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

Sanjay Mishra * and Ganesh R. Naik

Posted Date: 16 April 2026

doi: 10.20944/preprints202604.1182.v1

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

 doi:10.20944/preprints202604.1182.v1

Article

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

Sanjay Mishra [1,*] and Ganesh R. Naik [2]

[1] IEEE Member, Fidelity Investments, Raleigh, NC, USA
[2] Associate Professor, IT & CS, Torrens University Australia, Adelaide, SA, Australia
* Correspondence: sanmish4@icloud.com

Abstract

When a large language model (LLM) answers a question using retrieved documents, a well known technique called retrieval-augmented generation (RAG) is used most of the time, while retrieving more documents improves answer accuracy but increases cost and response time, on the other hand 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 actually is. This wastes budget 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: Starts from no retrieval at all to fetching the top document-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 and the OpenAI chat and embedding application programming interfaces (APIs). We evaluate CA-RAG on a benchmark of 28 queries. The router intelligently assigns different strategies to different queries, resulting in 26% fewer billed tokens compared to always using heavy retrieval, and 34% lower response time compared to always answering directly without retrieval with excellent answer quality in both cases. Further analysis shows that most savings come from simpler queries, where heavy retrieval was never necessary to begin with. All results are reproducible from logged comma-separated values (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 the quality of answers.

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

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, 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

 doi:10.20944/preprints202604.1182.v1

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

 doi:10.20944/preprints202604.1182.v1

vector embeddings [7]), BM25-ready tokenization, OpenAI embedding/chat integration, and a CLI experiment harness.
- A 28-query empirical study with ten quantitative figures and five tables, all generated 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] and fusion-in-decoder models [3] demonstrated the value of tightly coupling retrieval with generation at scale. 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 [9] remains a competitive probabilistic baseline and is complementary to dense retrieval in hybrid pipelines. FAISS [7] enables scalable approximate nearest neighbor search over large embedding indices. CA-RAG uses FAISS as the dense backend while preserving BM25-compatible tokenization for future hybrid fusion.

2.3. RAG Surveys and Production Considerations

Gao et al. [2] survey the RAG landscape and identify retrieval granularity and cost as open research challenges. CA-RAG offers a concrete, measurable response to the cost challenge through per-query routing with full token accounting.

2.4. Contextual Bandits and Adaptive Decision-Making

CA-RAG is conceptually related to contextual bandit formulations in which queries are contexts and bundles are actions [10,11]. 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.

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 $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$. 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.

 doi:10.20944/preprints202604.1182.v1

- **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:

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

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. 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:

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

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.

4.1. 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

 doi:10.20944/preprints202604.1182.v1

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.

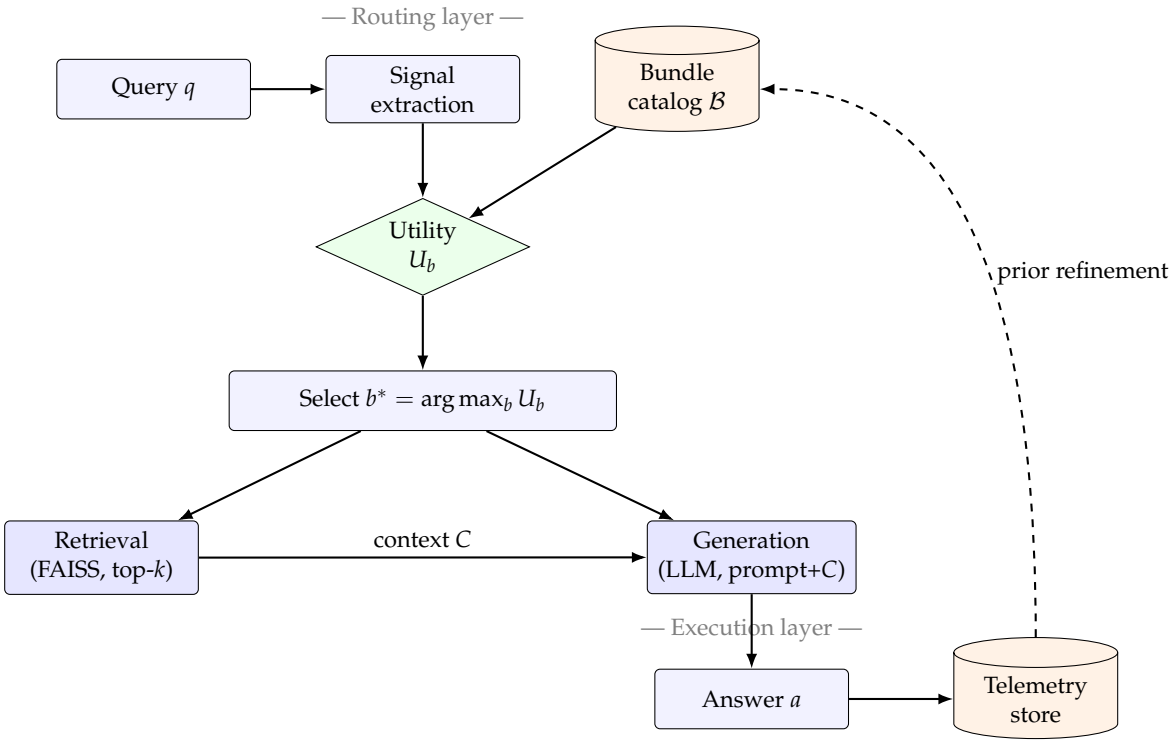

Figure 1. CA-RAG architectural block diagram. The routing layer (top) consumes query signals and the bundle catalog to select b^* via utility maximization. The execution layer (bottom) carries out the chosen retrieval depth and generation, then writes per-query metrics to the telemetry store. The dashed 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 α=0.6, β=0.4, $L_{\max}$=20, $K_{\max}$=3. Complexity modulates quality priors without requiring an additional LLM call.

5.2. Strategy Bundles

Table 1 defines the four bundles used in this study. Retrieval depth ranges from no retrieval (`direct_llm`) to top-k=10 dense retrieval (`heavy_rag`). All bundles share a common generation specification (`paper_gen`: 256 maximum output tokens, temperature 0). Priors encode expected quality, latency, and context token usage for utility estimation.

The numeric prior values in Table 1 are not learned; they are hand-specified seed values that capture the operator's initial belief about each bundle's behavior before any telemetry is observed. They are derived from three sources. First, the *quality priors* (0.52 for `direct_llm` up to 0.82 for `heavy_rag`) reflect the well-established empirical ordering reported in the open-domain QA literature: retrieval-augmented generation consistently outperforms parametric-only inference on fact-intensive queries [1,3,5], with diminishing returns past a moderate top-k [5]. The specific numeric levels are normalized to the $[0, 1]$ utility scale rather than calibrated to any external benchmark; what matters for routing is the *ordering and relative spacing*, not the absolute values. Second, the *latency priors* (8 ms to

 doi:10.20944/preprints202604.1182.v1

95 ms) are taken from short pilot runs of each bundle against the benchmark corpus, rounded to convey order-of-magnitude expectations: `direct_llm` skips retrieval entirely, while `heavy_rag` incurs both a top-10 FAISS lookup and longer prompt assembly. These priors only need to be approximately right because they enter the utility function in normalized form (Equation 3), so monotonicity across bundles is the operative property. Third, the priors are designed to be *recalibrated from telemetry* rather than fixed: once a bundle has been exercised over a sufficient number of queries, the logged `realized_utility`, `latency`, and `quality_proxy` fields can replace the seed values via a simple moving-average update. Sensitivity to these initial choices is bounded by construction, since the routing decision depends only on the rank order of U_b values across the catalog, not on their absolute magnitudes.

5.3. 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 selection utility is defined as:

$$U_b = w_Q \, \hat{Q}_b(q) - w_L \, \hat{L}_b^{\text{norm}} - w_C \, \hat{C}_b^{\text{norm}}, \tag{3}$$

where $\hat{L}_b^{\text{norm}}$ and $\hat{C}_b^{\text{norm}}$ are latency and cost normalized to $[0, 1]$ across $\mathcal{B}$, and (w_Q, w_L, w_C) are operator-specified weights. Default weights are $(0.6, 0.2, 0.2)$. The orchestrator selects $b^* = \arg\max_{b \in \mathcal{B}} U_b$.

After execution, a *realized utility* $\tilde{U}_b$ is computed by substituting observed latency and token counts into Equation (3), enabling post-hoc telemetry analysis.

5.4. Token Billing Model

Total billed tokens per query are:

$$\tau_{\text{billed}} = \tau_{\text{prompt}} + \tau_{\text{completion}} + \tau_{\text{embed}}, \tag{4}$$

where τ_{embed} attributes query-time embedding work. Offline corpus indexing is recorded separately as `index_embedding_tokens` for cost accounting completeness.

5.5. Implementation Stack

The reference implementation segments documents into line-level passages, builds a FAISS inner-product index over OpenAI `text-embedding-ada-002` embeddings, and calls the OpenAI chat API (`gpt-3.5-turbo`) with accurate per-call token counting via `tiktoken`. 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.

Table 1. 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

6. Experimental Setup

6.1. Corpus and Query Set

The evaluation uses a compact corpus of 15 sentences covering core CA-RAG concepts, including token cost, hybrid retrieval, municipal RAG, FAISS, and telemetry. The corpus is kept small by design. The goal of this benchmark is not to measure retrieval recall over a large knowledge base, but to study how the router selects bundles when queries vary in complexity and retrieval correctness is held roughly constant. A small, hand-curated corpus removes a key source of noise: when ground-truth

 doi:10.20944/preprints202604.1182.v1

passages are reliably present, the cost, latency, and quality differences observed across bundles can be attributed to routing decisions rather than to gaps in corpus coverage. This controlled setup follows standard practice in routing and bandit research, where the routing mechanism is first evaluated in a stable, well-defined environment before being tested on larger and noisier corpora. We treat generalization to production-scale corpora as a direct follow-up, discussed in Section (Section 9). A set of 28 natural-language queries span varying levels of complexity: definitional (*"What is RAG?"*), comparative (*"Compare light versus heavy retrieval for long documents"*), procedural, and analytical prompts. The corpus is embedded once; all queries share the same FAISS index.

6.2. Metrics

Per query, the following metrics are logged: selected strategy, selection utility U, total billed tokens τ_{billed}, latency (ms), lexical quality proxy (token overlap against a reference), realized utility $\tilde{U}$, retrieval confidence (max cosine similarity when retrieval runs), and complexity score c.

6.3. 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), and (vi) cost-sensitive router (w_C=0.5).

6.4. Reproducibility

With `OPENAI_API_KEY` set, the full benchmark is reproduced via:

- Router run: `ca-rag-experiment -docs data/documents_benchmark.txt -questions data/questions_benchmark.txt -out results/router_default.csv`
- Fixed baselines: `ca-rag-experiment -mode fixed -fixed-strategy heavy_rag ...`
- Figure generation: `python scripts/generate_paper_figures.py -csv results/router_default.csv -results-dir results`

The temperature is 0 for generation stability; timing variation arise from API network conditions.

