Latency-Quality Routing for Functionally Equivalent Tools in LLM Agents

Anonymous ACL submission

Abstract

Tool-augmented LLM agents increasingly access the same tool type through multiple functionally equivalent providers, such as web-search APIs, retrievers, or LLM backends behind a shared interface. This creates a same-function provider-routing problem under runtime load: the router must choose among providers that differ in latency, reliability, and answer quality, often without deployment-time gold labels. Existing priority/fallback policies are largely quality-blind, while additive latency-quality rewards can let low latency compensate for poor answers. We introduce **LQM-ContextRoute**, a contextual bandit router that performs latency-quality matching by ranking providers according to expected answer quality per service cycle, using query-specific quality estimates and LLM-as-judge feedback. Across search, QA, retriever, and LLM-backend provider pools, LQM-ContextRoute reaches 0.583 F1 at 431 ms in the search API pool while remaining on the latency-quality frontier. Under stronger provider-quality heterogeneity, it avoids additive-reward collapse, improving StrategyQA accuracy by up to $+18$ pp and heterogeneous-retriever NDCG by $+2.91$–$+3.22$ pp over a load-aware online baseline. These results show that same-function tool routing benefits from treating latency as service capacity rather than as an additive reward offset.

1 Introduction

Tool-augmented LLM agents increasingly call external services through shared tool interfaces. A single `web_search` call, for example, may be served by Tavily, Brave, or DuckDuckGo behind an MCP (Anthropic, 2024) or function-calling interface. These providers are *functionally equivalent* in the API sense: each accepts the same kind of query and returns the same kind of response. They are not operationally equivalent. Latency, rate limits, failure modes, and answer contribution vary across providers and across load regimes. Recent stress reports on MCP servers and tool-using agents (Digital Applied, 2026; Gupta, 2026) document failures from timeouts, schema mismatch, upstream errors, and rate limiting; our live profile of 270 calls across three search providers also shows distinct latency shapes even when all calls succeed (App. D). Provider choice is therefore part of the agent's runtime behavior, not just a deployment detail.

Prior work on tool-augmented agents primarily studies *which tool type* an agent should invoke, such as whether to search, retrieve, call an API, or reason directly (Schick et al., 2023; Patil et al., 2024; Qin et al., 2024; Yao et al., 2023). We study the gateway decision that follows tool selection: once the agent has selected a tool interface, which provider of that interface should receive the call under current load, when online gold labels are unavailable? This provider-level view matches how gateways are configured in practice: routing is usually scoped within a selected interface, so the candidate set is the provider pool for that interface rather than the agent's full tool inventory. Production routers such as Portkey (Portkey, 2025b) and LiteLLM (BerriAI, 2026) address availability with priority lists, cooldowns, and fallbacks. These mechanisms are useful, but they are largely reactive and quality-blind; App. E includes a priority-router replay showing that cooldown policies can be sensitive to the operator-specified priority order.

A natural learning alternative is to make a bandit load-aware with an additive latency-quality reward, $r = \alpha u - (1-\alpha)\tilde{\tau}$. Such rewards are simple and reasonable when providers have similar answer quality, but they can fail when quality is heterogeneous. Low latency can compensate for a provider that contributes little to the answer, allowing the router to satisfy an SLA while silently degrading task performance. The central question is therefore not only whether a router adapts to load, but how

latency should enter the learning objective.

We propose **LQM-ContextRoute**, a contextual bandit router for same-function providers. Its main design choice is *latency-quality matching*: instead of treating latency as an additive reward penalty, the router scores each provider by a renewal-reward rate, $\hat{u}_i/(1+\hat{\tau}_i/L_{\text{ref}})$. This treats latency as service-cycle cost, so a fast low-quality provider is not rewarded for being fast alone. The router combines this objective with a LinUCB quality head and LLM-as-judge feedback, allowing it to learn query-specific quality estimates without online gold labels.

Contributions.

