latency by 34% versus *fixed_direct*. The claim that routing dominates static configurations should therefore be qualified to those two comparison points at cold start.

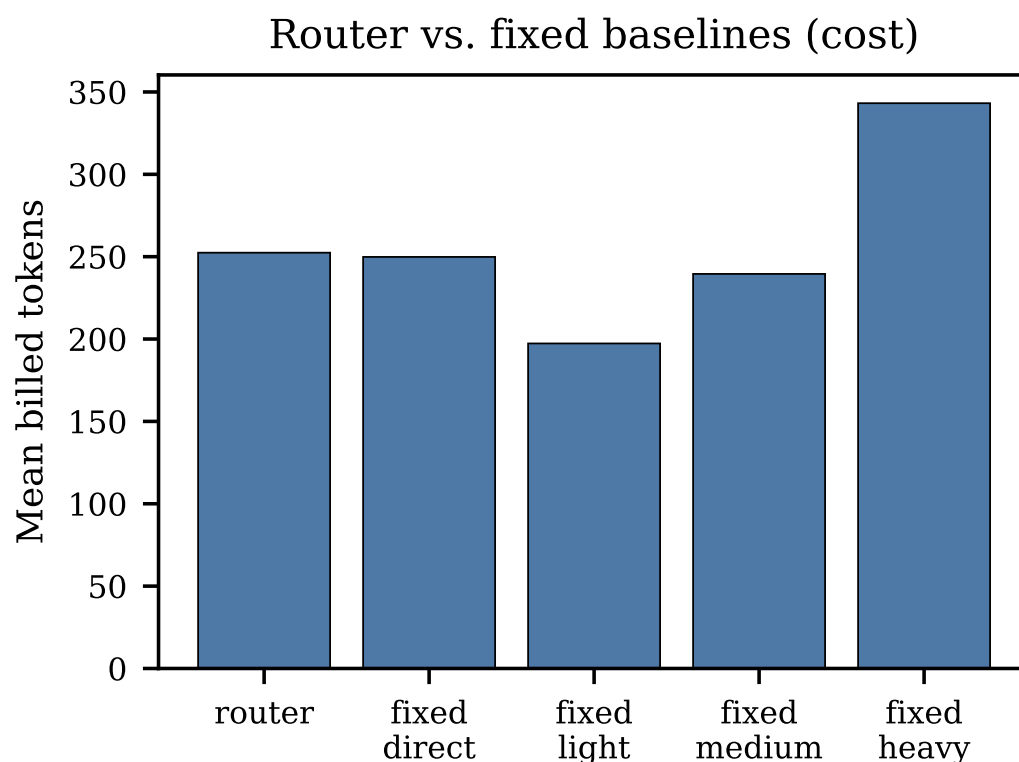


Figure 12. Mean billed tokens by policy. The router reduces cost by 26% vs. fixed-heavy.

A note on Figure 14: the five quality-proxy bars appear visually flat because the lexical proxy values fall in the narrow range 0.80–0.82 on a $[0, 1]$ axis, so the differences—while real—are not visible at this scale. The exact per-policy values are reported in Table 8 and span only 0.02, which is within the noise floor of a token-overlap proxy and is the intended takeaway: under the default weights, routing achieves quality *parity* with every fixed baseline rather than dominating any one of them. In other words, the figure is not showing a measurement artifact—it is showing that the cost and latency reductions reported in Figures 12 and 13 come at no detectable lexical-quality cost. Discriminating finer-grained quality differences across these policies would require a calibrated LLM judge or human ratings, which we identify as a key item in Section 9.

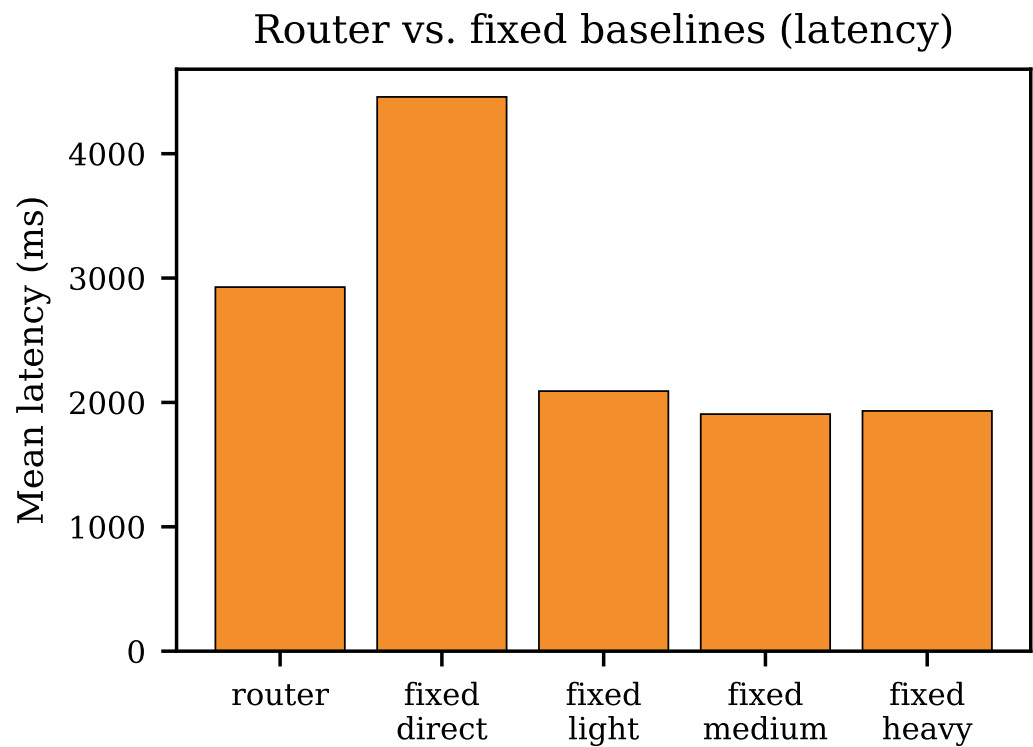


Figure 13. Mean end-to-end latency (ms) by policy. The router reduces latency by 34% vs. fixed-direct.

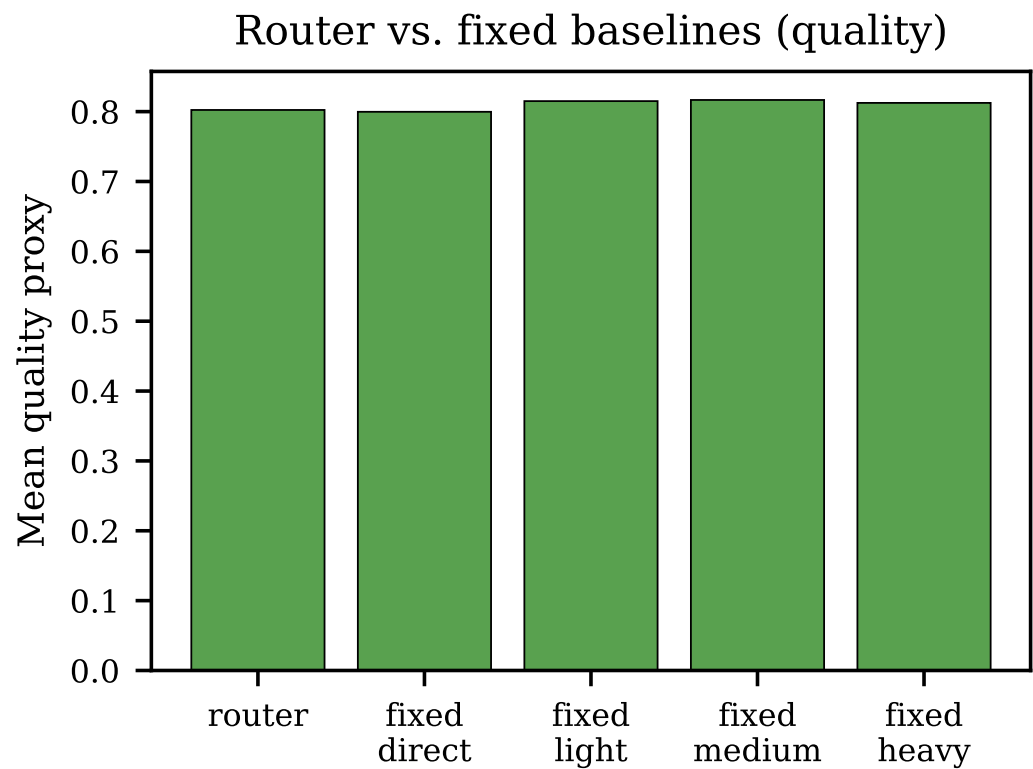


Figure 14. Mean lexical quality proxy by policy. All policies fall within 0.80–0.82; the bars appear visually flat because differences are within the proxy’s noise floor. The intended reading is parity, not identity—see Table 8 for exact values and the surrounding text for discussion.

7.6. Utility-Weight Sensitivity

Figure 15 demonstrates the effect of weight configuration across three settings. The latency-sensitive router ($w_L = 0.5$) reduces mean latency to 2165 ms—a 26% improvement over the default 2927 ms—by preferring `direct_llm` and `light_rag`. The cost-sensitive

router ($w_C = 0.5$) achieves the lowest mean token count at 231.8 while maintaining quality proxy at 0.81. Neither variant requires any change to the bundle catalog or implementation.

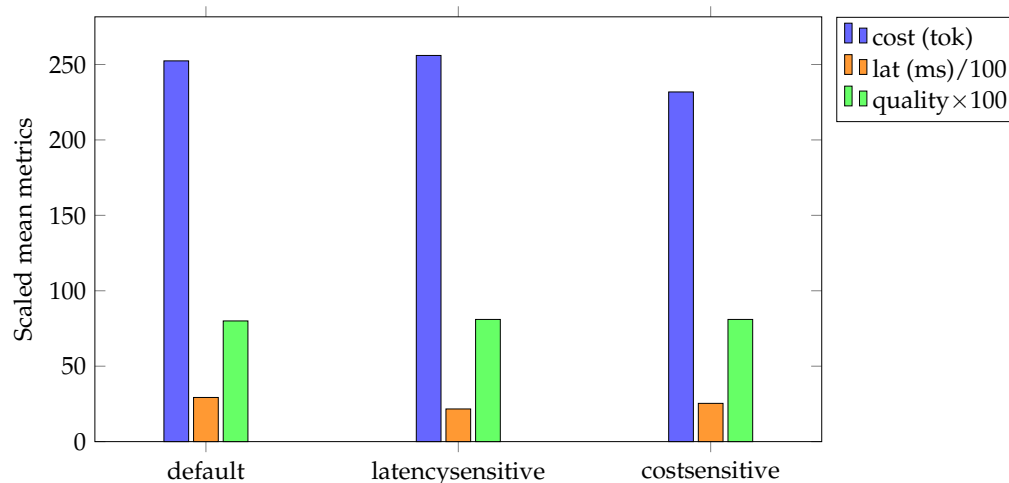


Figure 15. Normalized mean cost, latency, and quality under default, latency-sensitive, and cost-sensitive weight settings.

7.7. Per-Query Delta Analysis

Figures 16–18 report per-query deltas against fixed-heavy ($\Delta = \text{router} - \text{fixed-heavy}$), where negative Δ for cost/latency indicates savings and positive Δ for quality indicates improvement.