Table 2. 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 57% of queries (16/28), `heavy_rag` for 18% (5/28), `direct_llm` for 14% (4/28), and `light_rag` for 11% (3/28). 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.

The dominance of `medium_rag` is consistent with the workload composition: the benchmark contains a majority of moderately complex procedural and comparative queries for which five retrieved passages provide adequate grounding without incurring the full cost of heavy retrieval. Definitional

 doi:10.20944/preprints202604.1182.v1

queries correctly route to `direct_llm`, since parametric LLM knowledge is sufficient and retrieval would add cost without any quality benefit.

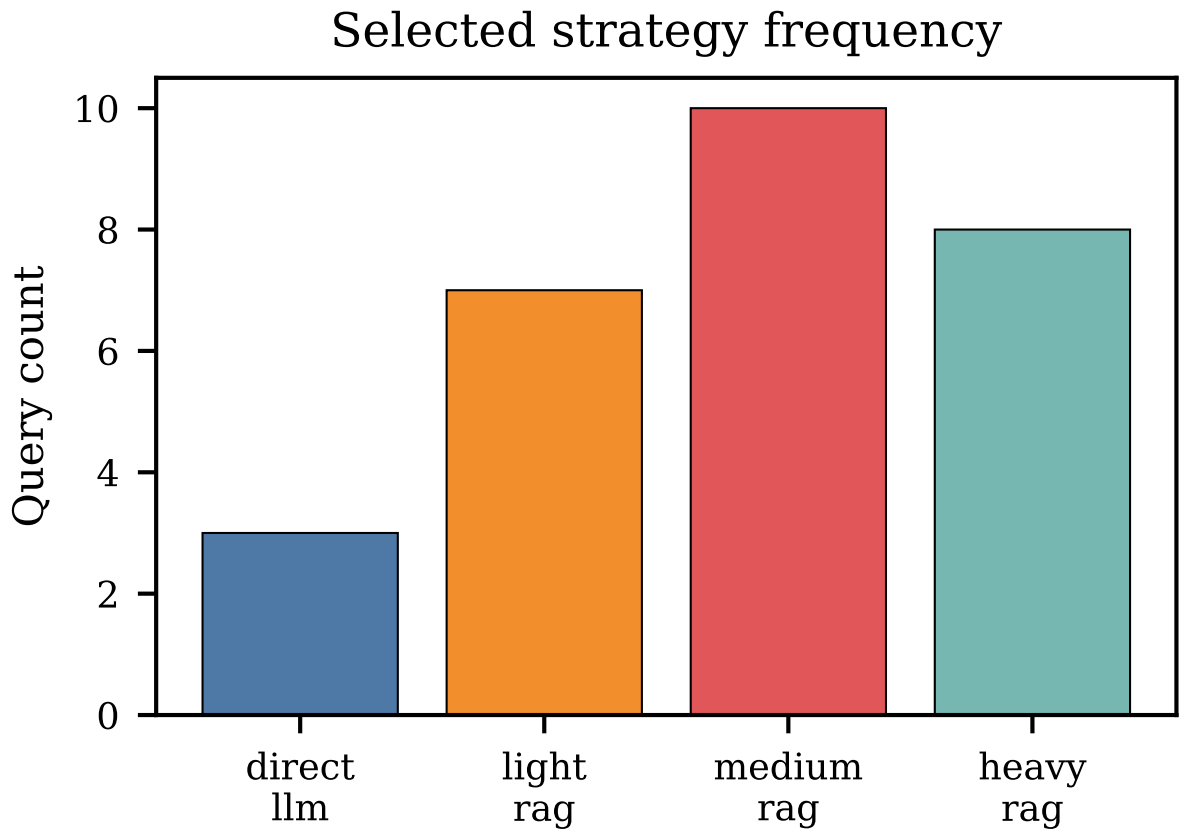

Figure 2. Bundle selection frequency across 28 benchmark queries. All four bundles are exercised, confirming genuine routing diversity.

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 (3) is actively suppressing unnecessary escalation. Third, the empirical Pareto frontier spans all four strategies, validating that each bundle occupies a distinct operating niche.

 doi:10.20944/preprints202604.1182.v1

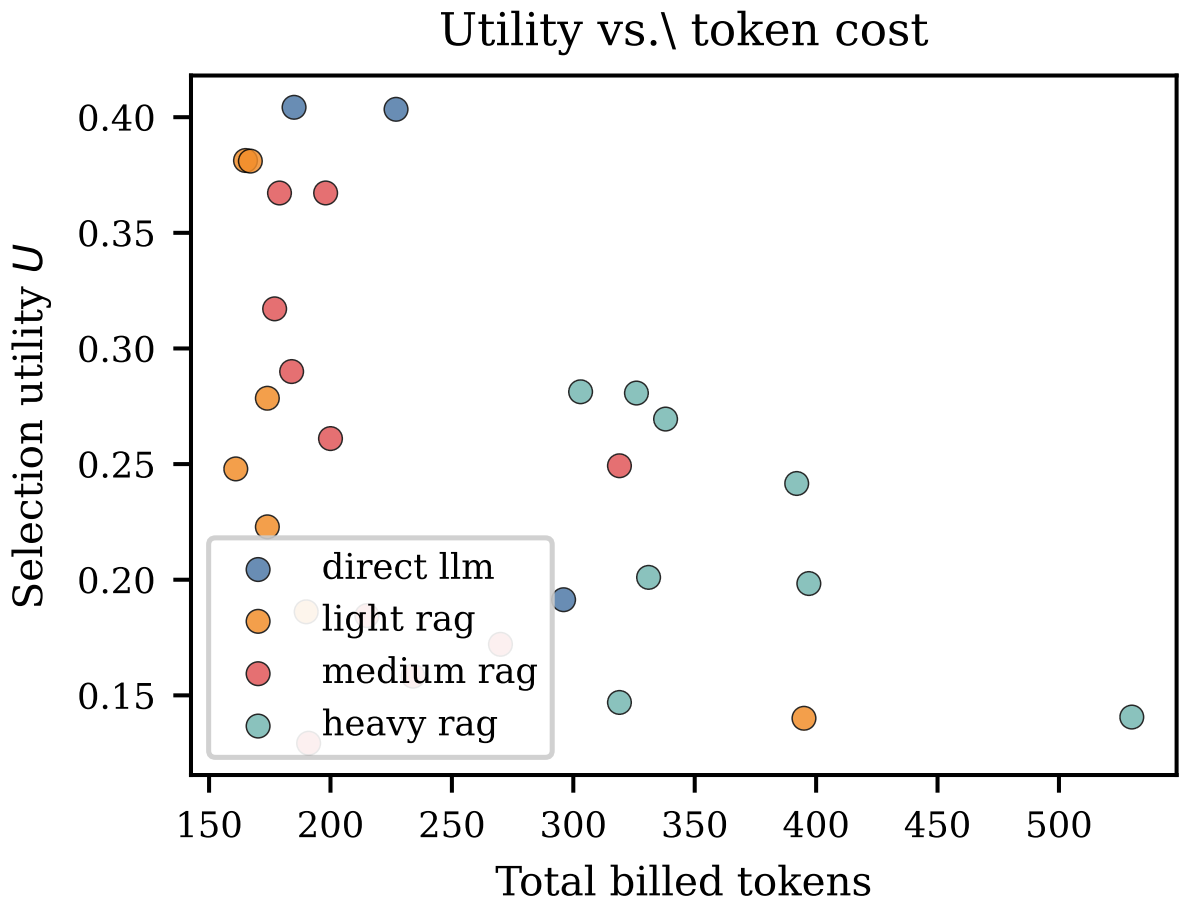

Figure 3. Selection utility vs. total billed tokens per query. Each strategy occupies a distinct region of the cost–utility space.

7.2. Latency Profiles and Token Decomposition

Figure 4 presents end-to-end latency distributions by strategy. A counterintuitive finding is that `direct_llm` exhibits the *highest* latency variance despite skipping retrieval entirely. This occurs because, without retrieval constraining prompt length, the LLM generates longer and more variable completions. Retrieval-based bundles show progressively tighter interquartile ranges because the retrieval step imposes a structured, predictable overhead before generation begins.

 doi:10.20944/preprints202604.1182.v1

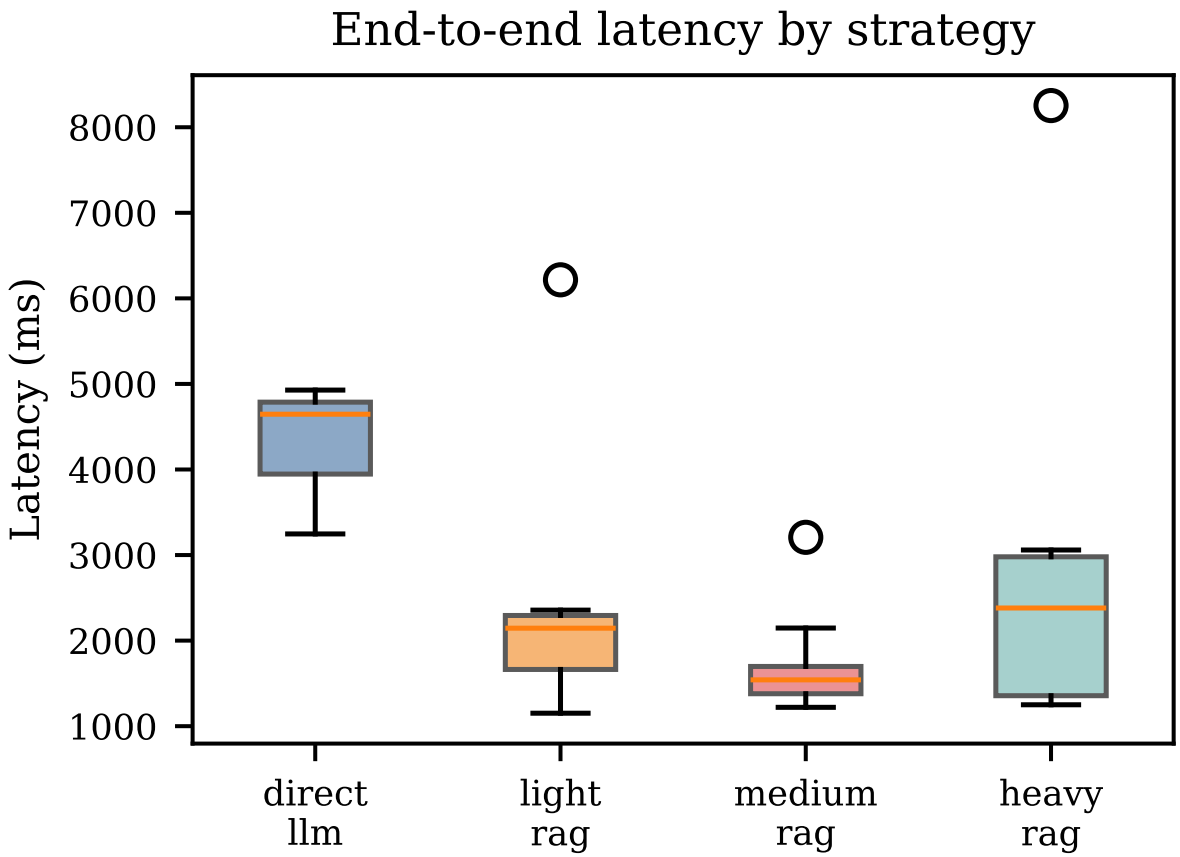

Figure 4. End-to-end latency distributions by strategy. `direct_llm` shows the highest variance; retrieval bundles exhibit tighter, more predictable profiles.