- We introduce same-function provider routing as a gateway mechanism for LLM agents: after a tool interface has been selected, the router chooses among interchangeable providers under runtime load, partial feedback, and no deployment-time gold labels.
- We design LQM-ContextRoute, a contextual bandit router whose scoring rule matches expected quality to service-cycle cost instead of mixing quality and latency additively. This lets the router adapt to changing provider load without allowing fast low-quality providers to win by speed alone.
- We evaluate LQM-ContextRoute across search, QA, retriever, and LLM-provider pools against production policies, online learning routers, additive latency-quality variants, and oracle frontiers. It remains on the search-pool latency-quality frontier, avoids additive-reward collapse with up to $+18$ pp StrategyQA accuracy, and improves heterogeneous-retriever NDCG by $+2.91$–$+3.22$ pp over a load-aware online baseline.

2 Background and Problem Formulation

Provider-routing setting. A selected function category C is served by a pool $\mathcal{T}_C=\{T_1,\ldots,T_K\}$ of K functionally equivalent providers. Each provider implements the same external interface, but may differ in answer quality, latency, rate limits, failure behavior, or backend modality. At round t, a query q_t arrives after the upstream agent has already selected category C. The router chooses provider $i_t \in [K]$, observes latency $\tau_{i_t}(t) \geq 0$ immediately, and later receives a quality scalar $u_{i_t}(t)\in[0,1]$ from an online evaluator. The router observes feedback only for the provider it calls.

Feedback and deployment constraints. This setting differs from both upstream tool selection and a vanilla contextual bandit. The action space is not the agent's full tool inventory, but the provider pool behind one selected interface. Load is non-stationary, quality labels are not available online, and provider preferences can depend on the query. External stress reports and our live search profile show that runtime state varies across providers even for the same interface (App. D), motivating a router that tracks service state while still accounting for answer quality.

Objective and additive mismatch. The operator specifies a reference service-time budget L_{ref}; in our experiments it matches the SLA threshold. Ideally, a routing policy should maximize expected answer quality while keeping service time within this budget:

$$\pi^\star = \arg\max_{\pi} \mathbb{E}_{i\sim\pi}[u_i] \;\text{ s.t. }\; \mathbb{E}_{i\sim\pi}[\tau_i] \leq L_{\text{ref}}, \quad (1)$$

A common load-aware alternative is to scalarize quality and latency as $r=\alpha u-(1-\alpha)\tilde{\tau}$, where $\tilde{\tau}$ is normalized to $[0,1]$ (Poon et al., 2025; Chadderwala, 2025). This additive form makes the two axes compensate: at $\alpha{=}0.4$, $\tilde{\tau}{=}0, u{=}0.1$ scores 0.04, while $\tilde{\tau}{=}1, u{=}0.65$ scores -0.34, so the lower-quality low-latency provider is ranked higher. The mismatch is not that latency is irrelevant; it is that latency should consume service capacity rather than directly offset answer quality.

3 Empirical Motivation

To motivate provider-level routing, we profile three web-search providers—Tavily, Brave, and DuckDuckGo—on 200 questions from HotpotQA and TriviaQA. All three are called through the same `web_search` interface, so observed differences reflect the provider, not the call site.

Providers differ on every axis. Figure 1 shows that the three providers occupy distinct points in cost, latency, F_1, and success rate. The usual cost–quality correlation holds only in aggregate; neither cost nor latency alone identifies the best provider for a given query.

Quality is query-specific. Embedding queries and projecting them with UMAP (Figures 2, 3) reveals two patterns: many queries are ties across all

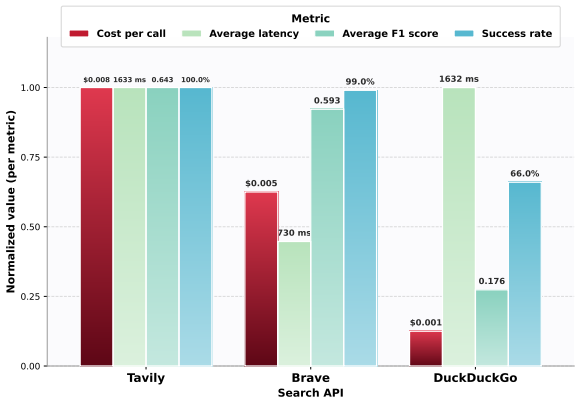
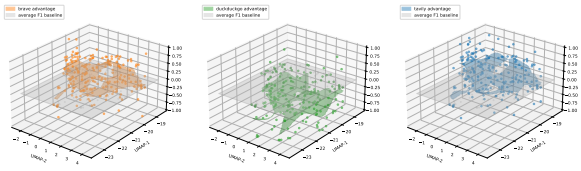


Figure 1: Per-provider cost, latency, F_1, and success rate. Bars are min–max normalised; in-bar labels show raw values.