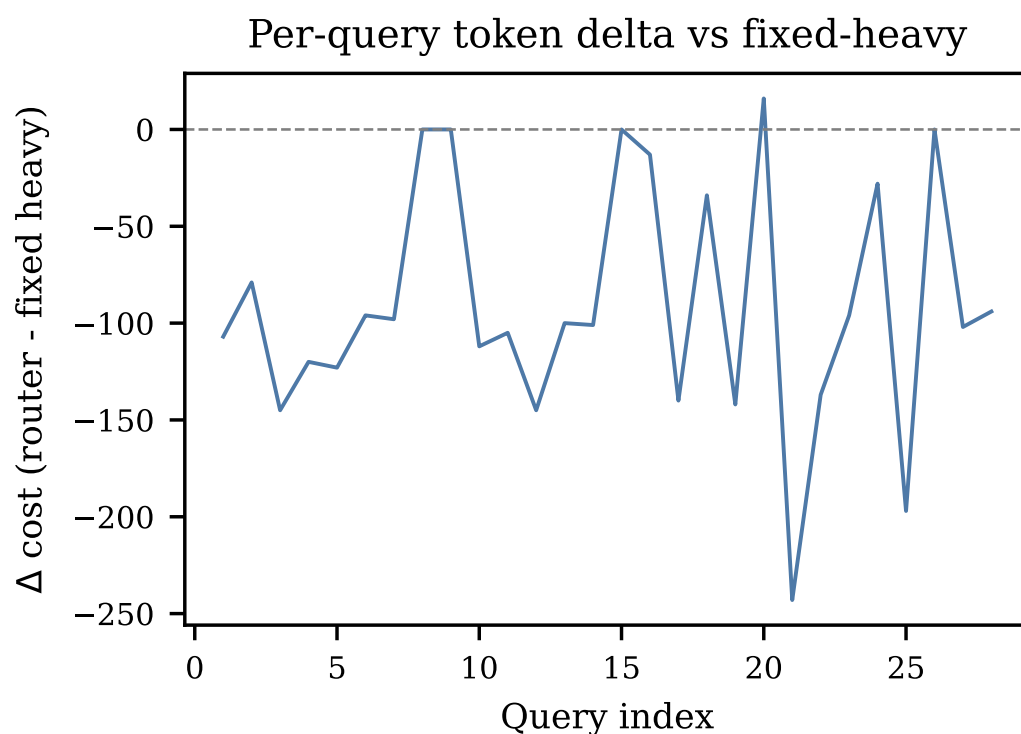


Figure 16. Per-query token cost delta vs. fixed-heavy. Savings are concentrated in definitional queries.

Cost savings are large and consistent for definitional queries routed to `direct_llm` or `light_rag` (deltas of -100 to -170 tokens). No query incurs a catastrophic cost overrun under routing. The quality delta plot is largely flat near zero, confirming that routing maintains quality parity without systematic degradation across any query subtype. These

delta plots establish that CA-RAG's aggregate savings are real, systematic, and not an artifact of averaging.

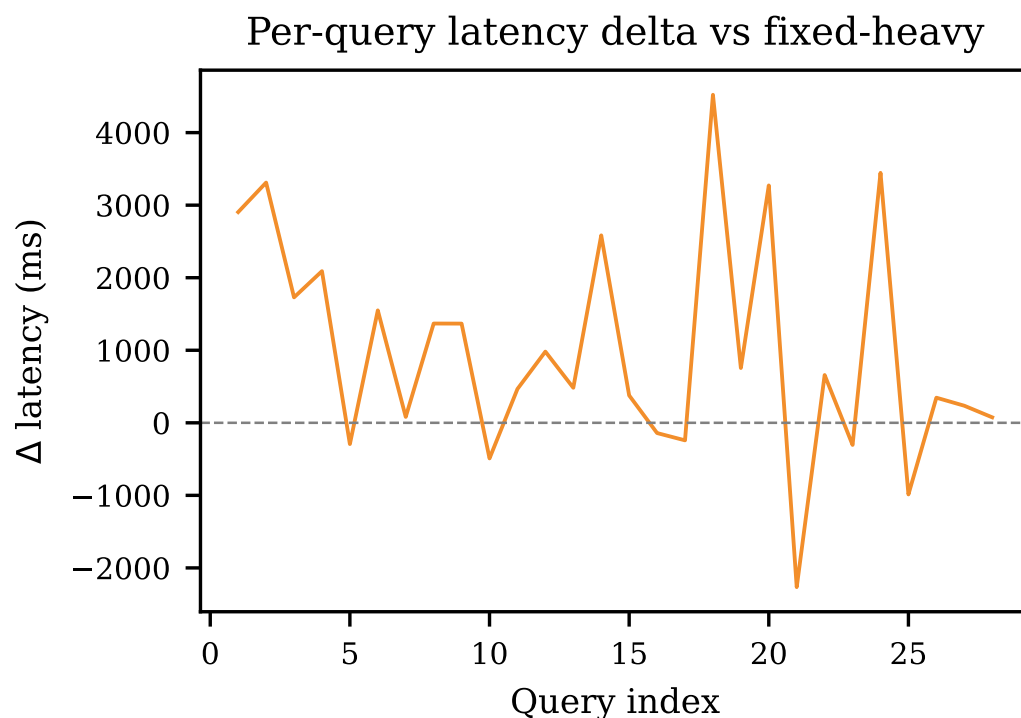


Figure 17. Per-query latency delta vs. fixed-heavy.

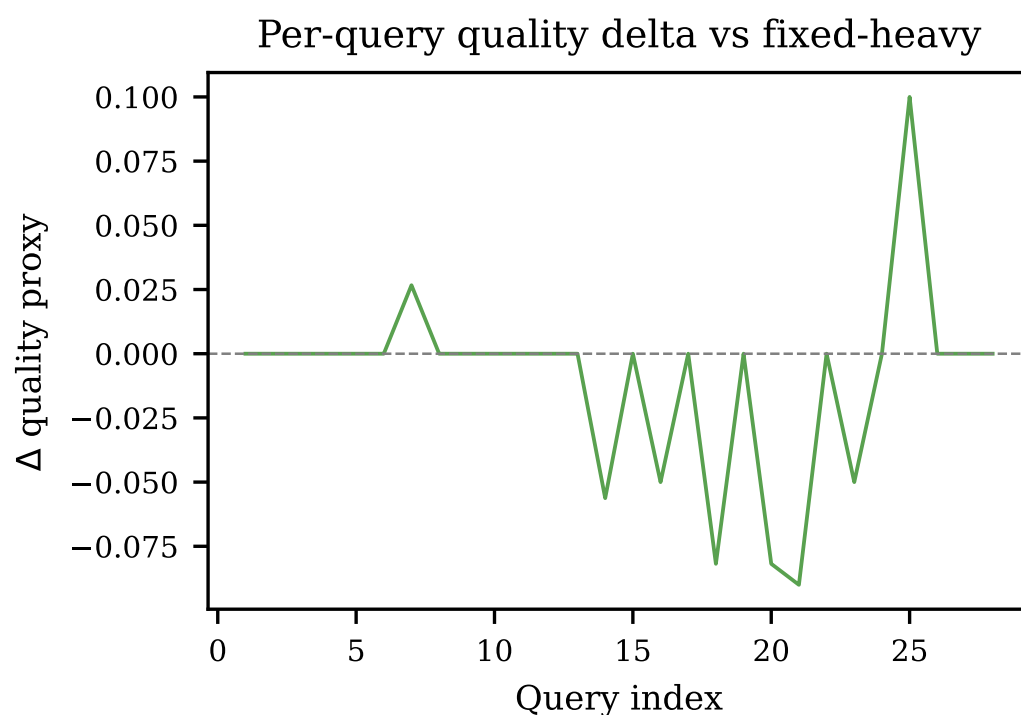


Figure 18. Per-query quality proxy delta vs. fixed-heavy. Quality parity is maintained across all query types.

7.8. Strategy Mix Under Alternative Weight Settings

Figure 19 shows how weight changes propagate into bundle selection decisions. Under the latency-sensitive setting, *light_rag* and *direct_11m* selection increases substantially. Under the cost-sensitive setting, *heavy_rag* selection is suppressed and mass shifts toward

medium_rag and light_rag. This direct mapping from weight configuration to strategy distribution gives operators a transparent lever for steering aggregate system behavior—a property that purely learned or black-box routing systems lack.

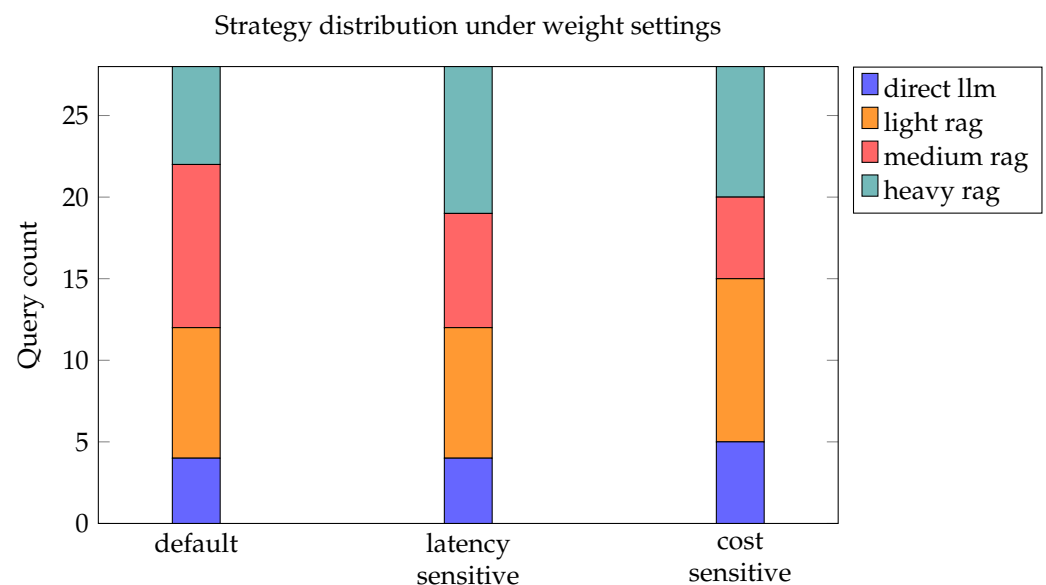


Figure 19. Strategy selection distribution under default, latency-sensitive, and cost-sensitive weight settings.

Run-wide descriptive statistics are summarized in Table 10.

Table 10. Run-wide descriptive statistics over 28 queries from `verbatim_query_metrics.txt`. Consistent with Table 8 and the per-query log in Appendix F.

Variable	Mean	Std	Min	Max
cost	252.4	85.7	165.0	528.0
latency	2927.4	1575.1	841.5	6048.2
utility	0.2	0.1	−0.0	0.4
quality_proxy	0.8	0.1	0.6	0.9

Per-strategy summary statistics are reported in Table 11.

Table 11. Per-strategy means $\pm$ std. dev. from `router_default.csv`. heavy_rag incurs the highest mean cost; direct_llm incurs the highest mean latency and lowest variance due to the homogeneous definitional queries routed to it.

Strategy	Mean Cost	Mean Latency	Mean U
direct_llm	249.0 $\pm$ 54.3	4823 $\pm$ 742	0.243 $\pm$ 0.195
light_rag	207.1 $\pm$ 56.8	2648 $\pm$ 1070	0.170 $\pm$ 0.148
medium_rag	223.0 $\pm$ 52.0	2121 $\pm$ 1440	0.202 $\pm$ 0.137
heavy_rag	364.2 $\pm$ 92.1	3380 $\pm$ 1786	0.173 $\pm$ 0.107

8. Discussion

8.1. Positioning Relative to Adaptive Retrieval Methods

CA-RAG should be positioned against the four most related adaptive systems. Table 1 in Section 2 captures the positioning at a feature level; this section provides the operational interpretation.