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.

 doi:10.20944/preprints202604.1182.v1

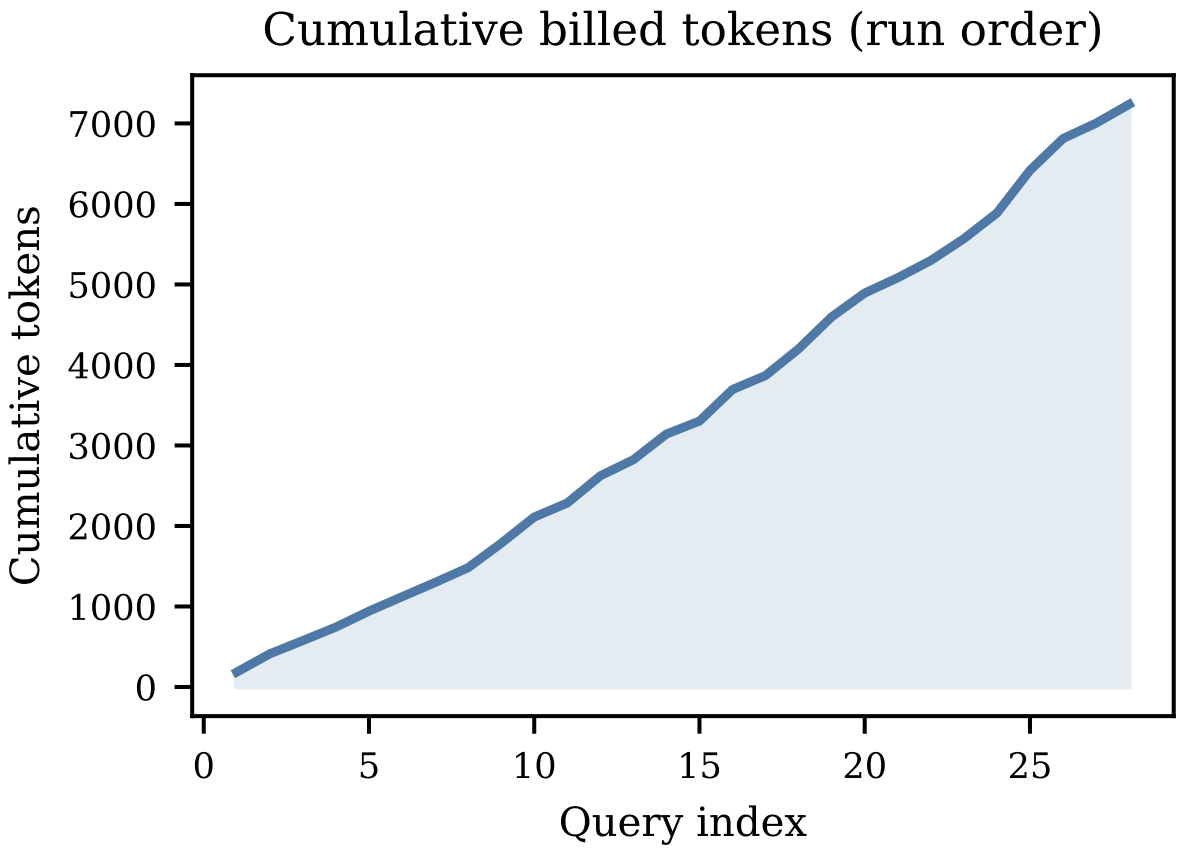

Figure 5. Cumulative billed tokens in query-log order. Slope changes correspond to strategy transitions.

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.

 doi:10.20944/preprints202604.1182.v1

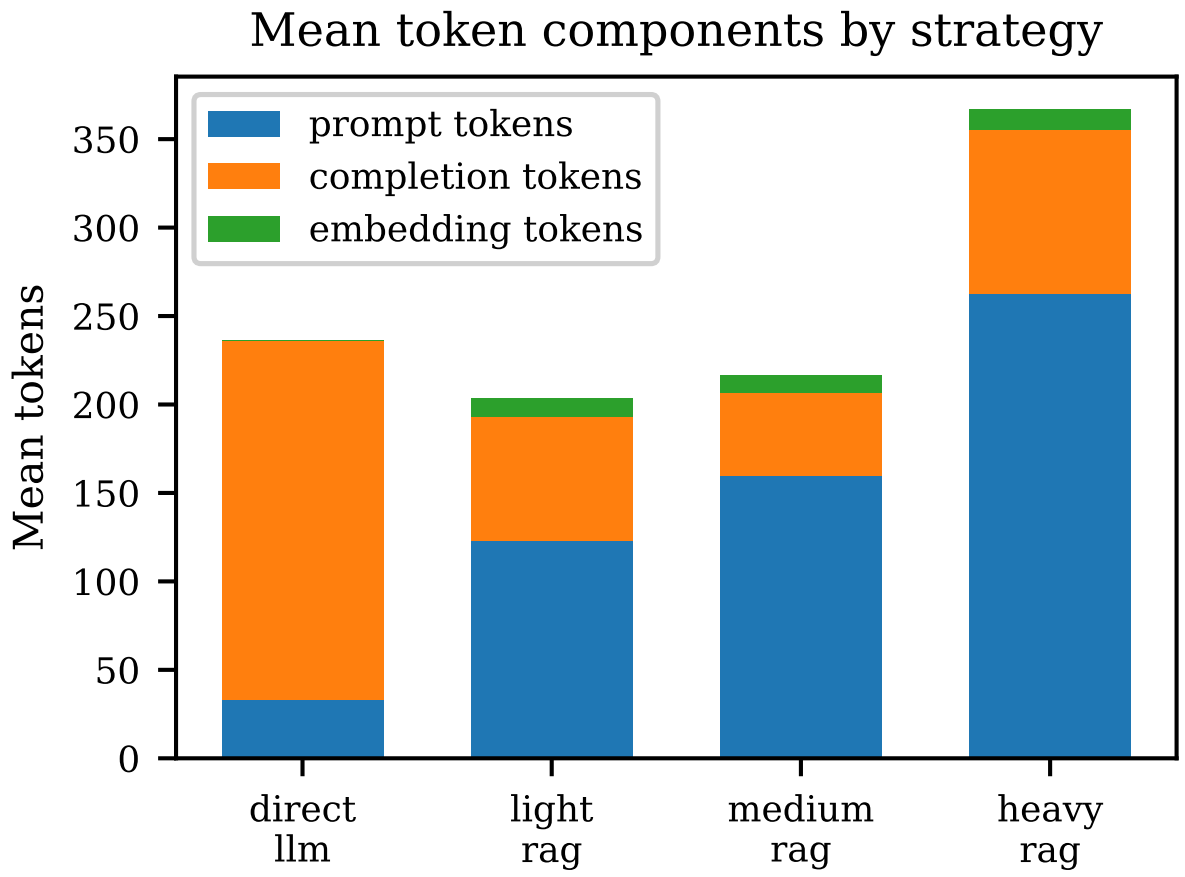

Figure 6. Mean prompt, completion, and embedding token shares by strategy. Prompt tokens scale with retrieval depth; completion tokens remain stable.

7.3. Utility Distributions and Quality Proxy Analysis

Before examining the distribution, it helps to understand what a utility score represents. For each incoming query, CA-RAG assigns a score to every available strategy bundle. This score is called utility. Utility combines three factors: how accurate the answer is expected to be, how long the response will take, and how many tokens it will consume. The bundle with the highest utility score is selected. A high utility means the router found a bundle that is both likely to produce a correct answer and cheap to run. A low utility means every available bundle carries a meaningful cost or latency penalty, and even the best option is a compromise. Figure 7 plots these winning utility scores across all 28 queries. The histogram leans to the right, which means, most of the queries receive high utility scores. This makes intuitive sense: simple queries have a clear winner among the four bundles, because one option is obviously cheaper and faster without sacrificing answer quality. The smaller group of queries on the left side of the histogram are the complex analytical ones, where every bundle is expensive and slow, and the best available score is still low. The shape of this distribution is itself a useful diagnostic: if the distribution shifts leftward over time, it signals that the workload is becoming harder or that cost penalties are growing, and the bundle catalog may need to be recalibrated.

 doi:10.20944/preprints202604.1182.v1

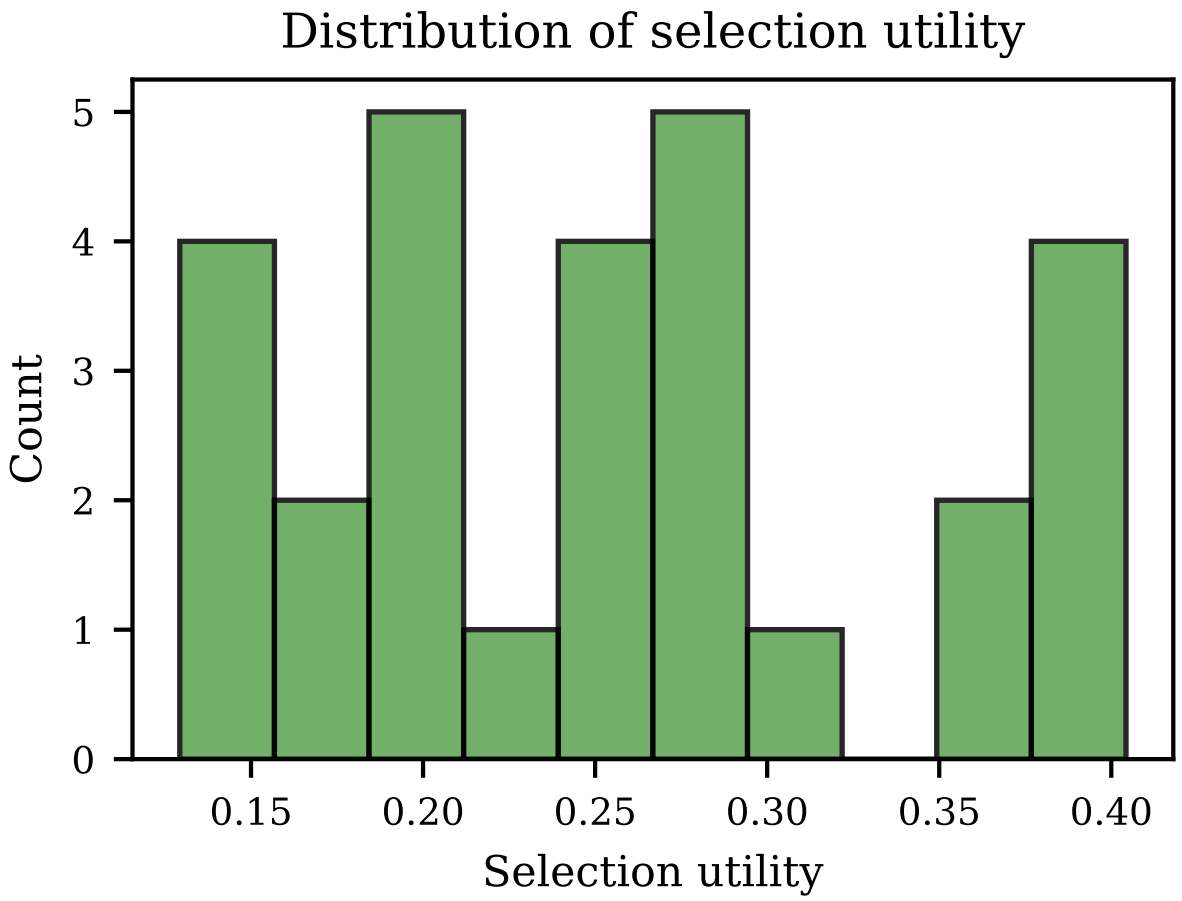

Figure 7. Histogram of per-query selection utilities. The right-skewed distribution reflects higher quality priors for simple queries where cost and latency penalties are minimal.