(a) Brave (b) DuckDuckGo (c) Tavily

Figure 2: Per-provider F_1 advantage over the other two, plotted on the UMAP embedding of the query.

three providers, and the non-tied winners are scattered rather than clustered. Tavily wins most often overall, but Brave and DuckDuckGo each dominate on substantial subsets that are not separable by coarse query type.

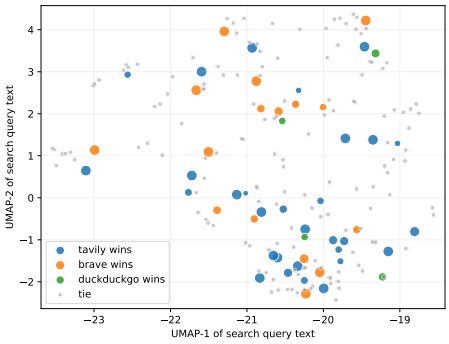


Figure 3: UMAP of query embeddings, coloured by the highest-F_1 provider. Grey points are ties.

Routing dominates any fixed-provider policy. Figure 4 compares single-provider policies against a per-query oracle that picks the highest-F_1 provider within a 1065 ms SLO, breaking ties by cost. Any fixed choice trades cost for quality; the oracle cuts cost by 37.5% *and* improves F_1 by 11.8% over the best single-provider baseline by

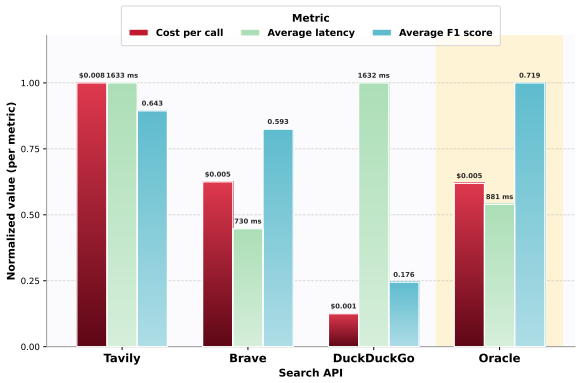


Figure 4: Single-provider policies vs. the per-query oracle (SLO 1065 ms, ties broken by cost).

spending the expensive provider only where it matters.

These observations motivate a per-query router. The remaining question is how to approximate the oracle online, without gold labels and under non-stationary load—the problem addressed in the rest of the paper.

4 Method: LQM-ContextRoute

4.1 Router Overview

LQM-ContextRoute is a router for a fixed pool of functionally equivalent providers. At round t, it receives a query representation $\mathbf{x}_t$, scores each active provider, calls one provider i_t, observes latency $\tau_{i_t}(t)$, receives an online quality score $u_{i_t}(t)$, and updates only the selected provider. Figure 5 summarizes this online loop. Each provider has a query-conditional quality head, a provider-level latency tracker, and an uncertainty estimate for exploration. The selection rule combines them into one score: predicted quality is divided by current service cost, and an optimism bonus encourages exploration when quality is uncertain.

The state separation mirrors the deployment setting. Load is primarily a property of an upstream service at a given time, so latency is tracked at the provider level. Quality is query-conditional because two providers that share an API can still differ by question type, retrieval coverage, or reasoning fit. The renewal-rate term is the objective; the LinUCB head estimates query-specific quality; and the EMA tracks current service cost.