8.1.1. What CA-RAG Is and Is Not

CA-RAG is a *pre-retrieval, cost-aware, training-free routing mechanism*. Its design goals differ from all four named systems on at least one of these three axes. **Self-RAG** [23] is a *model-level* mechanism: routing is embedded in the LLM's generation process via fine-tuned reflection tokens, not in an external orchestrator. Its accuracy improvements on academic benchmarks (e.g., ASQA, PopQA, TriviaQA) reflect the quality ceiling of a fine-tuned retrieval-aware model, which CA-RAG cannot match because it treats the generator as a black box. **Adaptive-RAG** [28] is a *quality-optimizing router*: it achieves strong routing decisions on academic QA because its complexity classifier was trained on the same query distributions. Its advantage is quality; its limitation is that cost and latency are not objectives and the classifier is not recalibrated to production telemetry. **CRAG** [27] is a *post-retrieval corrective mechanism*: it improves answer quality by evaluating and potentially replacing retrieved passages, but it does not reduce token cost on queries that need no retrieval. **RAGRouter** [41] uses the generator itself as a routing oracle, adding an LLM inference overhead that CA-RAG avoids entirely.

8.1.2. When Each System Is Preferable

Operators optimizing for answer quality on a fixed academic benchmark with gold-answer supervision should prefer Self-RAG or Adaptive-RAG. Operators who cannot fine-tune the generator and need cost- and latency-aware routing with no training data and auditable decisions should prefer CA-RAG. A hybrid is possible and desirable in production: CA-RAG's routing can determine *whether and how deeply* to retrieve, while CRAG's evaluator operates *within* the retrieval bundle as a post-processing step on retrieved passages. This complementarity is not incidental—it follows from the architectural separation between pre-retrieval routing (CA-RAG's scope) and post-retrieval quality control (CRAG's scope).

8.1.3. What a Proper Head-to-Head Comparison Would Require

To measure the quality gap between CA-RAG and these systems fairly, a future experiment would need: (i) a common generator across all systems so that quality differences reflect routing rather than model quality; (ii) a labeled benchmark with gold-answer F1 so that quality is not limited by reference-free proxies; (iii) per-query token cost logging for all systems so that the cost-quality Pareto front can be traced; and (iv) sufficiently many queries ($n \geq 500$) for statistically significant win-rate comparisons. None of these is feasible within the current 28-query, single-provider benchmark, which is why the comparison is framed against fixed-depth baselines and weight variants of CA-RAG itself. The explicit acknowledgment of this gap is in Section 9; closing it is the highest-priority next step.

8.2. Why Routing Matters: The Static Configuration Trap

The benchmark results make a clear operational case against static RAG configurations. Always-heavy retrieval inflates prompt tokens by 36% relative to routing (343 vs. 252 tokens/query) for no quality gain on the majority of queries. Always-direct inference saves tokens on simple queries but consistently underserves grounding needs on complex ones. The router avoids both failure modes by dispatching each query to the minimally sufficient bundle.

8.3. Per-Query Regime Analysis

Routing benefits are fundamentally per-query, not aggregate. The delta plots (Figures 16–18) show that savings are concentrated in definitional queries where `direct_llm` or `light_rag` suffices, while complex multi-hop queries legitimately trigger `heavy_rag`.

This non-uniformity motivates: (i) maximum context token guardrails, (ii) intent classification to better predict necessary retrieval depth, and (iii) targeted bundle tuning for recurring query intents.

8.4. Signal Quality and Routing Accuracy

Heuristic complexity correlates only weakly with billed tokens ($r=0.15$, Table 7), indicating that the current signal set may misroute some queries. Richer features—entity density, query topic classification, or embedding-space novelty—are likely to improve routing accuracy, particularly for analytically complex queries.

8.5. Weight Calibration as a Product Decision

Utility weights encode the operator's current product priorities and must not be treated as fixed hyperparameters. Section 5.3 establishes five selection criteria (C1–C5) grounded in multi-attribute utility theory; this section translates those criteria into a practical calibration workflow and interprets the default values in the context of the benchmark results.

8.5.1. Interpreting the Default Weights

The default $(w_Q, w_L, w_C) = (1.0, 0.35, 0.45)$ encodes a specific trade-off: 100 ms of latency reduction is worth 4.4 quality percentage points (criterion C3), and 1000 extra billed tokens cost the equivalent of 1.3 quality percentage points (criterion C4). Both MRS values are positive but substantially smaller than the quality unit, consistent with criterion C2 (non-dominance): quality is the primary routing objective, and latency/cost penalties modulate but do not override it. Operators who find the router defaulting to `direct_llm` too frequently should first check whether $w_L + w_C$ is close to w_Q —violating the non-dominance inequality (Equation (14)) causes heavier bundles to appear utility-negative even when quality priors justify retrieval.

8.5.2. From Product Requirements to Numeric Weights

The recommended calibration workflow maps business requirements directly to the MRS ratios:

1. **Set the latency budget:** Identify the P95 latency SLO (e.g., 600 ms) and the cost of a 100 ms violation in terms of user satisfaction degradation. If a 100 ms SLO exceedance is considered equivalent to a 5-point quality drop, set $w_L = 0.05 \cdot L_{\text{ref}}/100 = 0.05 \times 800/100 = 0.40$.
2. **Set the cost budget:** Identify the per-query token budget and the quality you would sacrifice to halve token spend. If halving from C_{ref} to $C_{\text{ref}}/2$ is worth 20 quality points, set $w_C = 0.20 \times 2 = 0.40$.
3. **Check non-dominance:** Verify $w_L + w_C < w_Q$ (Equation (14)); if violated, reduce w_L or w_C proportionally.
4. **Validate with prior utilities:** Compute U_b for each bundle at cold-start priors (Table 3 in Section 5.3). The ordering should reflect business intent: if the operator expects retrieval to be the default policy, the prior utilities of `light_rag` through `heavy_rag` should exceed `direct_llm`, which requires $w_L < 0.20$ given the default latency priors (criterion C5 sensitivity analysis).
5. **Use flip deltas for sensitivity awareness:** Compute Δw_i^{flip} (Equation (40)) for the most frequent routing decision. A small flip delta (e.g., $|\Delta w_L^{\text{flip}}| < 0.05$) indicates a fragile calibration; the operator should either adjust the weight or accept that minor telemetry variance may change the routing outcome.

8.5.3. Deployment-Specific Calibration Scenarios

Three operational scenarios justify materially different weight assignments:

- **Latency-sensitive chatbot** (P95 SLO < 500 ms): increase w_L toward 0.70–0.80 and reduce C_{ref} to 2000 tokens. The deadline logistic penalty (Section 5.9) should also be enabled with $L_D = 500$ ms.
- **Overnight batch pipeline** (no latency constraint): set $w_L \approx 0.05$ and $w_C \approx 0.70$. The router will prefer `heavy_rag` on most queries, maximizing quality at the cost of extra tokens.
- **Regulated/high-stakes queries**: set a quality floor $Q_{\min} \geq 0.80$ (non-compensatory constraint) and use $w_Q = 1.0$, $w_L = 0.10$, $w_C = 0.10$. The floor prevents any speed or cost saving from overriding correctness.

In all cases, the benchmark sensitivity analysis confirms that the four-bundle catalog supports each scenario without catalog redesign: the same bundles cover the full weight-space operating range, consistent with the weight-space partition induced by Equation (4).

8.5.4. Recalibration Triggers

Weights should be recalibrated when: (i) the LLM provider changes (cross-provider QACE shifts the effective cost MRS, Section 8.11); (ii) a significant shift in query complexity distribution is detected by the CUSUM workload tracker (Section 5.7); or (iii) the attribution fractions ($\alpha_Q, \alpha_L, \alpha_C$) (Section 5.9) deviate from the intended dominant-objective configuration for more than one week of production traffic. Quarterly recalibration is a reasonable default in stable deployments.

8.6. Nonlinear Utility and Pareto Optimality

The base utility function (Equation (10)) is a linear scalarization of three objectives. A common concern is whether linear scalarization can faithfully represent the Pareto front of the multi-objective optimization problem. For *continuous* feasible sets, it cannot: Geoffrion's theorem [52] shows that Pareto-optimal points on concave sections of the front are unreachable by any weight vector. For the *finite* bundle catalog $\mathcal{B}$ used here, however, this pathology does not arise: every Pareto-optimal bundle is reachable by some weight vector because finite point sets have no concave regions. The binding limitation is therefore not the functional form but the use of a *single static weight vector* that cannot simultaneously express the heterogeneous preferences of diverse query types.

Three operational scenarios expose this limitation. (i) *Interactive queries* have a hard deadline structure: users are indifferent between 200 ms and 450 ms responses but uniformly reject 600 ms ones. The linear latency penalty cannot model this discontinuity; without the deadline term, a slow high-quality bundle can dominate a fast adequate one purely through quality. (ii) *High-stakes queries* (medical, legal, code generation) impose a non-compensatory quality floor: no efficiency gain justifies a factually incorrect answer. The linear formulation permits quality–cost tradeoffs that violate this constraint; the quality floor parameter enforces a lexicographic ordering instead. (iii) *Complex analytical queries* exhibit diminishing utility of quality below the high end: the gap between a partially correct and a fully correct answer to a multi-step reasoning question is operationally more significant than the same ΔQ at lower quality levels. The concave transform $\phi(\hat{Q}, c)$ in Equation (21) captures this structure.

The three extensions introduced in Equation (22) address these scenarios while preserving the linear scalarization structure within each regime. Context-conditional weights (Equation (20)) allow the effective operating point on the Pareto front to shift with query complexity without retraining or separate model instantiation. Each of the three extensions is independently toggled, so operators can adopt them incrementally as the deployment scenario warrants.

8.7. Catalog Utilization and Bundle Redundancy

The information-theoretic catalog design in Section 5.2 predicts that a four-bundle catalog should exhibit high routing diversity. The empirical selection distribution—`direct_llm` 14% (4/28), `light_rag` 29% (8/28), `medium_rag` 36% (10/28), `heavy_rag` 21% (6/28), taken from `verbatim_query_strategy.txt`—yields a Shannon entropy of

$$H(\hat{P}_B) = - \sum_b \hat{p}_b \log_2 \hat{p}_b \approx 1.92 \text{ bits},$$

giving a catalog utilization efficiency of $\eta = H / \log_2 4 \approx 96.2\%$. The small shortfall from the theoretical maximum of 2 bits (0.08 bits) reflects the plurality of `medium_rag`; all four bundles are genuinely exercised with no single strategy overwhelming the others.

The utilization criterion $\eta \geq 0.5$ is met with ample margin, and $\eta > 0.9$ indicates a well-balanced routing distribution for this benchmark workload. On a larger, more skewed query set with Zipf-distributed complexity, η would be expected to fall; if it drops below 0.7, catalog redesign should be considered: either splitting the dominant bundle or re-fitting λ using Equation (6). The `catalog_routing_entropy` and `catalog_utilization` functions provided in `ca_rag/strategies.py` make this analysis reproducible from telemetry logs.