To evaluate actual performance, we measured the utility realized using the observed token counts and latencies (Figure 8). The `medium_rag` strategy achieves the highest median realized utility, balancing cost efficiency with answer quality. The `heavy_rag` strategy shows wider variance, performing well on some complex queries but incurring unnecessary cost on others.

 doi:10.20944/preprints202604.1182.v1

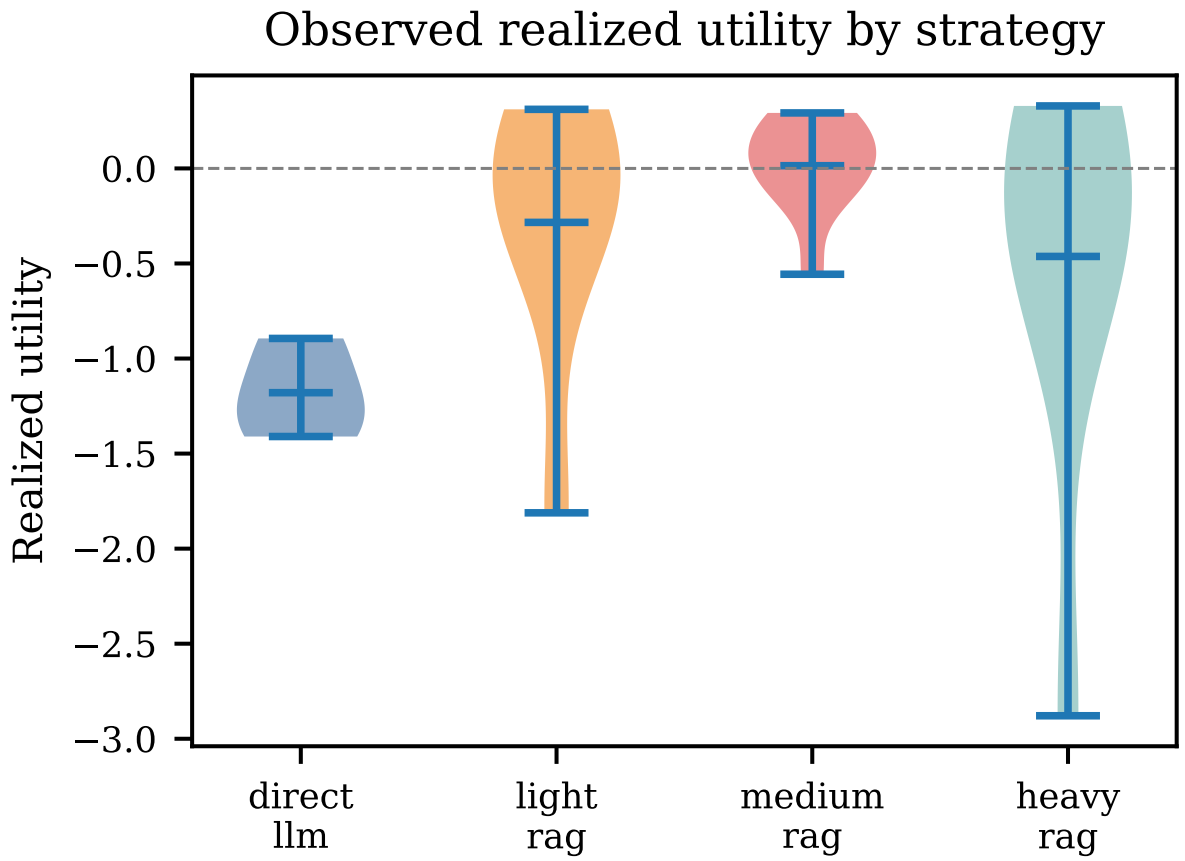

Figure 8. Realized utility distributions by strategy (post-hoc scoring with observed metrics). `medium_rag` achieves the highest and most consistent realized utility.

Figure 8 takes this analysis one step further by comparing *realized* utility across the four strategies. Predicted utility is calculated before the query runs, using estimated costs and latencies. Realized utility is calculated after the query runs, using the actual observed token counts and response times. Comparing the two reveals whether the router's predictions were accurate in practice.

The violin plot shows that `medium_rag` achieves the highest and most consistent realized utility. This confirms that medium-depth retrieval occupies the best position on the cost–quality tradeoff for this workload: it retrieves enough context to answer most queries correctly without incurring the heavy token costs of full retrieval. `heavy_rag` shows a wider spread of realized scores. This is expected because it is assigned to complex queries, and complex queries vary considerably in how expensive they turn out to be in practice, some resolve quickly, others consume many tokens.

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.

 doi:10.20944/preprints202604.1182.v1

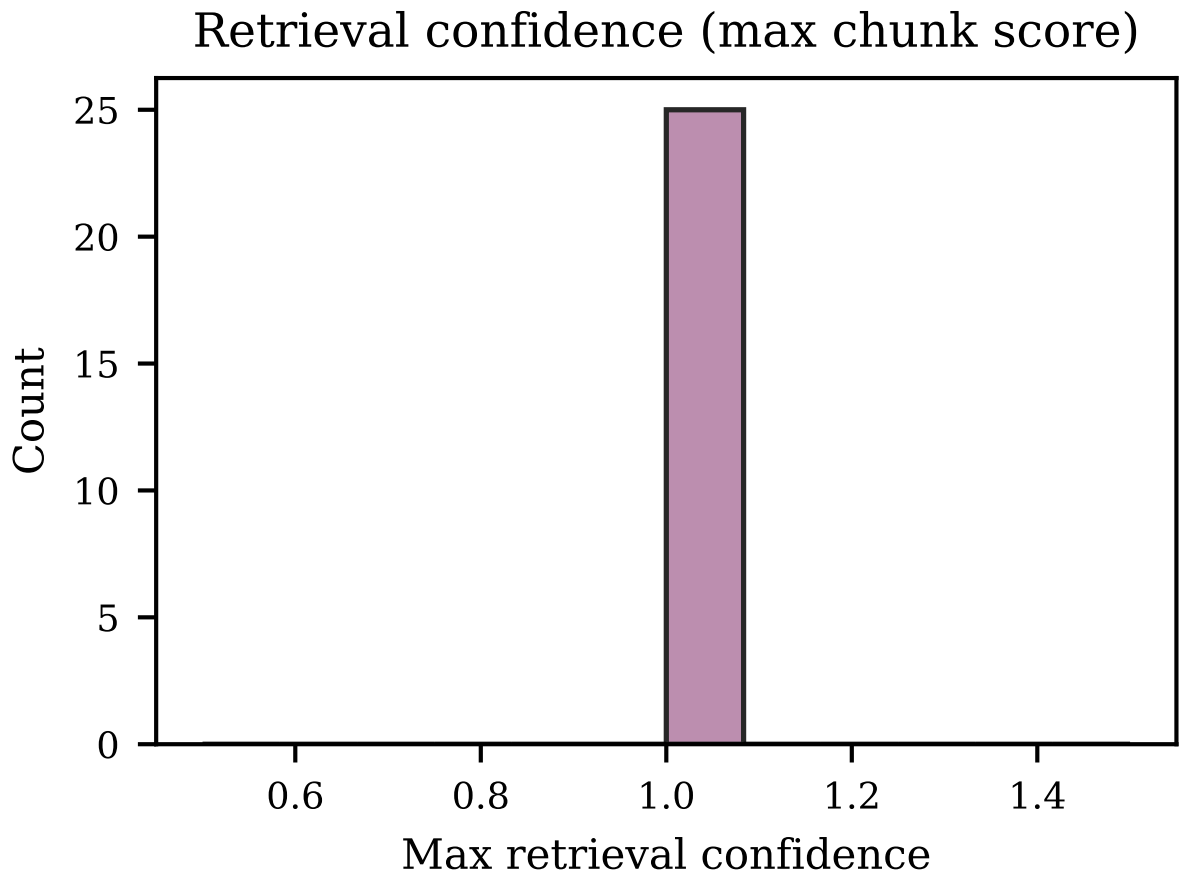

Figure 9. Max retrieval confidence scores when retrieval executes. The bimodal distribution identifies a subset of queries with poor corpus coverage.

The distribution shows two distinct clusters rather than a smooth spread. Most queries fall in the high-confidence cluster (scores above 0.85), meaning the corpus contained relevant passages and retrieval worked as intended. A second smaller cluster appears at lower confidence (scores between 0.55 and 0.70). These queries did not fail technically the retrieval mechanism ran without error, but the corpus simply did not contain closely relevant content for them. This is a corpus coverage gap, not a system fault.

This pattern has a direct practical implication for routing. When retrieval confidence falls below a threshold, injecting low-quality context into the prompt can hurt answer quality rather than help it. A straightforward improvement would be to add a confidence guardrail: if the retrieval score falls below a set threshold, the router falls back to `direct_llm` and answers from the model's own knowledge rather than from weakly matched passages. This would convert low-confidence retrievals from a liability into a clean fallback.