4.2 Latency-Quality Matching

The core design choice is the selection objective. Additive rewards $\alpha u - (1-\alpha)\tilde{\tau}$ allow a low-quality

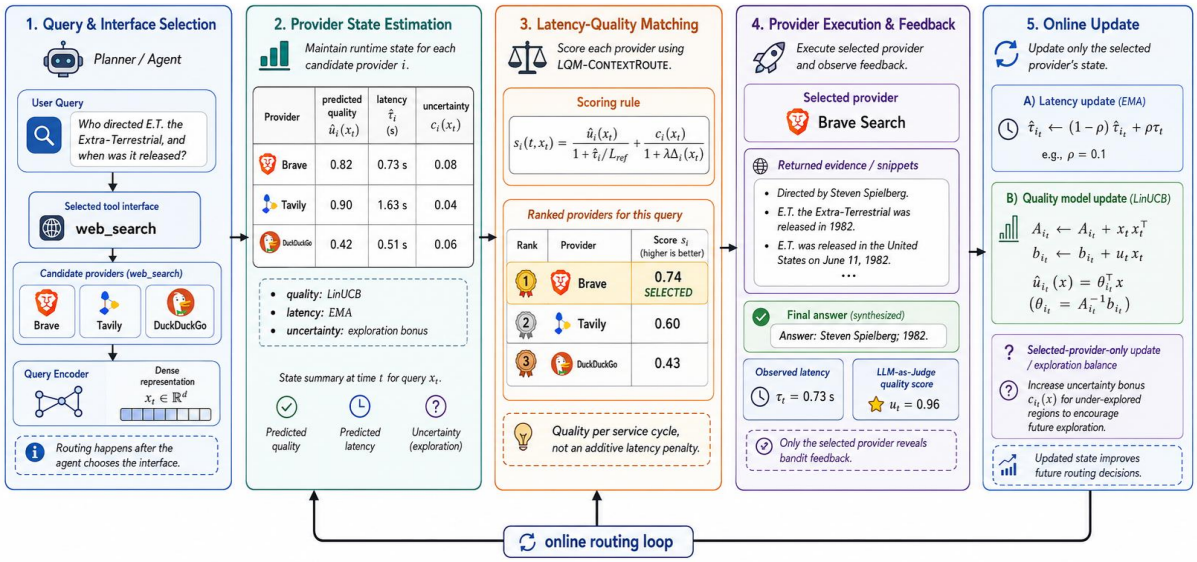


Figure 5: Architecture of LQM-ContextRoute. A query embedding is scored against one quality head per provider; latency is tracked as provider-level service cost; the selected provider receives bandit feedback from latency and an online quality evaluator. The routing score divides predicted quality by service-cycle cost and uses gap-deflated uncertainty for exploration.

provider to be rescued by low latency. We instead score a provider by the renewal-reward rate

$$V_i = \frac{u_i}{1 + \tau_i / L_{\text{ref}}}, \tag{2}$$

where L_{ref} is the operator-set latency budget. This score gives a capacity-aware surrogate for the constrained goal in Eq. 1: quality is the reward, while latency consumes service capacity. Under the linear-cycle calibration, a call takes one accounting unit plus a service-time penalty τ_i/L_{ref}. The renewal-reward theorem gives the long-run reward rate $u_i/(1+\tau_i/L_{\text{ref}})$ (Ross, 1996, Thm. 3.6.1). Appendix C gives the formal characterisation and discusses alternative calibrations. The important property for routing is non-compensation: if u_i is near zero, the score is near zero regardless of how fast the provider is.

This objective also gives L_{ref} a direct operational meaning: it is the latency scale at which one call consumes roughly one extra unit of service time. A smaller value makes the router more latency-sensitive, while a larger value approaches quality-first routing. In the experiments we set L_{ref} to the SLA threshold and check sensitivity in Appendix D.

4.3 Contextual Quality and Latency Estimates

LQM-ContextRoute estimates quality with one LinUCB head per provider and estimates latency with an exponential moving average. For provider i, let $\hat{u}_i(\mathbf{x}_t) = \mathbf{x}_t^\top A_i^{-1}\mathbf{b}_i$ be the contextual quality estimate and $\hat{\tau}_i$ the current latency estimate. After selecting provider i_t, the router updates $\hat{\tau}_{i_t} \leftarrow (1-\rho)\hat{\tau}_{i_t} + \rho\tau_t$ and the selected quality head with $A_{i_t} \leftarrow A_{i_t} + \mathbf{x}_t\mathbf{x}_t^\top$ and $\mathbf{b}_{i_t} \leftarrow \mathbf{b}_{i_t} + u_t\mathbf{x}_t$. Sliding-window removals use Sherman–Morrison updates to maintain A_i^{-1} efficiently. Unselected providers are not updated, matching the bandit feedback available in the gateway.