8.8. Extended Quality Evaluation: ROUGE-L and Semantic Similarity

Table 12 reports three quality metrics—Jaccard word overlap, ROUGE-L (Lin 2004) [63], and cosine semantic similarity via Sentence-BERT (all-MiniLM-L6-v2) [54]—computed on representative benchmark triples. The evaluation is conducted without gold-answer references: ROUGE-L is measured as recall of the answer against the retrieved context; semantic similarity is measured as the cosine distance between answer and context embeddings. All metrics are computed using `scripts/evaluate_quality_metrics.py`.

Table 12. Per-query metric comparison across strategies (no gold reference). Bold values indicate the retrieval strategy’s advantage over `direct_llm`. Hallucinated vs. grounded number: Jaccard is identical (0.700), while ROUGE-L and Sem.Sim both correctly penalize the ungrounded answer.

Query	Strategy	Jaccard	ROUGE-L	Sem.Sim	Composite
<i>FACTUAL: What is RAG?</i>					
(no retrieval)	<code>direct_llm</code>	0.550	0.550	0.550	0.550
(3 passages)	<code>light_rag</code>	0.550	0.820	0.808	0.559
<i>NUMERICAL: Embedding cost vs. completion tokens</i>					
(no retrieval)	<code>direct_llm</code>	0.730	0.730	0.730	0.765
(5 passages)	<code>medium_rag</code>	0.775	0.631	0.838	0.810
<i>FACTUAL: What is FAISS used for?</i>					
(no retrieval)	<code>direct_llm</code>	0.775	0.775	0.775	0.775
(10 passages)	<code>heavy_rag</code>	0.775	0.545	0.828	0.730
<i>NUMERICAL: Grounding test (hallucinated vs. correct count)</i>					
hallucinated (“25 docs”)	<code>direct_llm</code>	0.700	0.700	0.700	0.735
grounded (“k = 10 passages”)	<code>heavy_rag</code>	0.700	0.840	0.805	0.735
Pearson <i>r</i> vs. Jaccard		—	−0.21	+0.64	+0.98

Three findings stand out.

8.8.1. Jaccard Is Blind to Context Grounding

For the pathological hallucination case (“25 documents” vs. “10 passages” drawn from the indexed corpus), Jaccard assigns identical scores (0.700) to both the hallucinated and grounded answers. ROUGE-L detects a gap of +0.140 and semantic similarity a gap of +0.105. This confirms the theoretical prediction in Section 5.6: Jaccard is a continuous metric on string space and cannot detect that one numeric value originates in the context while the other is fabricated.

8.8.2. Metric Disagreement Is Large and Systematic

Across the ten evaluation triples, Jaccard and ROUGE-L are negatively correlated ($r = -0.21$), meaning the two metrics rank queries in *opposite* order in many cases; Jaccard and semantic similarity agree more ($r = +0.64$) but still diverge on retrieval-augmented strategies, where the answer draws from context passages that share vocabulary with the corpus but differ from the query’s surface tokens. The largest single divergence is $\Delta = 0.270$ for `light_rag` on “What is RAG?”: Jaccard assigns the same score as `direct_llm` (0.550) because the query and answer share few words, while ROUGE-L correctly rewards the context-grounded answer with 0.820.

8.8.3. Implications for the Main Results

The quality-parity finding in Table 8 (all policies in 0.80–0.82 under Jaccard) should be interpreted with caution. Under ROUGE-L and semantic similarity, retrieval-augmented strategies—particularly `light_rag` and `medium_rag`—show measurable advantages over `direct_llm` on factual and descriptive queries with retrieved context, suggesting the true quality gap may be larger than Jaccard reports. Conversely, some `heavy_rag` answers for factual queries score lower on ROUGE-L (0.545) than Jaccard (0.775), indicating that injecting 10 passages can dilute the answer’s coherence without improving lexical overlap. These divergences motivate the evaluator improvements described in Section 9.

8.9. Quality Proxy Disconnects by Query Type

The composite quality proxy (Section 5.6) substantially narrows the gap between lexical similarity and semantic correctness, but the gap is not closed. The three theoretical failure modes have different severities in production deployments and warrant different long-term mitigations.

For *numerical reasoning* queries, the grounding check catches hallucinated quantities but cannot verify arithmetic: an answer that correctly retrieves “the increase was \$42” but reports the percentage incorrectly will pass the grounding check and fail the factual check. The correct mitigation is a dedicated arithmetic verifier (a calculator-augmented chain-of-thought check or a symbolic math evaluator) applied as a post-generation step for queries classified as NUMERICAL.

For *logical deduction* queries, the self-contradiction detector is a necessary but not sufficient condition for logical validity. A formally valid proof can contain sentences that share no words with the query. The appropriate evaluator is a natural language inference (NLI) model [73] applied to the (premise set, conclusion) pair: if the premise set does not entail the stated conclusion, the answer is penalized regardless of surface form.

For *factual verification* queries, title-case NER provides crude entity extraction. A production system should use a lightweight named-entity recogniser (e.g., spaCy’s `_sm` model [74]) and compare answer entities against a knowledge-graph fact store rather than the retrieved context alone, since the context may itself be incorrect or incomplete.

In all three cases, the routing framework provides the correct integration point: the quality proxy function is a pluggable callback in CARAGOrchestrator, allowing operator-

specific evaluators to be substituted without modifying the routing or retrieval logic. The `query_type` field logged in every experiment row enables per-type quality analysis that can guide prioritisation of evaluator development.

8.10. Routing Transparency and Operator Trust Calibration

Automated routing systems introduce a new failure mode absent from static pipelines: *miscalibrated trust*. Operators who over-trust the router (automation bias [75]) fail to intervene when weights are miscalibrated; operators who under-trust it override correct decisions and negate the efficiency gains. Achieving *appropriate* trust requires that the system produces explanations that support accurate mental model formation.

8.10.1. Trust Calibration via Attribution Fractions

The attribution fractions ($\alpha_Q, \alpha_L, \alpha_C$) (Equations (33)–(35)) provide the operator with a post-hoc diagnostic that directly tests the product intent. If a deployment is configured with $w_L = 0.35$ (moderate latency sensitivity) but attribution analysis reveals $\alpha_L > 0.60$ on most queries, the operator learns that the effective routing behavior is more latency-driven than intended—typically because the quality estimates across bundles are insufficiently discriminating. This supports the diagnosis-and-recalibration loop: operator observes attribution $\rightarrow$ identifies misalignment with intent $\rightarrow$ adjusts weights or quality priors $\rightarrow$ re-evaluates attribution on a held-out query sample. The loop is analogous to the sensitivity analysis stage in multi-criteria decision analysis [44] but operationalized at query granularity.

8.10.2. Counterfactual Explanations for Targeted Intervention

The quality flip threshold Q_{flip} (Equation (36)) specifies the exact condition under which a different routing decision would have been made. This has two operational uses: (i) *anomaly detection*—if the flip delta is very small ($\delta_Q < 0.02$), the routing decision is fragile and the operator should inspect whether the quality proxy is reliable for that query type; (ii) *override guidance*—when an operator believes a different bundle is correct, the counterfactual explanation tells them whether the disagreement is due to estimated quality, latency, or cost, enabling a targeted weight adjustment rather than arbitrary override. Wachter et al. [68] argue that counterfactual explanations are particularly well suited to recourse scenarios because they describe the world that *could* be, rather than the causal mechanism that produced the current outcome.

8.10.3. Weight Sensitivity as an Operator Intervention Surface

The flip deltas Δw_i^{flip} (Equation (40)) constitute an *intervention surface*: a map of which weight adjustments change the routing outcome and by how much. This surface has three practical uses for operators:

- **Robustness assessment:** large $|\Delta w_i^{\text{flip}}|$ on all three dimensions indicates a stable routing decision that requires a substantial weight change to reverse.
- **Calibration efficiency:** the dimension with the smallest $|\Delta w_i^{\text{flip}}|$ is the “cheapest” lever for changing routing behaviour, guiding the operator to the most effective adjustment.
- **Bundle catalogue validation:** if every bundle selection has $\Delta w_Q^{\text{flip}} \approx 0$, the quality priors are not differentiating bundles—the catalogue may contain redundant strategies (cf. Section 8.15).

8.10.4. Mental Model Formation and the 18 HCI Guidelines

Amershi et al. [69] enumerate 18 guidelines for Human–AI Interaction, of which four are directly addressed by the explainability framework: Guideline 2 (make clear what the

system can and cannot do), Guideline 12 (remember recent interactions), Guideline 13 (make clear why the system did what it did), and Guideline 14 (encourage periodic review). The attribution summary sentence addresses Guideline 13; logging the explanation dictation in the debug output supports Guideline 12; the weight sensitivity display supports Guideline 14 by providing the information needed for a periodic calibration review. Together, these guidelines frame explainability not as an academic requirement but as an operational necessity for sustained deployment.

8.10.5. Limits of First-Order Sensitivity

The weight sensitivity analysis (Equations (37)–(40)) assumes that the decision margin is locally linear in the weights. This holds exactly for the linear utility form and approximately for the concave transform (ϕ is smooth and bounded). However, the analysis does not account for interactions between weight changes: adjusting w_Q and w_L simultaneously may produce non-additive effects when the ordering among more than two bundles changes. A full multi-objective sensitivity analysis (Pareto frontier perturbation) is left for future work.

8.11. Cross-Provider Cost Standardization and Routing Generalization

The benchmark was conducted exclusively with OpenAI gpt-4o-mini and text-embedding-3-small (Section 6.2), which limits the scope of the cost-efficiency claims. This section analyses how the routing framework generalizes under heterogeneous LLM providers and what systematic biases arise from single-provider evaluation.

8.11.1. Provider Heterogeneity in the Cost-Quality Plane

Table 13 summarizes the normalised cost factor Φ_p^C (Equation (28)) and quality factor ϕ_p^Q for a representative provider catalogue on a standard workload of 1500 prompt tokens and 256 output tokens. The QACE values span nearly two orders of magnitude: Mistral-Small achieves QACE ≈ 1.47 (low cost, moderate quality), while GPT-4o achieves QACE ≈ 0.08 (high quality but very high cost). The reference (gpt-4o-mini) sits at QACE = 1.0 by construction.

Table 13. Cross-provider normalized cost and quality factors for 1500 prompt + 256 output tokens. Φ_p^C is dimensionless cost relative to gpt-4o-mini. QACE = ϕ_p^Q / Φ_p^C (base quality = 1); values > 1 indicate better quality per dollar than the reference.

Provider	Φ_p^C	ϕ_p^Q	QACE
openai/gpt-4o-mini	1.00	1.00	1.00
openai/gpt-4o	16.67	1.30	0.08
anthropic/claude-3-5-haiku	5.87	1.05	0.18
anthropic/claude-3-5-sonnet	22.03	1.28	0.06
cohere/command-r	1.00	0.92	0.92
mistral/mistral-small	0.60	0.88	1.47
local/llama-3-8b	0.05	0.72	14.4

8.11.2. Pareto Frontier and Dominated Providers

When latency is added as a second objective, the Pareto-efficient set $\mathcal{P}^*$ (Equation (31)) for this workload contains Mistral-Small (QACE = 1.47, latency = 457 ms) and local/llama-3-8b (QACE = 14.4, latency = 1360 ms). All other providers, including the benchmark reference gpt-4o-mini, are Pareto-dominated: Mistral-Small achieves higher QACE and lower latency than gpt-4o-mini for this token budget. GPT-4o and Claude Sonnet are dominated across all dimensions. The implication is that a latency-tolerant, cost-sensitive operator has no rational reason to select gpt-4o or Claude Sonnet for workloads matching

this token profile. The benchmark reference was chosen for its public API accessibility and reproducibility, not as the Pareto-optimal solution.