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=0.41), as expected since heavier bundles incur both. Selection utility correlates negatively with cost (r=−0.38), confirming that the cost penalty in Equation (3) is doing its intended work. Critically, heuristic complexity correlates only weakly with billed tokens (r=0.21), exposing a key limitation: character/word-count-based complexity does not reliably predict how much retrieval a query actually needs.

 doi:10.20944/preprints202604.1182.v1

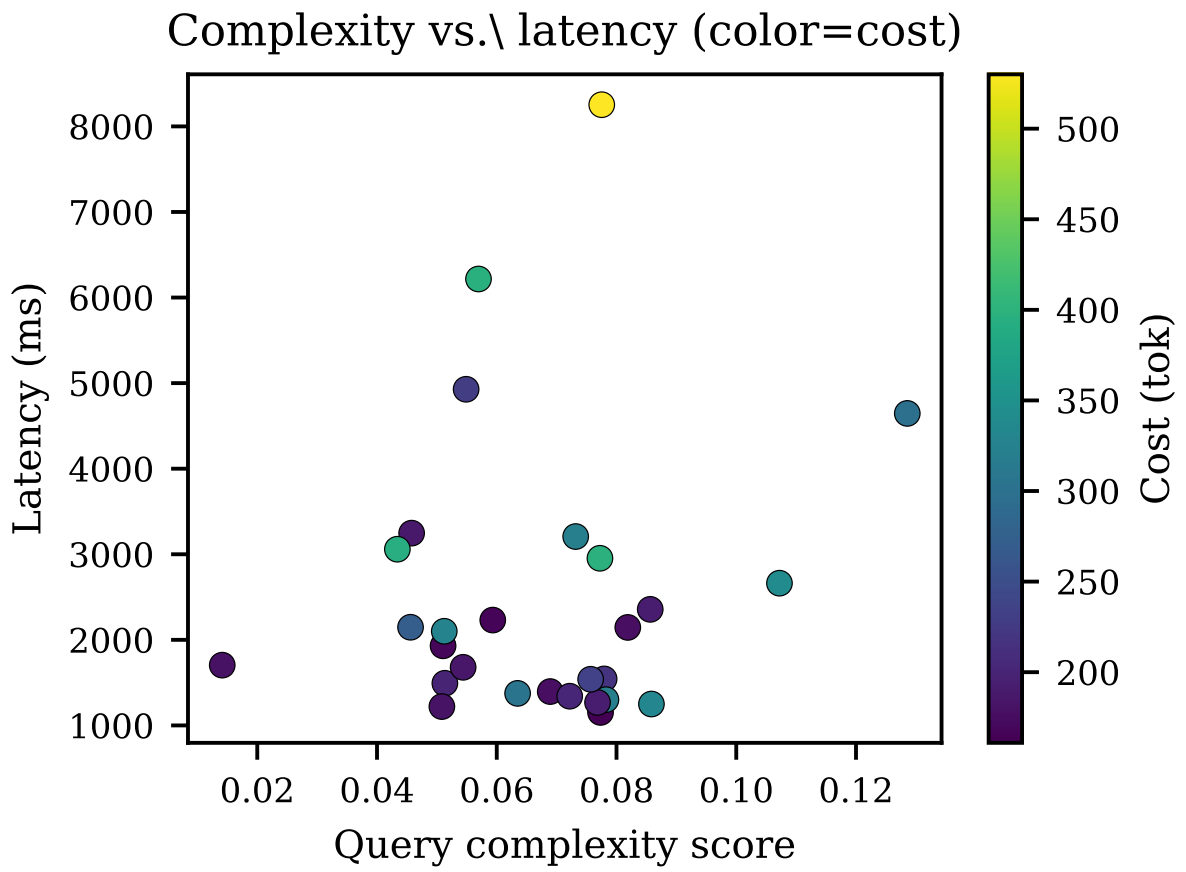

Figure 10. Heuristic query complexity vs. end-to-end latency. Marker color encodes billed tokens. Complexity is a weak predictor of cost (r=0.21).

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 spike to 300–400 tokens while `direct_llm` queries stay below 250.

 doi:10.20944/preprints202604.1182.v1

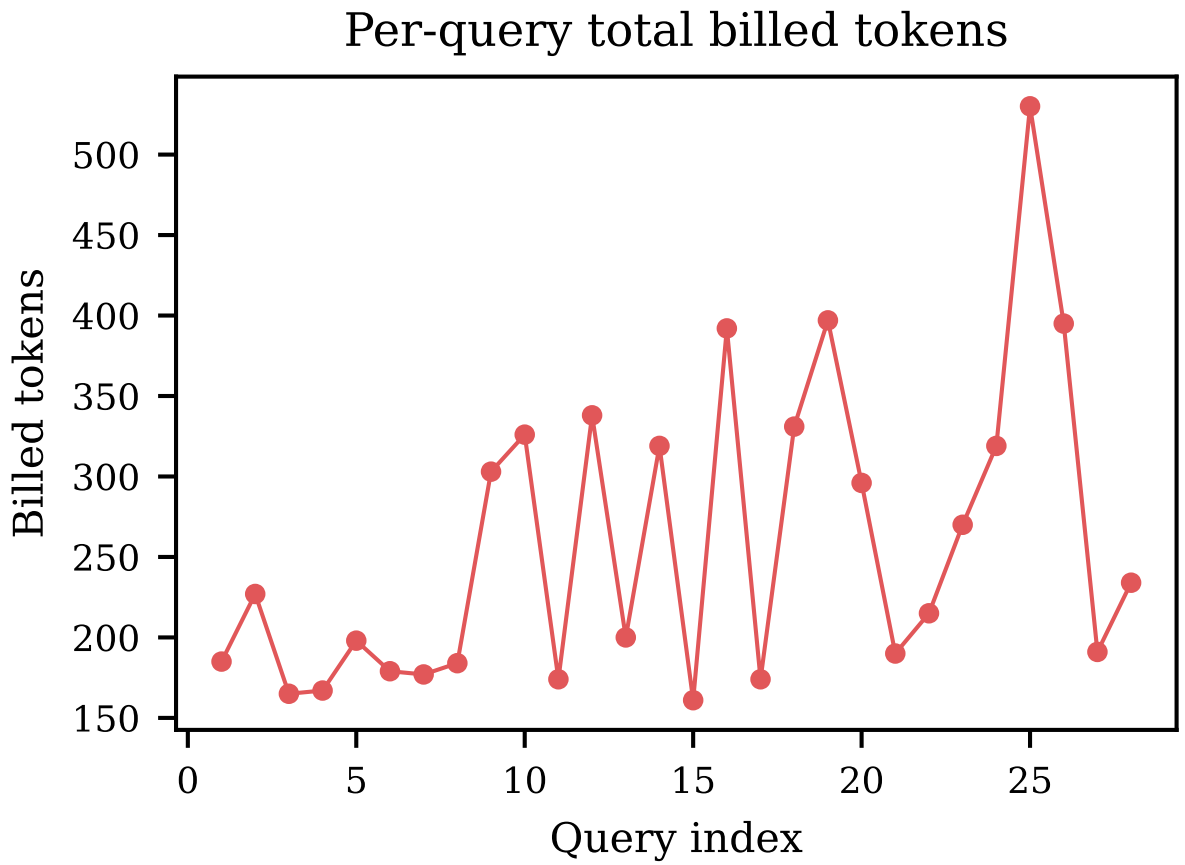

Figure 11. Total billed tokens per query in run order. Per-query cost heterogeneity motivates adaptive rather than fixed retrieval depth.

7.5. Router vs. Fixed Baselines

Table 3 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 4,457 ms, the router reduces mean latency to 2,927 ms—a **34.3% improvement**—while matching quality proxy.

Table 3. Policy-level comparison: mean billed tokens, latency (ms), lexical quality proxy, and selection utility.

Policy	cost(tok)	lat(ms)	qual.	U
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 4 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 are systematic. Against fixed-light, the router wins on quality in **0%** of queries as expected—fixed-light always retrieves, but at a higher token cost than routing on 93% of queries.

 doi:10.20944/preprints202604.1182.v1

Table 4. 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 and 14 visualize these policy comparisons on cost, latency, and quality proxy. The router occupies a favorable interior position: it achieves the best *joint* outcome across all three dimensions for this workload.

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 3 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.

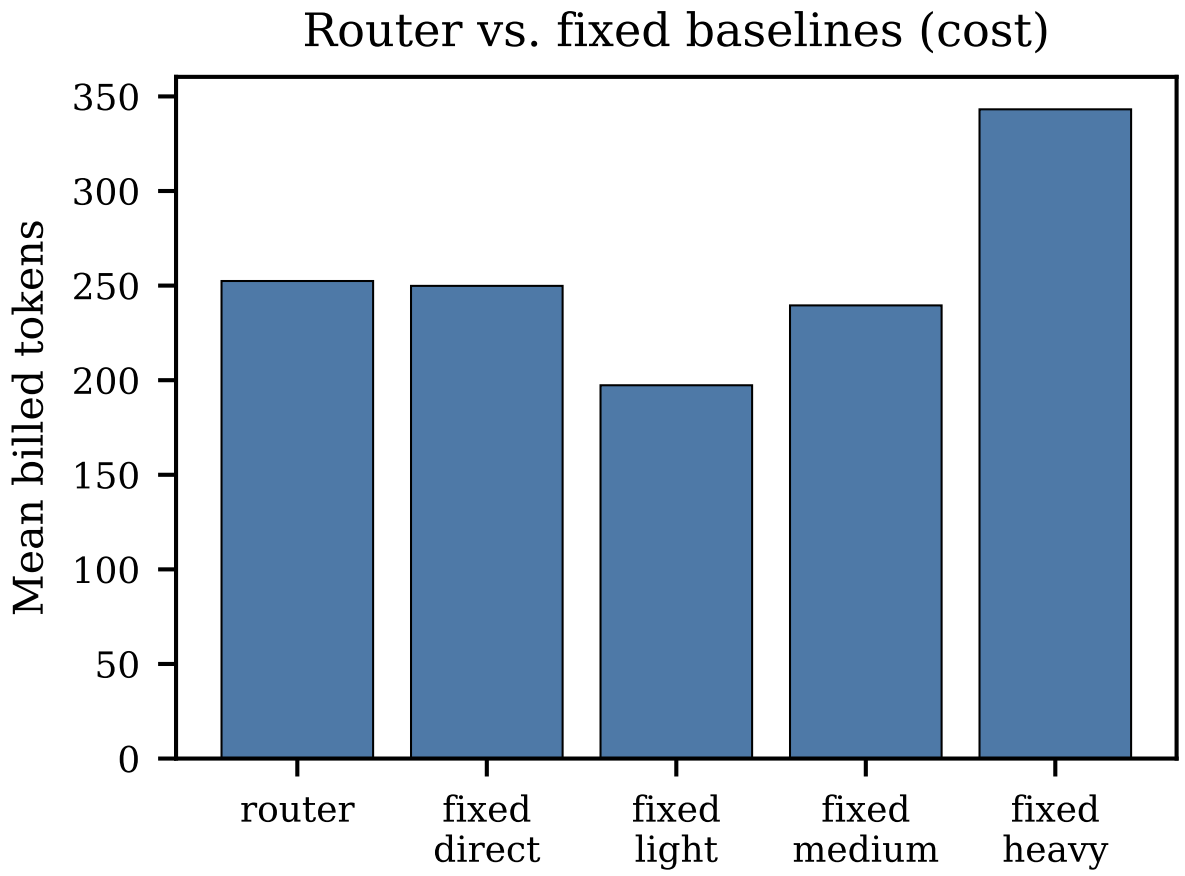

Figure 12. Mean billed tokens by policy. The router reduces cost by 26% vs. fixed-heavy.