4.4 Selection Rule

The router selects the provider with the largest score

$$s_i(t, \mathbf{x}_t) = \frac{\hat{u}_i(\mathbf{x}_t)}{1 + \hat{\tau}_i/L_{\text{ref}}} + \frac{\alpha_{\text{ucb}}\sqrt{\mathbf{x}_t^\top A_i^{-1}\mathbf{x}_t}}{1 + \lambda\,\Delta_i(\mathbf{x}_t)}, \tag{3}$$

where $\Delta_i(\mathbf{x}_t) = \max(0, \max_j \hat{u}_j(\mathbf{x}_t) - \hat{u}_i(\mathbf{x}_t))$. The first term is capacity-aware exploitation: the router matches predicted answer quality to the current service cost of the provider. The second term is uncertainty-driven exploration: it is the LinUCB confidence bonus, deflated for arms whose estimated quality is already dominated. We use $\lambda = 1$ in all experiments; Appendix A gives the complete online loop.

The deflation term is deliberately asymmetric. It does not punish a slow provider merely for being slow, because a slow provider may be the only one with useful evidence for a query. It only reduces exploration when the current contextual model al-

ready estimates that another provider has higher quality. This is the mechanism that distinguishes LQM-CONTEXTROUTE from a pure load balancer: the router may spend latency when the expected quality return is large, while reducing exploration of arms whose predicted quality is already dominated for the current query distribution.

4.5 Online Update and Feedback

After a provider returns, the latency observation updates $\hat{\tau}_{i_t}$ and an online evaluator supplies $u_{i_t}(t) \in [0, 1]$ from the chosen provider's response. In our experiments this evaluator is a local LLAMA-3.2-1B judge (mean 64 ms, P95 115 ms), though the router only requires a scalar reward proxy available to the gateway. We validate this proxy against a 30B reference in App. D.

The proposed router throughout the paper is LQM-CONTEXTROUTE. We use one ablation, LQM-ONLY, only to isolate the scoring objective from the contextual quality model. It removes $\mathbf{x}_t$ and replaces the LinUCB quality head with EMA quality:

$$s_i(t) = \frac{\hat{u}_i}{1 + \hat{\tau}_i / L_{\text{ref}}} + \frac{\beta \sqrt{\log(t)/(n_{w,i}+1)}}{1 + \lambda \Delta_i}. \tag{4}$$

When the pool is quality-homogeneous, Eq. 4 behaves like load-greedy SW-UCB; when one provider dominates on both axes, the quality term drives selection. These limits are tested in §5.4 and §5.3.

4.6 Score-Level Properties

Theorem 2 (App. C) states the main score-level distinction: for every $\alpha \in (0, 1)$, there are 2-provider instances where an additive composite ranks the lower-quality fast provider above the higher-quality provider, while Eq. 2 ranks them correctly. This theorem explains the failure mode targeted by the experiments; bandit-level performance still depends on exploration.

For the unmodulated optimistic bonus ($\lambda = 0$), the plug-in throughput estimator concentrates at the EMA rate with Lipschitz constants $c_u \leq 1$ and $c_\tau \leq L_{\text{ref}}^{-1}$, yielding

$$R_T \leq \sum_{i:\Delta_i^V > 0} \frac{C(1 + L_{\text{ref}}^{-1})^2 \log T}{\Delta_i^V} + o(\log T).$$

Appendix C gives the proof sketch and states the additional assumptions; the $\lambda > 0$ deflation used in Eq. 3 is treated as an empirical exploration control rather than as a separate optimism guarantee.

5 Evaluation

The method in §4 makes two linked claims: same-function provider routing should improve service behavior under runtime load, and latency should affect selection through a capacity-aware quality rate rather than an additive penalty. We evaluate these claims across four provider-pool families—web-search APIs, QA providers, retrievers, and LLM backends—that vary in load non-stationarity and provider-quality heterogeneity. We test three questions:

- **Q1: Load-aware routing.** Under non-stationary provider load, routing should reduce latency and SLA misses without sacrificing answer quality.
- **Q2: Latency-quality matching.** When fast providers are not also high-quality providers, LQM-CONTEXTROUTE should avoid the additive-compensation failure predicted by Theorem 2.
- **Q3: Transfer and limits.** The latency-quality objective should transfer across provider types when quality differs, and it should remain stable when one provider is already dominant.