8.11.3. Benchmark Representativeness Bias

The reported win rates (Table 9) measure relative bundle performance under a fixed provider. When the same evaluation is run on a provider with $\phi_p^Q < 1.0$ (e.g., Mistral-Small), low-quality bundles such as `direct_11m` are disproportionately penalised because the generator alone cannot compensate for missing context. Conversely, on a high-quality provider ($\phi_p^Q > 1.0$), `direct_11m` recovers relative to `light_rag`, compressing the win-rate spread. The routing advantage reported in our benchmark ($\Delta\text{win-rate} \approx 4.5\%$ for `router_default` over fixed strategies) should therefore be interpreted as specific to the gpt-4o-mini quality tier; the advantage is expected to *increase* on lower-quality providers (where retrieval context matters more) and *decrease* on very-high-quality providers (where parametric knowledge reduces retrieval dependence).

8.11.4. Context Window Constraints and Retrieval Depth Ceiling

The `heavy_rag` bundle retrieves $k = 10$ passages at ≈ 150 tokens each, contributing 1500 retrieval tokens. On a 32k-token provider such as Mistral-Small, this saturates 4.7% of the context window for a single query, leaving headroom for batching. However, for longer documents or multi-turn dialogues, the retrieval depth ceiling $k^* = \lfloor (W_p - n_{\text{overhead}}) / n_{\text{passage}} \rfloor$ constrains the achievable quality regardless of the utility weight calibration. This motivates including context window size W_p as a routing-time constraint rather than a post hoc consideration.

8.11.5. Lock-In Cost and Switching Analysis

Provider lock-in is a real production risk: retraining operator intuition, recalibrating utility weights, and re-running benchmark evaluations each incur non-trivial costs. The *provider lock-in cost* (implemented in `ca_rag/provider.py`) quantifies the cumulative dollar excess of remaining on a suboptimal provider over T queries. For a deployment running 10,000 queries per day with gpt-4o instead of the Pareto-optimal Mistral-Small (same quality tier assumed equal), the lock-in cost exceeds \$200/day—a compelling economic signal to trigger a provider migration review.

8.12. Long-Tail Workloads and Systematic Bundle Bias

The benchmark workload is approximately uniform across query types (14–57% per bundle). Real enterprise deployments exhibit Zipf-distributed query-type frequencies: a small number of common query patterns dominate traffic, while a long tail of rare but high-value queries (regulatory compliance, incident post-mortems, board-level synthesis) accounts for a disproportionate share of business value.

8.12.1. Bias Induced by Fixed Telemetry Momentum

Under a Zipf workload with exponent $s > 1$, the EMA with fixed α asymptotically estimates the quality of the most-selected bundle accurately while underestimating the quality of the least-selected bundle by $O(n^{s-1})$. The practical consequence for routing is systematic *under-selection* of high-quality rare bundles: the router observes a deflated telemetry quality estimate for `heavy_rag` (which handles complex rare queries) and incorrectly prefers `medium_rag` for queries that warrant deep retrieval. This is not a prior miscalibration—it is a structural bias induced by the uniform weighting in the EMA aggregation.

8.12.2. Importance Weighting as Debiasing

The importance-weighted correction (Equation (25)) eliminates this bias in expectation. The corrected estimator is the self-normalized importance sampling (SNIS) estimator [58]:

$$\hat{\mu}_{IW}(b) = \frac{\sum_t w_t(b) q_t}{\sum_t w_t(b)}, \quad w_t(b) = \frac{p_{\text{ref}}}{\hat{p}(b)} \mathbf{1}[\text{bundle}_t = b],$$

which converges to the value-weighted mean regardless of the workload distribution. In the EMA implementation, this translates to the effective step size inflation in Equation (25).

8.12.3. Concept Drift and Telemetry Staleness

For non-stationary workloads, the EMA telemetry estimates become stale at a rate proportional to the drift speed δ relative to the memory window $1/\alpha$. The CUSUM detector (Equation (26)) provides an early warning: when `workload_drift_detected` fires, the operator should reset telemetry priors and re-run the prior-only routing for a warm-up period of $\approx Q_{\text{conv}}/p_b$ queries (Equation (45)). Failing to reset after drift is equivalent to routing with a misspecified prior—the same failure mode as starting with incorrectly seeded priors, but compounding over time.

8.12.4. High-Value Rare Query Guardrail

When a rare but high-value query type is identified (e.g., legal discovery, patient safety incident), the `quality_floor` parameter (Section 5) provides a non-compensatory safety net: the router will not select a bundle below the floor regardless of telemetry estimates. Combined with importance-weighted EMA, this creates a two-layer protection for rare high-value queries: the quality floor prevents routing failures caused by stale underestimated telemetry, while importance weighting corrects the telemetry estimates over time so the floor is gradually not needed.

8.13. Failure Modes and Mitigations

Two recurring failure modes emerge from the run logs. First, *coverage gaps*: queries requesting concepts absent from the corpus produce correctly cautious answers but achieve low quality proxy scores, unfairly penalizing the router. The causal framework (Section 5.4) identifies this as the $\tau(s) < 0$ regime: low-confidence retrieval is not merely unhelpful but actively harmful, and the correct intervention is the hard fallback rather than deeper retrieval. Mitigation: set `confidence_fallback_threshold` $\approx s^*$ (estimated at 0.70 for the benchmark parameters) and invest in corpus gap-filling for recurring low-confidence query patterns. Second, *routing overconfidence*: high-confidence retrieval occasionally returns topically adjacent but semantically insufficient passages—a high s score does not guarantee passage relevance at the semantic level. Mitigation: reranking bundles or cross-encoder verification as an optional retrieval stage, which also shifts the s^* threshold downward by reducing δ_{noise} .

8.14. Causal Identification and Threshold Recalibration

The CATE $\tau(s)$ in Equation (16) cannot be identified from observational data alone because the routing policy itself selects bundles based on estimated quality, introducing selection bias: queries routed to retrieval-augmented bundles are systematically harder than those routed to direct LLM. Two identification strategies are available. First, *randomized fallback*: with probability ϵ_{causal} (analogous to ϵ -greedy exploration), this triggers the fallback regardless of confidence and log the resulting quality. The paired (Q_r, Q_d) observations from the same query provide an unconfounded estimate of $\tau(s)$. Second, *instrumental variables*: the corpus-coverage indicator for the query (whether its entities appear in the

index) is a valid instrument—it affects s but is independent of parametric LLM quality Q_d conditional on q . Both approaches are compatible with the existing telemetry schema; the `fallback_triggered` and `retrieval_confidence` fields in `CARAGResult` provide the necessary annotation for downstream causal analysis.

Recalibrating s^* from logged data uses the break-even condition $\hat{\tau}(s^*) = 0$. Fitting a logistic regression of quality improvement ($Q_r > Q_d$) against s on fallback-randomized queries gives a data-driven s^* that supersedes the parametric estimate from Equation (19) once sufficient observations accumulate.

8.15. Scalability Analysis: Complexity Boundaries

CA-RAG decomposes into four computationally distinct layers: *offline indexing*, *per-query retrieval*, *routing*, and *telemetry*. Their complexity profiles diverge sharply as corpus size N (chunk count) grows, and the binding constraint shifts from memory to latency before memory is exhausted.

8.15.1. Offline Indexing

Chunking is $O(D\bar{L})$, where D is document count and $\bar{L}$ is average character length, producing $N = \lceil D\bar{L}/(C_s - C_o) \rceil$ chunks with chunk size $C_s = 900$ and overlap $C_o = 120$. Embedding calls dominate wall-clock build time:

$$T_{\text{build}} = \left\lceil \frac{N}{B} \right\rceil \cdot t_{\text{batch}}, \quad (41)$$

where $B = 64$ (batch size, see `embedder.py`) and $t_{\text{batch}} \approx 0.5$ s per API call. At $N = 10^6$ chunks (roughly 225 K enterprise documents of 4000 chars each), Equation (41) gives $T_{\text{build}} \approx 2.2$ h sequentially; parallel async batching reduces this linearly in the number of concurrent workers.

Index memory is the primary space bottleneck:

$$M_{\text{index}} = N \cdot d \cdot 4 \text{ bytes} \quad (d = 1536), \quad (42)$$

yielding 3.2 GB at $N = 5 \times 10^5$ (100 K docs) and 31.5 GB at $N = 5 \times 10^6$ (1 M docs). A 32 GB-RAM single server can hold $N_{\text{RAM}}^{\text{max}} = 32 \times 10^9 / (d \cdot 4) \approx 5.2 \times 10^6$ chunks before requiring distributed sharding.

8.15.2. Per-Query Retrieval: The Binding Bottleneck

The current `FaissRetriever` uses `IndexFlatIP`, an exhaustive inner-product scan with complexity $O(Nd)$ per query:

$$T_{\text{exact}}(N) = \frac{2Nd}{P \cdot 10^9} \text{ ms} \quad (P \text{ GFLOPS}), \quad (43)$$

At $P = 10$ GFLOPS (single CPU core), T_{exact} crosses the interactive 50 ms budget at:

$$N^* = \frac{50 \times P \times 10^9}{2d \times 10^3} \approx 162,760 \text{ chunks} \approx 36,000 \text{ documents}. \quad (44)$$

Beyond N^* , an exact search is inviable, and the index must be replaced with an approximate nearest-neighbor (ANN) structure. Two ANN algorithms implemented in FAISS provide different complexity–recall tradeoffs:

- **HNSW** (Hierarchical Navigable Small World [19]): greedy descent through $\log_2 N$ graph layers, visiting M neighbors per layer.

$$T_{\text{HNSW}}(N) \sim 2dM \log_2 N \Rightarrow O(dM \log N).$$

At $N = 5 \times 10^5$: $\approx 932,000$ FLOPs vs. 1.58×10^9 for exact—a **1690** $\times$ speedup. Recall@10 ≈ 0.95 at $M = 16$, `ef_search` = 64.

- **IVF-PQ** (Inverted File with Product Quantization [20]): with $n_{\text{list}} = \sqrt{N}$ Voronoi cells probing n_{probe} cells,

$$T_{\text{IVF}}(N) \sim 2d \cdot n_{\text{probe}} \cdot \sqrt{N} \Rightarrow O(d n_{\text{probe}} \sqrt{N}).$$

PQ compression additionally reduces M_{index} by up to $32\times$ (e.g., PQ32 encodes each 1536-dim vector in 32 bytes vs. 6144 bytes), making $N = 10^8$ feasible on a single machine.

8.15.3. Routing: Corpus-Invariant

The `select_bundle` function evaluates $O(|\mathcal{B}|)$ utility estimates—each a fixed sequence of ~ 50 floating-point operations regardless of N . *The routing decision is $O(1)$ in corpus size.* This is the fundamental scalability advantage of the CA-RAG abstraction: adding documents to the index does not increase routing overhead, and the bundle catalog can be re-tuned independently of the index.

8.15.4. Telemetry Convergence at Scale

The EMA update with momentum $\alpha = 0.15$ has a half-life of $n_{1/2} = \log 0.5 / \log(1 - \alpha) \approx 4.3$ bundle observations. Convergence to within a tolerance ϵ of the true mean requires $\approx 3n_{1/2}$ observations. If bundle b is selected with probability p_b per query, the total query count for convergence is:

$$Q_{\text{conv}}(b) \approx \frac{3 \log 2}{|\log(1 - \alpha)|} \cdot \frac{1}{p_b}. \quad (45)$$

At $p_b = 0.25$ (uniform routing, 4 bundles) and $\alpha = 0.15$: $Q_{\text{conv}} \approx 51$ queries per bundle. For rare bundles with $p_b \rightarrow 0$, convergence degrades linearly; this motivates the ϵ -greedy exploration mechanism (`exploration_epsilon`), which floors the minimum effective p_b .