 doi:10.20944/preprints202604.1182.v1

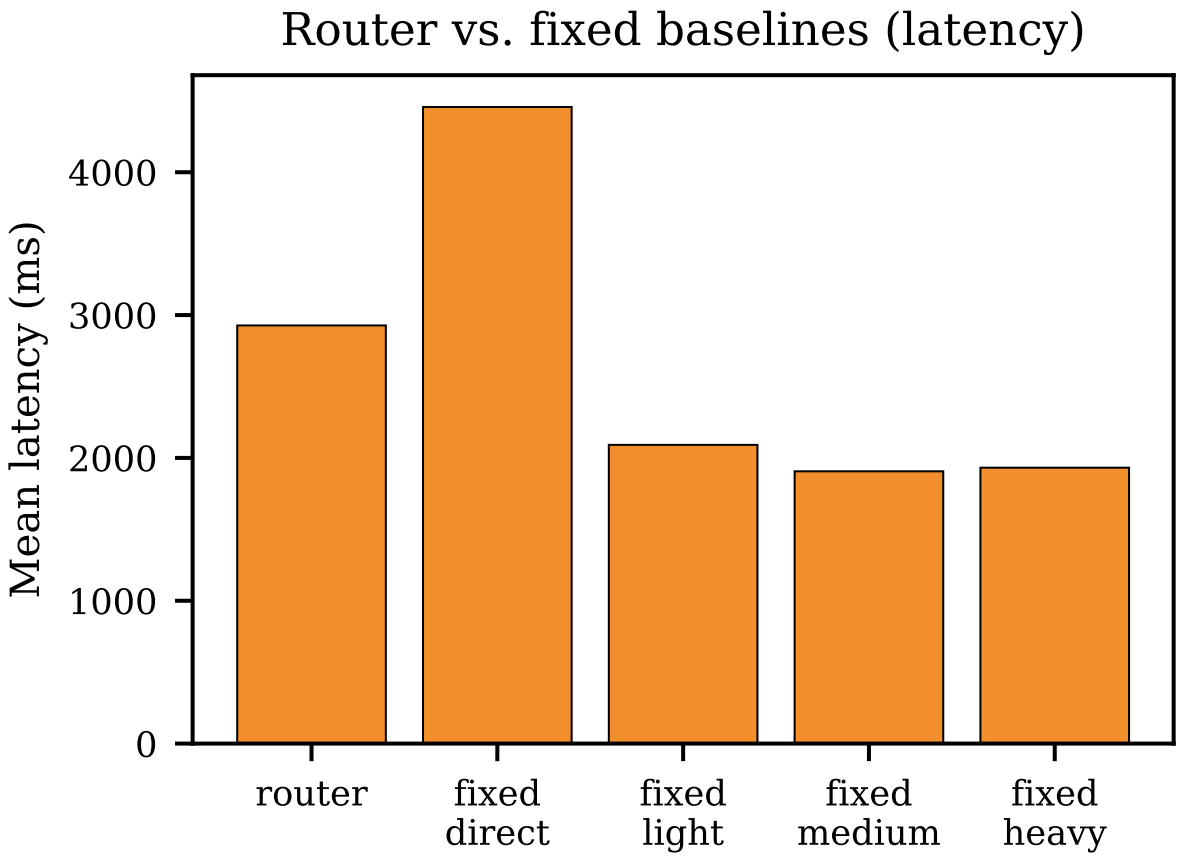

Figure 13. Mean end-to-end latency (ms) by policy. The router reduces latency by 34% vs. fixed-direct.

 doi:10.20944/preprints202604.1182.v1

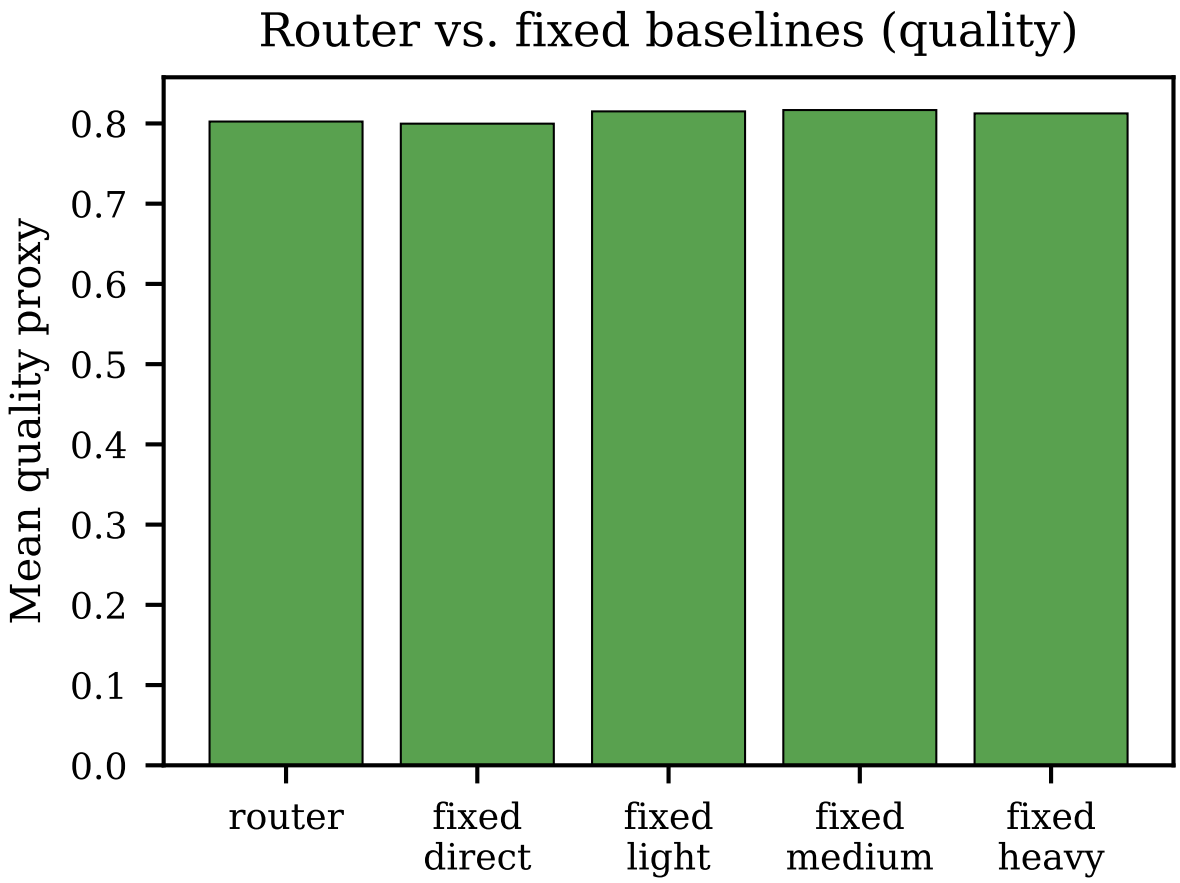

Figure 14. Mean lexical quality proxy by policy. All policies fall within 0.80–0.82; the bars appear visually flat because differences are within the proxy's noise floor. The intended reading is parity, not identity—see Table 3 for exact values and the surrounding text for discussion.

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 2,165 ms—a 26% improvement over the default 2,927 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.

 doi:10.20944/preprints202604.1182.v1

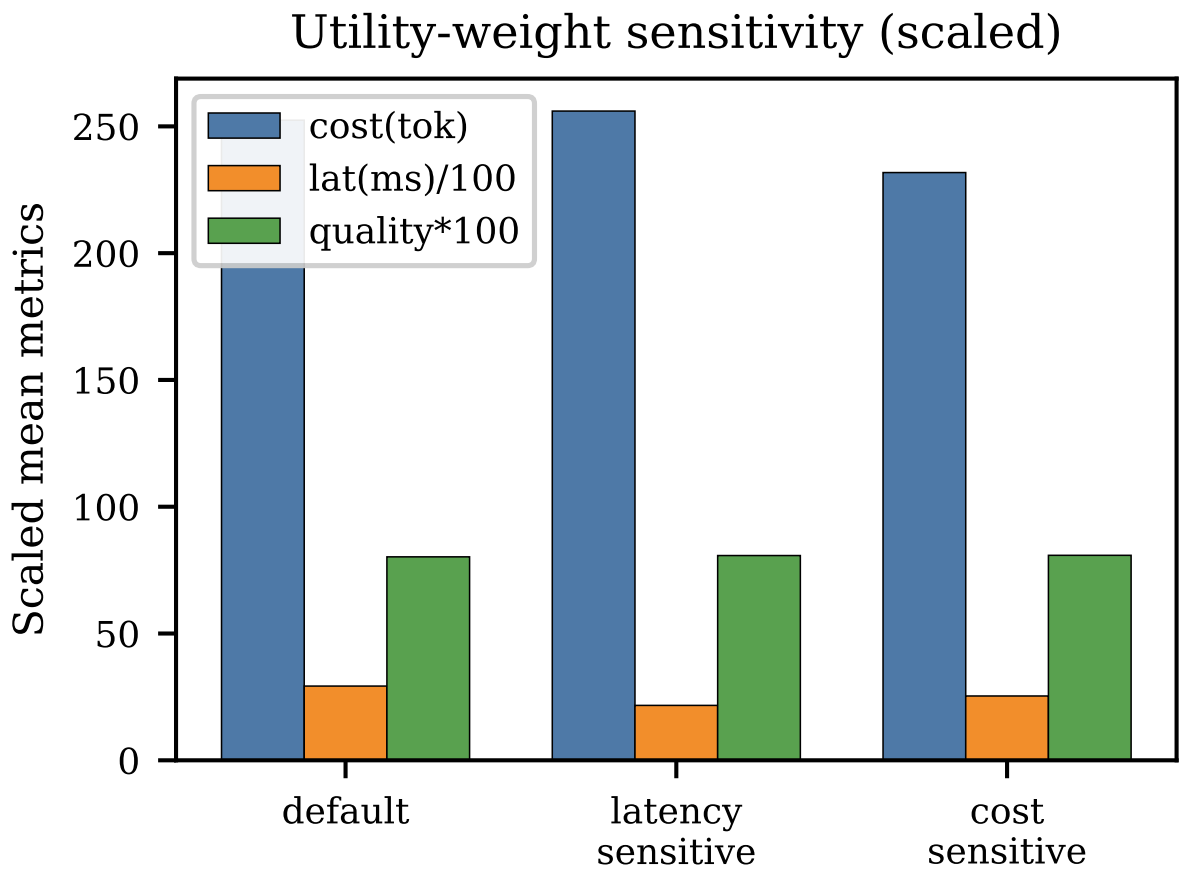

Figure 15. Normalized mean cost, latency, and quality under default, latency-sensitive, and cost-sensitive weight settings.