The search API pool is the most deployment-shaped instance because its providers are live services behind the same logical interface. The QA, retriever, and LLM-backend pools stress stronger quality heterogeneity, transfer, and stable-dominant boundary cases. We report both task quality and latency: a latency-only router can satisfy the SLA while returning poor answers, whereas a quality-only router can exceed the service budget.

5.1 Setup

Benchmarks and provider pools. The evaluation covers six same-function pools (Table 1). The search API pool uses 200 questions (100 HotpotQA + 100 TriviaQA) and three providers behind one shared `web_search` interface; for each (query, provider) pair, a Qwen3-30B reader answers from the returned evidence and we score F1. The remaining pools instantiate the same routing problem with high-heterogeneity StrategyQA, a moderate StrategyQA companion, an LLM-provider ladder, and two retriever pools (SciFact and NFCorpus). Within each pool, all routers operate over the same

response table, separating routing behavior from reader noise while varying the degree of latency-quality misalignment.

Runtime load. Latency is drawn online from calibrated service profiles anchored by vendor medians and our live Tavily/Brave/DDG measurements. We evaluate four non-stationary patterns reused across provider pools: STEP overload and recovery, ROTATION of the overloaded provider, random SPIKE bursts, and smooth GRADUAL drift. SLA is the fraction of calls below 1.5 seconds, matching L_{ref}.

Baselines. Since prior LLM-routing systems usually route among model endpoints with fixed cost tables or preference labels, we organize comparisons by transferable policy family rather than treating a single system as a direct comparator for this setting. We compare production-style policies (STATIC, ROUND-ROBIN, REACTIVE-COOLDOWN), online learning policies (EMA-GREEDY, SW-UCB, CONTEXTROUTE), additive latency-quality variants (Table 5, App. D), and LATENCY-ORACLE/F1-ORACLE frontiers. Static-T0/T1 denotes the fixed-priority provider for the corresponding pool. LQM-ONLY is included only as a scoring ablation ($L_{\text{ref}} = 1500$ ms, $\lambda = 1$). Unless otherwise noted, each router runs for $T = 200$ rounds over 50 seeds.

5.2 Q1: Routing under non-stationary provider load

The search API pool has three providers behind one logical interface, measured provider-response scores, and non-stationary load. Table 2 breaks the result down by load pattern. The Tavily/Brave quality gap is only about 1 pp, so this pool primarily tests latency and SLA behavior rather than the largest expected quality gain. All learning routers address the load problem: against STATIC-T1, they cut mean latency by 50–67% on STEP, lift SLA from 89% to $\geq$ 98%, and approach the latency oracle in service behavior while improving task quality. Within the learning routers, LQM-CONTEXTROUTE adds +0.85 pp F1 over CONTEXTROUTE averaged over patterns and +2.18 pp over SW-UCB, although individual load patterns still expose the expected latency-quality trade-off.

Figure 6 makes the latency-quality trade-off explicit. LQM-CONTEXTROUTE sits on the empirical Pareto frontier of the learning routers: relative to SW-UCB, it spends latency for +2.18 pp F1; relative to CONTEXTROUTE, it gains +0.85 pp F1 with slightly lower mean latency. A sweep over $L_{\text{ref}} \in \{750, 1500, 3000\}$ changes LQM-CONTEXTROUTE by at most 0.29 pp F1 and keeps SLA in 97.9–98.0% (App. D).