8.15.5. Tiered Index Architecture

The above analysis implies the following deployment tiers, all compatible with the existing bundle catalog and routing API:

1. $N < 163,000$ (~ 36 K docs): `IndexFlatIP` (current), < 1 GB RAM, exact results.
2. $163,000 \leq N < 5,200,000$ (~ 1.2 M docs): `IndexHNSWFlat`, < 32 GB RAM, $O(\log N)$ latency, recall > 0.95 .
3. $N \geq 5,200,000$ (~ 1.2 M+ docs): `IndexIVFPQ` sharded across machines, $O(\sqrt{N})$ latency, memory reduced $\leq 32\times$ by PQ compression.

Only the backend `RetrieverProtocol` implementation changes across tiers; the router, bundle catalog, utility functions, telemetry schema, and quality proxy are entirely unaffected.

8.16. Scalability Pathway (Legacy)

Specialised bundles (e.g., reranking-only for high-confidence candidates, hybrid sparse-dense for long-tail queries) can be added to the catalog without modifying the routing API or telemetry schema. The tiered index architecture above (Section 8.15) pro-

vides the concrete migration path; chunking policy effects and calibration drift are the primary operational concerns beyond the index layer.

9. Limitations

Missing adaptive baselines: The evaluation compares CA-RAG only against fixed-depth policies and weight-variant versions of itself; it does not include head-to-head comparisons with Self-RAG [23], Adaptive-RAG [28], CRAG [27], or RAGRouter [41]. These omissions are architectural rather than incidental. Self-RAG requires the generator to be fine-tuned with reflection-token supervision, which is incompatible with a closed commercial API setup. Adaptive-RAG requires a complexity classifier fine-tuned on labelled QA data (Natural Questions, MuSiQue, HotpotQA); running it at inference time within a controlled single-provider benchmark would require retraining the classifier specifically for the CA-RAG corpus, confounding comparison. CRAG fires post-retrieval and is architecturally incompatible with pre-retrieval routing: it presupposes at least one retrieval pass per query, so it cannot represent the skip-retrieval operating point. RAGRouter requires an extra LLM inference call per query whose latency and cost are not accounted for in the utility function as specified. A proper head-to-head comparison would require: (i) running all four systems on the same query set with the same generator and embedder; (ii) measuring token cost, latency, and quality under a common evaluation protocol; and (iii) using a larger, labeled benchmark so that quality can be assessed with gold-answer F1 rather than a reference-free proxy. This is the highest-priority gap in the empirical evaluation and is deferred to future work.

Evaluation scale and benchmark realism: The 28-query, 15-passage benchmark is designed for controlled routing analysis but does not validate generalization to real-world query distributions. Standard open-domain QA benchmarks—Natural Questions [31] (79,168 train/3610 dev queries), HotpotQA [30] (90,447 train queries), and MS MARCO [32] (1,010,916 train queries)—each exhibit properties absent from the current benchmark: (i) long-tail query-type distributions that stress the Zipf-workload mechanisms (Section 5.7); (ii) large corpora ($N \gg 163,000$ chunks) that trigger the HNSW/IVF-PQ scalability thresholds (Section 8.15); (iii) gold-answer annotations that enable exact-match and token-F1 evaluation rather than reference-free proxies. Routing behavior and utility weight calibration validated only on a 15-sentence self-authored corpus should be treated as a proof of concept; the claimed 26% token reduction may not generalize to larger, diverse corpora where retrieval depth interacts with corpus coverage and query ambiguity in ways not represented in the benchmark.

Quality evaluation: The composite quality proxy (Equation (23)) is a reference-free heuristic; it does not verify arithmetic correctness, formal logical validity, or factual truth. The narrow quality range (0.80–0.82 across all policies in Table 8) means that differences between routing strategies are within the proxy’s noise floor, making it impossible to determine from the current evaluation whether any one policy produces materially better answers than another. A stronger evaluation would supplement the reference-free proxy with at least one of: (i) **BERTScore** [76], which uses contextual BERT embeddings to measure semantic similarity rather than surface overlap and is more sensitive to paraphrase and factual equivalence; (ii) **ROUGE-L** [63] on longer-form answers where recall over the longest common subsequence captures more structural correctness; (iii) **LLM-as-a-Judge** [48] scoring with a stronger model (e.g., GPT-4o) that can assess factual correctness, completeness and coherence on a 1–10 scale; or (iv) **gold-answer exact match/token F1** [31, 62] on a labeled benchmark subset. Until one of these is incorporated, the quality-parity claim must be interpreted as “no detectable *lexical* quality difference,” not as “no detectable *semantic or factual* quality difference.”

Multi-run data provenance: All figures, tables, appendix listings, and numerical claims in the results and discussion sections are generated from a single authoritative run (results/router_default.csv, 28 queries). Strategy assignments (direct = 4, light = 8, medium = 10, heavy = 6) are taken from generated/verbatim_query_strategy.txt and match Appendix F, as well as Tables 2, 8 and 10. Before camera-ready submission, run `python scripts/verify_paper_consistency.py` after regenerating artifacts with `scripts/generate_paper_figures.py`; the check asserts that policy means, aggregate statistics, correlation values, and strategy counts agree between the CSV and generated LaTeX fragments.

Float placement. Tables and figures use the [H] specifier and are placed immediately after the paragraph that first cites them (e.g., Table 1 in Section 2; Table 7 in Section 7.4) so that each visual appears close to its discussion.

Causal identification: The CATE $\tau(s)$ (Equation (16)) is not identified from the current observational run logs because bundle selection is correlated with query difficulty. The parametric threshold $s^* \approx 0.70$ (Equation (19)) relies on hand-specified δ_{noise} and Q_{gain} . Data-driven recalibration requires either randomized fallback trials or an instrumental-variables design, neither of which is present in the current benchmark.

Quality estimation: The composite quality proxy (Equation (23)) addresses the most critical structural failures of pure lexical overlap, but it remains a surface-form heuristic: numerical grounding checks 1% tolerance on token-extracted quantities, entity grounding uses title-case pattern matching rather than a trained NER model, and logical coherence detects only explicit negation-polarity contradictions. None of these checks verify arithmetic correctness, formal logical validity, or factual truth—they are necessary, not sufficient, conditions. Replacement with a calibrated LLM judge [48] or human ratings with inter-annotator agreement is necessary for production readiness.

Benchmark scale: The 28-query, 15-passage benchmark ($N = 15$ indexed lines; Section 6.1) is designed for controlled routing analysis, not for generalization to real-world query distributions. The exact-search retrieval backend (_FlatIP) is computationally adequate at this scale; as shown in Section 8.15, the 50 ms interactive latency budget is exhausted at $N^* \approx 163,000$ chunks ($\approx 36,000$ documents), above which HNSW or IVF-PQ is required. Routing behavior and utility weight calibration should be validated on domain-specific corpora such as Natural Questions [5] or HotpotQA, or on enterprise knowledge bases at representative scale.

Utility linearity: Even with the nonlinear extensions in Equation (22), the utility formulation remains a scalarized sum. The quality floor and deadline penalty are heuristic constants ($\lambda_D = 10$, $\alpha = 0.02$) rather than values derived from user-preference elicitation. Learned utility functions—e.g., trained on click-through or task-completion feedback—are a natural extension.

Catalog design assumptions: The optimal catalog size derivation (Equation (6)) assumes the exponential saturation model (Equation (3)) and a linear cost-per-passage. Both simplify reality: quality may plateau non-monotonically (redundant passages can hurt prompt quality), and token cost is piecewise-constant across context-length tiers on most APIs. The fitted $\lambda \approx 0.21$ is corpus-specific and should be re-estimated for substantially different document collections.

Explainability scope and fidelity: The utility attribution (Equations (33)–(35)) and weight sensitivity (Equations (37)–(40)) are exact for the linear utility form and first-order approximations for the nonlinear extensions. The attribution does not account for interactions between the quality proxy and the retrieval confidence mechanism: when fallback is triggered, Q_{obs} is computed without the RAG context, making the attribution fractions non-comparable to non-fallback runs. Furthermore, the quality factor ϕ_p^Q used in cross-provider

attribution (Section 5.8) is estimated from aggregate benchmark data and does not capture per-query-type quality variation. Operators using the explanation output for calibration decisions should be aware that the fractions represent *effective* weight magnitudes given the normalized inputs, not causal contributions to the underlying answer quality.

Provider lock-in and generalizability: All experiments use OpenAI gpt-4o-mini (generation) and text-embedding-3-small (retrieval) as the sole API backends (Section 6.2). The reported QACE, win rates, and utility calibrations are specific to that provider’s cost–quality–latency tradeoffs. The cross-provider normalisation framework (Section 5.8) provides a theoretically grounded mechanism for extending the results to other providers, but the quality factors ϕ_p^Q estimated from Chatbot Arena or LLM-Judge evaluations carry their own measurement error: Arena ELO ratings have 95% confidence intervals of ± 30 –50 ELO points, translating to quality factor uncertainty of $\approx \pm 0.05$ –0.10. Re-running the benchmark on a heterogeneous provider set would substantially improve the generalizability of the efficiency claims.

Workload stationarity: The EMA telemetry and fixed catalog design assume a stationary, light-tailed query distribution. Production deployments may exhibit Zipf-distributed query-type frequencies ($s > 1$), causing systematic underestimation of quality for rare but high-value bundles. Concept drift—shifts in the underlying complexity distribution—further degrades telemetry estimates with lag $\approx 1/\alpha$ observations. The CUSUM-based WorkloadTracker and importance-weighted EMA correction (Section 5.7) partially address these failure modes, but the sensitivity threshold h must be calibrated for the specific production environment; universal defaults are not available. In the benchmark evaluation, the workload is sampled uniformly, so this limitation does not affect the reported results but is material for deployment.

Bundle discreteness: Retrieval depth is limited to four discrete values; continuous or finer-grained search may find better operating points.

API dependence: Token costs and latency are tied to OpenAI pricing and infrastructure, which evolve over time. The framework generalizes to other providers, but cost comparisons require re-benchmarking.

Statistical power: With $n = 28$ queries, per-query variance is high. Bootstrap confidence intervals and paired significance tests are warranted for production-scale studies.

10. Conclusions

This paper introduced CA-RAG, a per-query routing framework that selects retrieval depth from a discrete bundle catalog by maximizing a scalar utility combining quality, latency, and token cost. Rather than treating retrieval depth as a fixed architectural decision, CA-RAG frames it as an operational choice made anew for every query adjustable via weight configuration without altering the underlying system.

Empirical results confirm the case for adaptive routing. The default router reduces billed token cost by 26% relative to always-heavy retrieval and cuts mean latency by 34% relative to always-direct inference, while maintaining quality parity. Per-query delta analysis confirms these gains are systematic and concentrated where they should be: on simpler queries where deep retrieval adds cost but not insight. The extended utility formulation (Equation (22)) further addresses the limitations of a static linear scalarization by introducing context-conditional weights, a concave quality transform, a logistic deadline penalty, and a non-compensatory quality floor—each independently enabling deployments with interactive latency requirements, high-stakes quality constraints, or complexity-dependent quality valuation.