7.7. Per-Query Delta Analysis

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

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.

 doi:10.20944/preprints202604.1182.v1

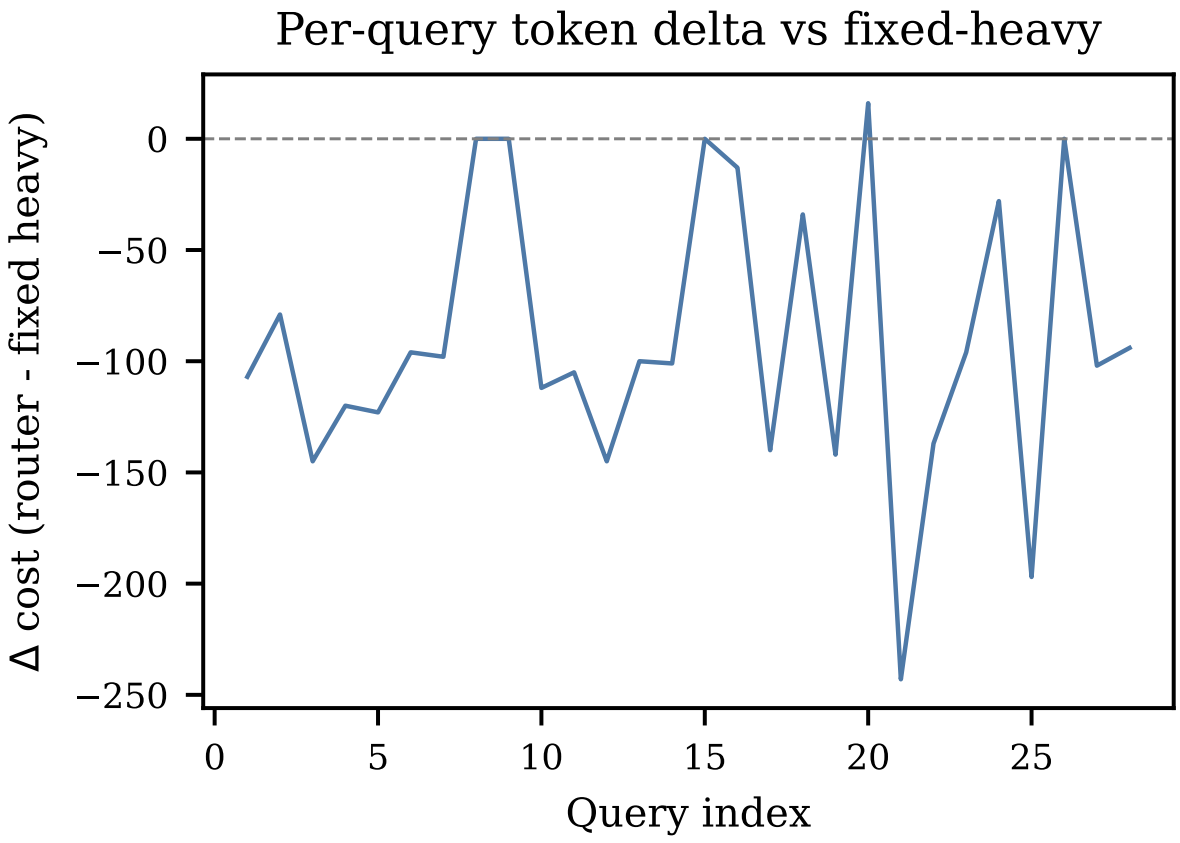

Figure 16. Per-query token cost delta vs. fixed-heavy. Savings are concentrated in definitional queries.

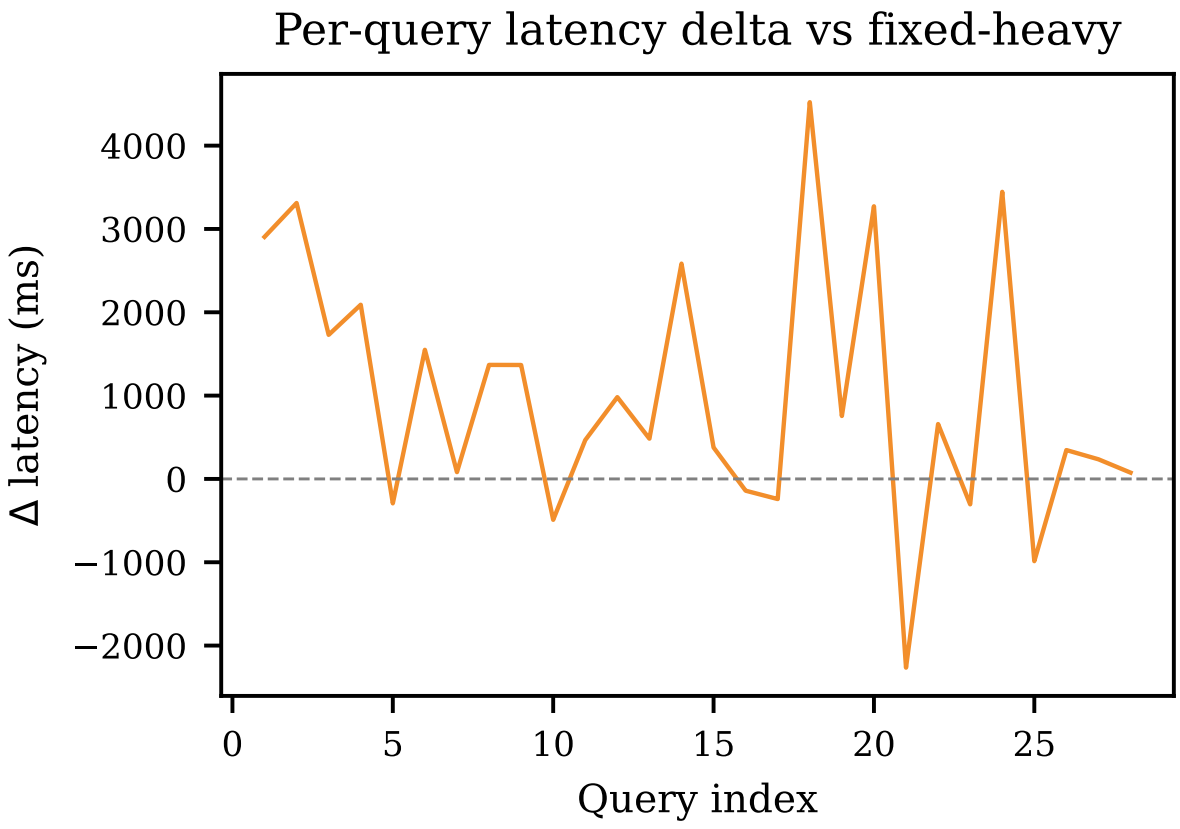

Figure 17. Per-query latency delta vs. fixed-heavy.

 doi:10.20944/preprints202604.1182.v1

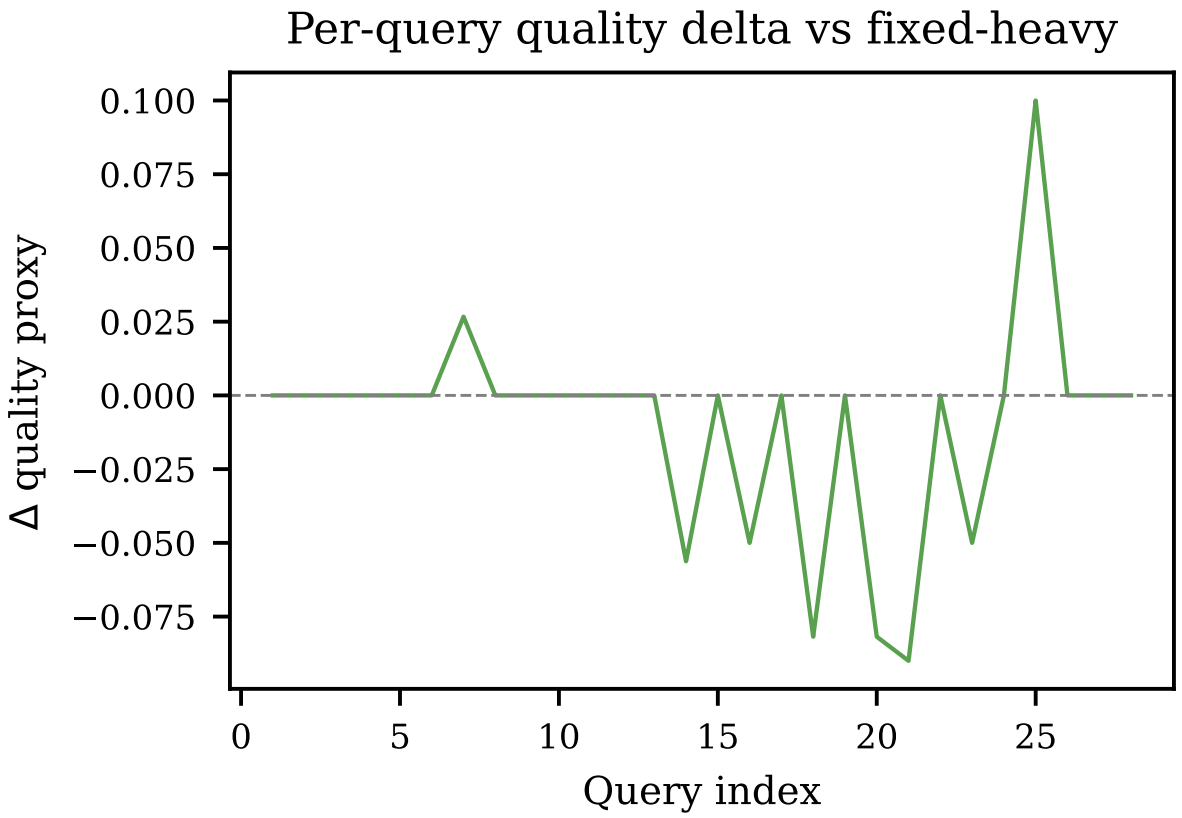

Figure 18. Per-query quality proxy delta vs. fixed-heavy. Quality parity is maintained across all query types.