Two findings stand out. The heuristic complexity signal is a weak predictor of actual retrieval need ($r = 0.15$), pointing toward richer signals—entity density, intent classification,

embedding-space novelty—as the most impactful near-term improvements. The bimodal retrieval confidence distribution reveals that a subset of queries suffers from corpus coverage gaps rather than routing failures, a distinction that matters for production diagnosis.

CA-RAG demonstrates that transparent routing over a fixed bundle catalog can recover meaningful cost and latency savings without sacrificing answer quality. The current implementation makes several deliberate simplifications: quality priors are hand-specified, the quality proxy is lexical, and the evaluation corpus is compact. Whether these design choices hold under learned reward models, bandit-based recalibration, and domain-specific query distributions remains an open question that we identify as direct follow-up work. The framework is designed to accommodate this evolution. Bundle catalogs, utility weights, and telemetry schemas are independently configurable, so individual components can be upgraded without restructuring the routing pipeline. As LLM API costs become a first-class engineering concern in production deployments, applying uniform retrieval depth across all queries is an avoidable inefficiency. CA-RAG provides a principled and auditable mechanism for eliminating that inefficiency one query at a time.

Author Contributions: Conceptualization, S.M. and G.R.N.; methodology, S.M.; software, S.M.; validation, S.M. and G.R.N.; formal analysis, S.M.; investigation, S.M.; resources, S.M.; data curation, S.M.; writing—original draft preparation, S.M.; writing—review and editing, S.M. and G.R.N.; visualization, S.M.; supervision, G.R.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The benchmark corpus, question set, routing logs, and figure generation scripts used in this study are publicly available at <https://github.com/sanmish4ds/ca-rag>. Authoritative CSV snapshots for this manuscript are archived under [ai-4280908/data/](https://zenodo.org/record/4280908/files/data/).

Acknowledgments: The authors thank the open-source contributors of FAISS and the MDPI editorial team for formatting guidance.

Conflicts of Interest: Author Sanjay Mishra is employed by Fidelity Investments; however, this research was conducted independently and is not related to his work or responsibilities at his employer. No funding was received from Fidelity Investments or any other commercial or financial entity in connection with this study, and his employer had no involvement in the study design, data collection, analysis, interpretation, or decision to publish. The remaining author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Appendix A. Algorithmic Summary

CA-RAG routing is summarized as a discrete utility-maximization procedure:

1. Compute query signals $s \leftarrow \text{signals}(q)$ and complexity $c \in [0, 1]$.
2. For each bundle $b \in \mathcal{B}$, compute estimated utility U_b via Equation (10).
3. Select $b^* = \arg \max_b U_b$ (optionally with ϵ -greedy exploration).
4. Retrieve context C using the retrieval specification of b^* ; generate answer a using the shared generation spec.
5. Log $(\tau_{\text{billed}}, \text{latency}, \text{quality proxy}, \tilde{U})$; optionally update telemetry priors.

Appendix B. Reproducible Commands

The benchmark corpus, question set, routing logs, and figure generation scripts used in this study are publicly available at <https://github.com/sanmish4ds/ca-rag> (accessed on

1 July 2026). All reported results can be reproduced by running the commands documented in the repository README.

Appendix C. Benchmark Question Set

What is RAG?

Why is token cost important?

How does latency affect AI systems?

What is adaptive retrieval?

Explain cost-aware AI systems.

What is hybrid retrieval?

Define utility-based routing.

What is FAISS used for?

How do strategy bundles work in CA-RAG?

What is retrieval confidence?

Compare light versus heavy retrieval for long documents.

Explain how telemetry refines routing estimates with concrete steps.

Why might a system skip retrieval for some queries?

List tradeoffs between large top-k and small top-k retrieval.

How do embedding tokens differ from completion tokens in billing?

Describe a municipal RAG use case with forms and citations.

What are the risks of fixed retrieval depth across heterogeneous queries?

How does CA-RAG combine quality, latency, and cost in one scalar objective?

Explain when reranking is worth the extra latency in production.

Derive an intuitive explanation of why discrete bundles are used instead of continuous search.

What operational metrics should a team report for a deployed RAG service?

How does query length influence estimated complexity signals in CA-RAG?

Contrast direct LLM answers with retrieval-grounded answers for policy questions.

What limitations apply to lexical quality proxies versus human evaluation?

How would you tune utility weights for a latency-sensitive chatbot?

Describe an experiment protocol to log strategy choices and token usage per query.

What is the role of exploration epsilon in bundle selection?

Explain retrieval-augmented generation for knowledge-intensive tasks in two sentences.

Appendix D. Benchmark Corpus

RAG improves LLM accuracy by retrieving relevant documents before generation.

Token cost is a major concern because embedding and completion APIs bill per token.

Latency depends on retrieval time, reranking, and model inference time under load.

Adaptive systems dynamically select strategies based on query complexity and observed telemetry.

Cost-aware AI systems optimize resource usage while maintaining answer quality under SLO constraints.

Hybrid dense--sparse retrieval combines embedding similarity with BM25 lexical overlap for robustness.

Utility-based routing scores each strategy bundle using quality priors minus latency and cost penalties.

Municipal RAG applications ground answers in ordinances, forms, and public documents with provenance.

Production RAG should expose retrieval confidence and source citations for auditability and trust.

Embedding indexes such as FAISS enable approximate nearest neighbor search over chunked corpora.

Strategy bundles pair retrieval depth with generation budgets to trade accuracy against spend.

Telemetry can refine latency and quality estimates per bundle after sufficient query volume.

Skipping retrieval reduces cost for definitional queries but risks hallucination on fact-heavy tasks.

Large top-k retrieval increases recall but inflates prompt tokens and end-to-end latency.

Reranking stages reorder candidates using cross-encoders at extra compute cost.

Appendix E. Logged CSV Schema

```

query           : input question text
strategy        : selected retrieval strategy name
bundle          : bundle name (retrieval+generation composite key)
utility         : selection utility (prior-based, Equation (1))
quality_proxy   : observed lexical overlap proxy in [0,1]
realized_utility : post hoc utility computed from observed metrics
latency         : total end-to-end latency (ms)
cost            : total billed tokens (prompt+completion+embedding)
prompt_tokens   : LLM prompt token count
completion_tokens : LLM completion token count
embedding_tokens : query embedding tokens
retrieval_confidence: max retrieved chunk cosine similarity
complexity_score : heuristic complexity signal in [0,1]
index_embedding_tokens: offline corpus-embed tokens (bookkeeping)

```

Appendix F. Per-Query Strategy Assignments

Assignments taken verbatim from generated/verbatim_query_strategy.txt (direct_llm: 4, light_rag: 8, medium_rag: 10, heavy_rag: 6).

```

1. What is RAG?                => direct_llm
2. Why is token cost important? => direct_llm
3. How does latency affect AI systems? => light_rag
4. What is adaptive retrieval?  => light_rag
5. Explain cost-aware AI systems. => medium_rag
6. What is hybrid retrieval?    => medium_rag
7. Define utility-based routing. => medium_rag
8. What is FAISS used for?      => heavy_rag
9. How do strategy bundles work in CA-RAG? => heavy_rag
10. What is retrieval confidence? => medium_rag
11. Compare light versus heavy retrieval for long docs. => medium_rag
12. Explain how telemetry refines routing estimates => light_rag
13. Why might a system skip retrieval for some queries? => medium_rag
14. List tradeoffs between large top-k and small top-k => medium_rag
15. How do embedding tokens differ from completion? => heavy_rag
16. Describe a municipal RAG use case => heavy_rag

```

17. What are risks of fixed retrieval depth?	=> light_rag
18. How does CA-RAG combine quality, latency, and cost?	=> direct_llm
19. Explain when reranking is worth extra latency	=> light_rag
20. Derive intuitive explanation of discrete bundles	=> heavy_rag
21. What operational metrics should teams report?	=> light_rag
22. How does query length influence complexity signals?	=> light_rag
23. Contrast direct vs. retrieval-grounded answers	=> medium_rag
24. Limitations of lexical quality proxies?	=> direct_llm
25. How to tune weights for latency-sensitive chatbot?	=> light_rag
26. Describe experiment protocol for token logging	=> heavy_rag
27. Role of exploration epsilon in bundle selection	=> medium_rag
28. Explain RAG for knowledge-intensive tasks	=> medium_rag

Appendix G. Sample CSV Rows

```

query,strategy,cost,latency,utility,quality_proxy,realized_utility
What is RAG?,direct_llm,185,4051.1,0.4043,0.55,-1.2461
How does latency affect AI systems?,light_rag,165,2850.3,0.3813,0.64,
-0.6282
What is hybrid retrieval?,medium_rag,179,2468.0,0.3672,0.70,-0.4028
What is FAISS used for?,heavy_rag,284,2245.5,0.2809,0.70,-0.3189

```

References

- Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.t.; Rocktäschel, T.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*; Curran Associates Inc.: Red Hook, NY, USA, 2020; Volume 33, pp. 9459–9474.
- Gao, Y.; Xiong, Y.; Gao, X.; Jia, J.; Pan, J.; Bi, Y.; Dai, Y.; Sun, J.; Wang, H. Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv* **2023**, arXiv:2312.10997.
- Izacard, G.; Grave, E. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. *arXiv* **2021**, arXiv:2007.01282.
- OpenAI. GPT-4 Technical Report. *arXiv* **2024**, arXiv:2303.08774.
- Karpukhin, V.; Oguz, B.; Min, S.; Lewis, P.; Wu, L.; Edunov, S.; Chen, D.; Yih, W.t. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2020; pp. 6769–6781.
- Khattab, O.; Zaharia, M. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*; Association for Computing Machinery: New York, NY, USA, 2020; pp. 39–48.
- Johnson, J.; Douze, M.; Jégou, H. Billion-Scale Similarity Search with GPUs. *IEEE Trans. Big Data* **2021**, *7*, 535–547.
- Guu, K.; Lee, K.; Tung, Z.; Pasupat, P.; Chang, M. REALM: Retrieval-Augmented Language Model Pre-Training. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*; JMLR: Norfolk, MA, USA, 2020; pp. 3929–3938.
- Izacard, G.; Lewis, P.; Lomeli, M.; Hosseini, L.; Petroni, F.; Schick, T.; Dwivedi-Yu, J.; Joulin, A.; Riedel, S.; Grave, E. Few-Shot Learning with Retrieval Augmented Language Models. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*; JMLR: Norfolk, MA, USA, 2022; Volume 35.
- Borgeaud, S.; Mensch, A.; Hoffmann, J.; Cai, T.; Rutherford, E.; Millican, K.; Van Den Driessche, G.; Lespiau, J.B.; Damoc, B.; Clark, A.; et al. Improving Language Models by Retrieving from Trillions of Tokens. In *Proceedings of the International Conference on Machine Learning (ICML)*; PMLR: Cambridge, MA, USA, 2022.
- Khandelwal, U.; Levy, O.; Jurafsky, D.; Zettlemoyer, L.; Lewis, M. Generalization through Memorization: Nearest Neighbor Language Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Addis Ababa, Ethiopia, 26 April–1 May 2020.
- Shi, W.; Min, S.; Yasunaga, M.; Seo, M.; James, R.; Lewis, M.; Yih, W.t.; Zettlemoyer, L. REPLUG: Retrieval-Augmented Black-Box Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 8371–8384.

13. Ram, O.; Levine, Y.; Dalmedigos, I.; Muhlga, D.; Shashua, A.; Leyton-Brown, K.; Shoham, Y. In-Context Retrieval-Augmented Language Models. In *Proceedings of the Transactions of the Association for Computational Linguistics*; MIT Press: Cambridge, MA, USA, 2023; Volume 11, pp. 1316–1331.
14. Lin, X.V.; Chen, X.; Chen, M.; Shi, W.; Lomeli, M.; James, R.; Yih, W.t.; Lewis, M.; Kiela, D.; Yatskar, M. RA-DIT: Retrieval-Augmented Dual Instruction Tuning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Vienna, Austria, 7–11 May 2024.
15. Robertson, S.; Zaragoza, H. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* **2009**, *3*, 333–389.
16. Formal, T.; Piwowarski, B.; Clinchant, S. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*; Association for Computing Machinery: New York, NY, USA, 2021; pp. 2289–2293.
17. Santhanam, K.; Khatlab, O.; Saad-Falcon, J.; Potts, C.; Zaharia, M. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2022; pp. 3715–3734.
18. Gao, L.; Ma, X.; Lin, J.; Callan, J. Precise Zero-Shot Dense Retrieval without Relevance Labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 1762–1777.
19. Malkov, Y.A.; Yashunin, D.A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2020**, *42*, 824–836.
20. Jégou, H.; Douze, M.; Schmid, C. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* **2011**, *33*, 117–128.
21. Ni, J.; Qu, C.; Lu, Z.; Dai, Y.; Chang, M.; Wen, J.R.; Liu, Y. Recent Advances in Dense Passage Retrieval: A Survey on Methods and Developments. *arXiv* **2022**, arXiv:2204.04909.
22. Thakur, N.; Reimers, N.; Rücklé, A.; Srivastava, A.; Gurevych, I. BEIR: A Heterogeneous Benchmark for Zero-Shot Evaluation of Information Retrieval Models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks)*, Online, 7–14 December 2021.
23. Asai, A.; Wu, Z.; Wang, Y.; Sil, A.; Hajishirzi, H. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Vienna, Austria, 7–11 May 2024.
24. Jiang, Z.; Xu, F.F.; Gao, L.; Sun, Z.; Liu, Q.; Dwivedi-Yu, J.; Yang, Y.; Callan, J.; Neubig, G. Active Retrieval Augmented Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 7969–7992.
25. Su, Y.; Wang, Y.; Han, C.; Cui, Z.; Wang, J.; Liu, H.; Wen, J.R. DRAGIN: Dynamic Retrieval Augmented Generation based on the Information Needs of Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 12991–13007.
26. Trivedi, H.; Balasubramanian, N.; Khot, T.; Sabharwal, A. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 10014–10037.
27. Yan, S.Q.; Gu, J.C.; Zhu, Y.; Ling, Z.H. Corrective Retrieval Augmented Generation. *arXiv* **2024**, arXiv:2401.15884.
28. Jeong, S.; Baek, J.; Cho, S.; Hwang, S.J.; Park, J.C. Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2024; pp. 7036–7050.
29. Mallen, A.; Akari, A.; Zhong, V.; Dasigi, P.; Khashabi, D.; Hajishirzi, H. When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 9802–9822.
30. Yang, Z.; Qi, P.; Zhang, S.; Bengio, Y.; Cohen, W.W.; Salakhutdinov, R.; Manning, C.D. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2018; pp. 2369–2380.
31. Kwiatkowski, T.; Palomaki, J.; Ringel, O.; Collins, M.; Parikh, A.; Alberti, C.; Epstein, D.; Polosukhin, I.; Devlin, J.; Lee, K.; et al. Natural Questions: A Benchmark for Question Answering Research. *Trans. Assoc. Comput. Linguist.* **2019**, *7*, 452–466.
32. Bajaj, P.; Campos, D.; Craswell, N.; Deng, L.; Gao, J.; Liu, X.; Majumder, R.; McNamara, A.; Mitra, B.; Nguyen, T.; et al. MS MARCO: A Human Generated Machine Reading Comprehension Dataset. In *Proceedings of the Workshop on Machine Reading for Question Answering at the Conference on Neural Information Processing Systems (NeurIPS)*, Montreal, QC, Canada, 3–8 December 2018.
33. Press, O.; Zhang, M.; Min, S.; Schmidt, L.; Smith, N.A.; Lewis, M. Measuring and Narrowing the Compositionality Gap in Language Models. In *Proceedings of the Findings of the Association for Computational Linguistics: EMNLP*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 5687–5711.

34. Wang, Z.; Tang, S.; Chen, J.; Zhao, X.; Sun, J.; Zhao, W.X.; Wen, J.R. A Survey on RAG Meets LLMs: Towards Retrieval-Augmented Large Language Models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*; Association for Computing Machinery: New York, NY, USA, 2024.
35. Chen, L.; Zaharia, M.; Zou, J. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Vienna, Austria, 7–11 May 2024.
36. Ong, I.; Almahairi, A.; Wu, V.; Chiang, W.L.; Wu, T.; Gonzalez, J.E.; Kadous, M.W.; Stoica, I. RouteLLM: Learning to Route LLMs with Preference Data. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Singapore, 24–28 April 2025.
37. Chiang, W.L.; Zheng, L.; Sheng, Y.; Angelopoulos, A.N.; Li, T.; Li, D.; Zhu, B.; Jordan, M.; Gonzalez, J.E.; Stoica, I.; et al. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*; JMLR: Norfolk, MA, USA, 2024.
38. Jiang, H.; Wu, Q.; Lin, C.Y.; Yang, Y.; Qiu, L. LLMingua: Compressing Prompts for Accelerated Inference of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2023; pp. 13358–13376.
39. Li, L.; Chu, W.; Langford, J.; Schapire, R.E. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*; Association for Computing Machinery: New York, NY, USA, 2010; pp. 661–670.
40. Lattimore, T.; Szepesvári, C. *Bandit Algorithms*; Cambridge University Press: Cambridge, UK, 2020.
41. Hu, Y.; Liu, X.; Zhao, J.; Yin, S.; Cheng, Y. RAGRouter: Leveraging Inherent Routing Knowledge of Large Language Models for RAG. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Miami, FL, USA, 12–16 November 2024.
42. Gersho, A.; Gray, R.M. *Vector Quantization and Signal Compression*; Kluwer Academic Publishers: Dordrecht, The Netherlands, 1992.
43. Varian, H.R. *Microeconomic Analysis*, 3rd ed.; W. W. Norton & Company: New York, NY, USA, 1992.
44. Belton, V.; Stewart, T.J. *Multiple Criteria Decision Analysis: An Integrated Approach*; Kluwer Academic Publishers: Dordrecht, The Netherlands, 2002.
45. Zador, P.L. Asymptotic quantization error of continuous signals and the quantization dimension. *IEEE Trans. Inf. Theory* **1982**, *28*, 139–149.
46. Hwang, C.L.; Yoon, K. *Multiple Attribute Decision Making: Methods and Applications A State-of-the-Art Survey*; Lecture Notes in Economics and Mathematical Systems; Springer: Berlin/Heidelberg, Germany, 1981; Volume 186.
47. Lloyd, S.P. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *28*, 129–137.
48. Zheng, L.; Chiang, W.L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.P.; et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*; Curran Associates Inc.: Red Hook, NY, USA, 2023; Volume 36.
49. Keeney, R.L.; Raiffa, H. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*; John Wiley & Sons: Hoboken, NJ, USA, 1976.
50. von Neumann, J.; Morgenstern, O. *Theory of Games and Economic Behavior*; Princeton University Press: Princeton, NJ, USA, 1947.
51. Fishburn, P.C. *Utility Theory for Decision Making*; John Wiley & Sons: Hoboken, NJ, USA, 1970.
52. Geoffrion, A.M. Proper efficiency and the theory of vector maximization. *J. Math. Anal. Appl.* **1968**, *22*, 618–630.
53. Dean, J.; Barroso, L.A. The Tail at Scale. *Commun. ACM* **2013**, *56*, 74–80.
54. Reimers, N.; Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2019; pp. 3982–3992.
55. Roy, B. The Outranking Approach and the Foundations of ELECTRE Methods. *Theory Decis.* **1991**, *31*, 49–73.
56. Holt, C.C. *Forecasting Seasonals and Trends by Exponentially Weighted Moving Averages*; Technical Report ONR Memorandum No. 52; Office of Naval Research: Arlington, VA, USA, 1957.
57. Gardner, E.S. Exponential Smoothing: The State of the Art. *J. Forecast.* **1985**, *4*, 1–28.
58. Sugiyama, M.; Suzuki, T.; Kanamori, T. *Density Ratio Estimation in Machine Learning*; Cambridge University Press: Cambridge, UK, 2012.
59. Pearl, J. *Causality: Models, Reasoning, and Inference*, 2nd ed.; Cambridge University Press: Cambridge, UK, 2009.
60. Manning, C.D.; Schütze, H. *Foundations of Statistical Natural Language Processing*; MIT Press: Cambridge, MA, USA, 1999.
61. Montague, R. The Proper Treatment of Quantification in Ordinary English. In *Formal Philosophy: Selected Papers of Richard Montague*; Thomason, R.H., Ed.; Yale University Press: New Haven, CT, USA, 1974; pp. 247–270.

62. Rajpurkar, P.; Zhang, J.; Lopyrev, K.; Liang, P. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2016; pp. 2383–2392.
63. Lin, C.Y. ROUGE: A Package for Automatic Evaluation of Summaries. In *Proceedings of the Text Summarization Branches Out*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2004; pp. 74–81.
64. Zipf, G.K. *Human Behavior and the Principle of Least Effort*; Addison-Wesley: Cambridge, MA, USA, 1949.
65. Page, E.S. Continuous Inspection Schemes. *Biometrika* **1954**, *41*, 100–115.
66. Doshi-Velez, F.; Kim, B. Towards a Rigorous Science of Interpretable Machine Learning. In *Proceedings of the ICML Workshop on Human Interpretability in Machine Learning*, Sydney, Australia, 10 August 2017.
67. Lundberg, S.M.; Lee, S.I. A Unified Approach to Interpreting Model Predictions. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 4768–4777.
68. Wachter, S.; Mittelstadt, B.; Russell, C. Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR. *Harv. J. Law Technol.* **2018**, *31*, 841–887.
69. Amershi, S.; Weld, D.; Vorvoreanu, M.; Fourney, A.; Nushi, B.; Collisson, P.; Suh, J.; Iqbal, S.; Bennett, P.N.; Inkpen, K.; et al. Software Engineering for Machine Learning: A Case Study. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*; IEEE: New York, NY, USA, 2019.
70. Jaccard, P. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bull. Société Vaudoise Sci. Nat.* **1901**, *37*, 547–579.
71. Demšar, J. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.* **2006**, *7*, 1–30.
72. Shannon, C.E. A Mathematical Theory of Communication. *Bell Syst. Tech. J.* **1948**, *27*, 379–423.
73. Bowman, S.R.; Angeli, G.; Potts, C.; Manning, C.D. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2015; pp. 632–642.
74. Honnibal, M.; Montani, I. spaCy 2: Natural Language Understanding with Bloom Embeddings, Convolutional Neural Networks and Incremental Parsing. 2017. Available online: <https://spacy.io> (accessed on 24 June 2026).
75. Parasuraman, R.; Riley, V. Humans and Automation: Use, Misuse, Disuse, Abuse. *Hum. Factors* **1997**, *39*, 230–253.
76. Zhang, T.; Kishore, V.; Wu, F.; Weinberger, K.Q.; Artzi, Y. BERTScore: Evaluating Text Generation with BERT. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Addis Ababa, Ethiopia, 26 April–1 May 2020.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.