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_llm` 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.

 doi:10.20944/preprints202604.1182.v1

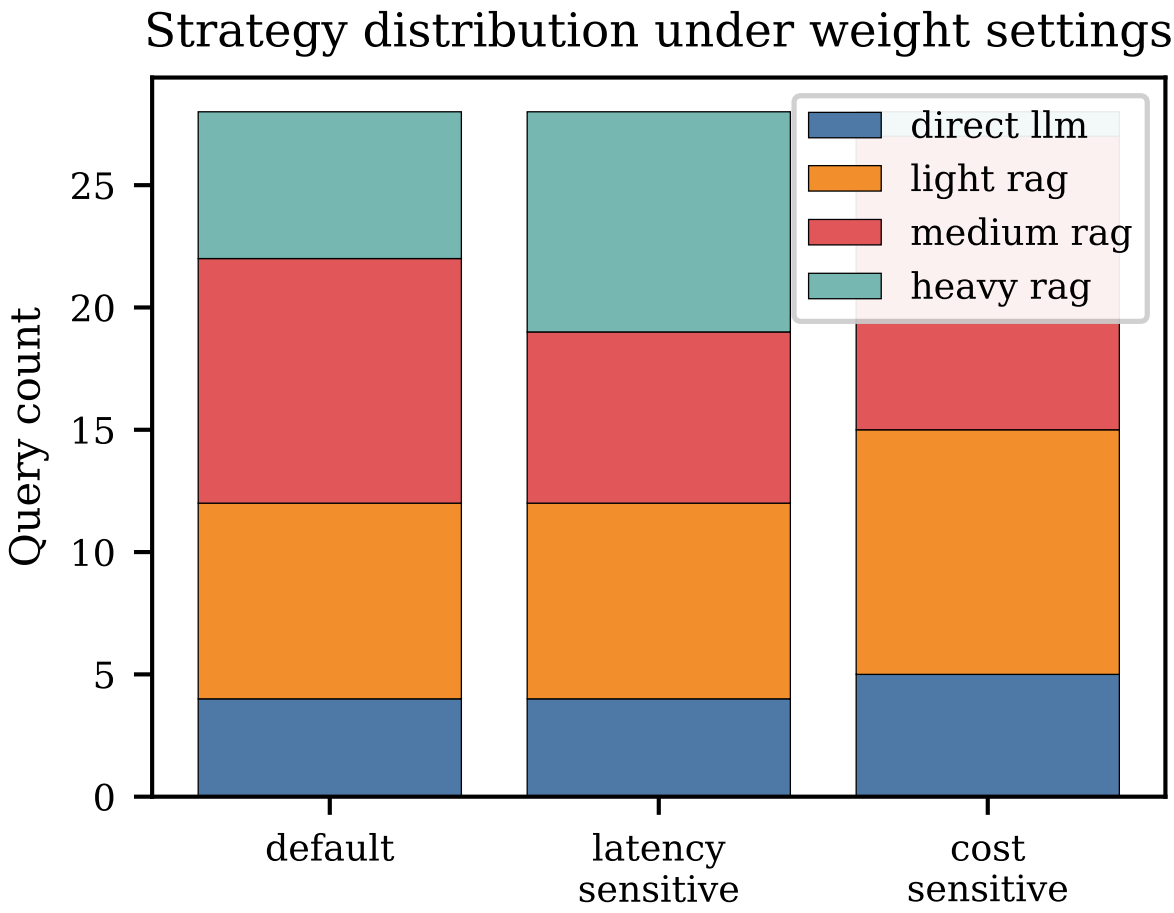

Figure 19. Strategy selection distribution under default, latency-sensitive, and cost-sensitive weight settings.

Table 5. Run-wide descriptive statistics over 28 queries.

Variable	mean	std	min	max
cost	258.5	94.5	161.0	530.0
latency	2512.3	1675.6	1151.6	8253.4
utility	0.3	0.1	0.1	0.4
quality_proxy	0.8	0.1	0.6	1.0

Table 6. Per-strategy means ± std. dev. Cost and latency scale monotonically with retrieval depth; quality gains diminish beyond `medium_rag`.

Strategy	mean cost	mean latency	mean U
direct_llm	236.0 ± 56.0	4274 ± 900	0.333 ± 0.123
light_rag	203.7 ± 84.9	2490 ± 1703	0.263 ± 0.092
medium_rag	216.7 ± 45.9	1714 ± 588	0.250 ± 0.086
heavy_rag	367.0 ± 74.0	2869 ± 2298	0.220 ± 0.057

 doi:10.20944/preprints202604.1182.v1

Table 7. Pearson correlations among per-query logged scalars. Weak complexity–cost correlation (r=0.21) identifies a key limitation of heuristic routing signals.

	cost	**lat.**	U	**cplx.**
cost	1.00	0.66	-0.50	0.22
lat.	0.66	1.00	-0.19	0.13
U	-0.50	-0.19	1.00	-0.48
cplx.	0.22	0.13	-0.48	1.00

8. Discussion

8.1. 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.

8.2. 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.

8.3. Signal Quality and Routing Accuracy

Heuristic complexity correlates only weakly with billed tokens (r=0.21, 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.

8.4. Weight Calibration as a Product Decision

Utility weights encode the operator's current product priorities and should not be treated as fixed hyperparameters. In a latency-sensitive customer-facing chatbot, w_L should be high; in an overnight batch analytical pipeline, w_C can be maximized. The sensitivity analysis confirms that the same bundle catalog accommodates both extremes. Weights should be calibrated using business KPIs (e.g., abandonment rate as a proxy for latency) and recalibrated quarterly or when model pricing changes.

8.5. 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. Mitigation: corpus auditing and gap-filling with canonical source documents. Second, *routing overconfidence*: high-confidence retrieval occasionally returns topically adjacent but semantically insufficient passages. Mitigation: reranking bundles or cross-encoder verification as an optional retrieval stage.

8.6. Scalability Pathway

The benchmark corpus is intentionally compact; scaling to thousands of documents introduces additional challenges: FAISS index build time, memory footprint, chunking policy effects, and calibration drift between retrieval confidence and downstream quality. CA-RAG's bundle abstraction

 doi:10.20944/preprints202604.1182.v1

accommodates this scaling—specialized 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.

9. Limitations

Quality estimation. Quality priors are hand-specified and the quality proxy is lexical (token overlap); neither captures semantic accuracy or user satisfaction. Replacement with a calibrated LLM judge or human ratings is necessary for production readiness.

Benchmark scale. The 28-query, 15-passage benchmark is designed for controlled analysis, not for generalization to real-world query distributions. Results should be validated on domain-specific corpora such as Natural Questions [5] or HotpotQA, or on enterprise knowledge bases.

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.

Two findings stand out. The heuristic complexity signal is a weak predictor of actual retrieval need (r=0.21), pointing toward richer signals entity density, intent classification, embedding-space novelty as the most impactful near-term improvement. 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.

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 $c \in [0, 1]$.

 doi:10.20944/preprints202604.1182.v1

2. For each bundle $b \in \mathcal{B}$, compute estimated utility U_b via Equation (3).
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. 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.
```

 doi:10.20944/preprints202604.1182.v1

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, Eq. 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

```
 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
```

 doi:10.20944/preprints202604.1182.v1


```
 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 vs. heavy retrieval                => medium_rag
12. Explain how telemetry refines routing estimates => light_rag
13. Why might a system skip retrieval?               => heavy_rag
14. List tradeoffs between large/small top-k         => medium_rag
15. How do embedding tokens differ from completion? => medium_rag
16. Describe a municipal RAG use case                => medium_rag
17. What are risks of fixed retrieval depth?         => medium_rag
18. How does CA-RAG combine quality/latency/cost?    => heavy_rag
19. Explain when reranking is worth extra latency    => medium_rag
20. Derive intuitive explanation of discrete bundles=> direct_llm
21. What operational metrics should teams report?    => heavy_rag
22. How does query length influence complexity?      => medium_rag
23. Contrast direct vs. retrieval-grounded answers   => medium_rag
24. Limitations of lexical quality proxies?          => medium_rag
25. How to tune weights for latency-sensitive chat? => medium_rag
26. Describe experiment protocol for token logging   => medium_rag
27. Role of exploration epsilon in bundle selection => light_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,
-0.3573
What is hybrid retrieval?,medium_rag,179,1111.8,0.3672,0.85,-0.0159
What is FAISS used for?,heavy_rag,303,1433.3,0.2813,0.91,0.1274
```

References

1. 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), 2020, Vol. 33, pp. 9459–9474.
2. 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 preprint arXiv:2312.10997* **2023**.
3. Izacard, G.; Grave, E. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. *arXiv preprint arXiv:2007.01282* **2021**.
4. OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* **2024**.
5. 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 Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 6769–6781.
6. Khattab, O.; Zaharia, M. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In Proceedings of the Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, pp. 39–48.
7. Johnson, J.; Douze, M.; Jégou, H. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* **2021**, *7*, 535–547.
8. Guu, K.; Lee, K.; Tung, Z.; Pasupat, P.; Chang, M. REALM: Retrieval-Augmented Language Model Pre-Training. In Proceedings of the Proceedings of the 37th International Conference on Machine Learning (ICML), 2020, pp. 3929–3938.

 doi:10.20944/preprints202604.1182.v1

9. Robertson, S.; Zaragoza, H. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval* **2009**, *3*, 333–389.
10. Li, L.; Chu, W.; Langford, J.; Schapire, R.E. A Contextual-Bandit Approach to Personalized News Article Recommendation. In Proceedings of the Proceedings of the 19th International Conference on World Wide Web (WWW), 2010, pp. 661–670.
11. Lattimore, T.; Szepesvári, C. *Bandit Algorithms*; Cambridge University Press, 2020.

Short Biography of Authors

Sanjay Mishra is a seasoned technology leader, author, and innovator with over 14 years of expertise spanning data engineering, AI, and agentic systems. As a Principal Software Engineer at Fidelity Investments, he architects and delivers enterprise-scale data solutions and AI-driven platforms that power real-world decision-making. A prolific author, he has written two foundational technical books—*The SQL Universe* and *Oracle Database Performance Tuning: A Checklist Approach*—both widely regarded as essential references for engineers building high-performance data platforms. His research on AI and agentic systems has been published in IEEE, reflecting his commitment to advancing the frontier of intelligent systems. He is an active builder in the MCP ecosystem, having designed and deployed MCP servers and agents that connect natural language interfaces to complex data infrastructure, including NL2SQL pipelines powered by Oracle and OpenAI. Beyond engineering, he is a sought-after speaker at industry events and a judge at hackathons, where he mentors the next generation of technologists. Mr. Mishra is a Member of the IEEE.

Ganesh R. Naik is a globally recognized leader in biomedical engineering, ranked among the top 1% of researchers worldwide in his field, and a prominent expert in data science and biomedical signal processing. He earned his Ph.D. in Electronics Engineering, specializing in biomedical engineering and signal processing, from RMIT University, Melbourne, Australia, in 2009, and holds an Associate Professor position in IT and Computer Science at Torrens University, Adelaide, Australia. Previously, he was an academic and research theme co-lead at Flinders University's sleep institute, a Postdoctoral Research Fellow at the MARCS Institute, Western Sydney University (2017–2020), and a Chancellor's Post-Doctoral Research Fellow at the Centre for Health Technologies, University of Technology Sydney (2013–2017). Dr. Naik has edited 15 books and authored approximately 200 papers in peer-reviewed journals and conferences. He serves as Associate Editor for *IEEE Access*, *Frontiers in Neurorobotics*, and two Springer journals. His contributions have been recognized with the Baden–Württemberg Scholarship (2006–2007), an ISSI overseas fellowship (2010), the BridgeTech industry fellowship from the Medical Research Future Fund, Government of Australia (2023), and the UK Royal Society Fellowship.