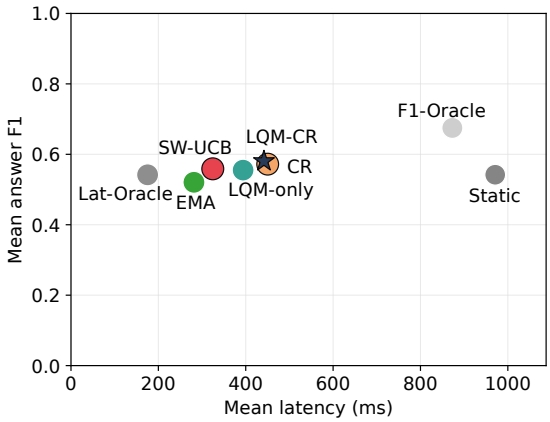


Figure 6: Latency-quality Pareto view of the search API pool. Marker size encodes SLA@1.5s.

The SPIKE pattern follows the same trend: LQM-CONTEXTROUTE obtains F1 0.591 at 387 ms mean latency, compared with CONTEXTROUTE at 0.583 and 415 ms, SW-UCB at 0.579 and 263 ms, and STATIC-T1 at 0.572 and 580 ms. These results support the load-aware routing claim: learning routers reduce latency under non-stationary service states, and LQM-CONTEXTROUTE remains on the quality/latency frontier among learnable policies. The comparison with static, cooldown, and load-aware bandit baselines also shows why latency alone is insufficient.

5.3 Q2: When does latency-quality matching matter?

The latency-quality matching score should matter most when the same interface hides large provider-quality differences. In the search API pool, a per-query provider-gap slice tests this mechanism: among 106 high-gap questions, LQM-CONTEXTROUTE gains +4.42 pp over SW-UCB and cuts DDG traffic by 14.4 pp; on 85 zero-gap questions, F1 deltas shrink to within $\pm$0.5 pp.

The DDG-share reduction is diagnostic: the router moves traffic away from the fast weak provider precisely where provider choice affects answer quality.

The slice shows the mechanism in the search pool; StrategyQA (Geva et al., 2021) examines it under sharper quality heterogeneity. The pool contains QWEN-7B (accuracy = 0.643), LLAMA-1B (0.520), and DDG+JUDGE (0.123). All three

Pool	providers	role	metric	quality	latency (ms)	SLA	ΔSW	ΔCR
WebSearch	search APIs	live APIs	F1	0.583	431	97.9	+2.18	+0.85
StrategyQA	QA providers	high heterogeneity	Acc.	0.601	645	90.0	+17.54	+4.65
StrategyQA-Synth	QA providers	moderate heter.	F1	0.624	628	91.1	+3.47	+2.76
LLM-Ladder	LLM backends	backend transfer	F1	0.353	977	89.3	+8.89	+4.64
SciFact	retrievers	retrieval	NDCG	0.711	196	97.8	+2.91	-0.46
NFCorpus	retrievers	retrieval	NDCG	0.294	325	95.1	+3.22	-0.52

Table 1: Unified same-function provider-pool results for LQM-CONTEXTROUTE. Quality is F1, accuracy, or NDCG depending on the pool; deltas are quality points relative to SW-UCB and CONTEXTROUTE.

	STEP		ROTATION		GRADUAL	
Router	F1	lat. (ms)	F1	lat. (ms)	F1	lat. (ms)
F1-Oracle	0.697	670	0.667	952	0.636	1287
Latency-Or.	0.574	141	0.555	156	0.465	276
Static-T1	0.558	783	0.539	1011	0.499	1511
EMA-Greedy	0.550	228	0.491	360	0.483	301
SW-UCB	0.580	296	0.515	358	0.562	381
ContextRoute	0.602	386	0.538	501	0.566	499
LQM-ONLY	0.573	345	0.531	431	0.551	465
LQM-CONTEXTROUTE	0.594	409	**0.550**	490	**0.588**	481

Table 2: Search API provider pool by load pattern. SPIKE is discussed in the text.

Gap bin	qids	top-2 gap	ΔF1 vs SW	ΔDDG
zero	85	0.000	-0.49	-13.9
mild	9	0.061	+0.92	-19.7
large	106	0.317	+4.42	-14.4

Table 3: Search-pool heterogeneity slice for LQM-CONTEXTROUTE vs. SW-UCB. F1 and DDG-share deltas are percentage points.

	Accuracy by pattern			mean	
Router	step	rot.	spike	lat. (ms)	SLA
Static-T0	0.644	0.633	0.657	1238	70
EMA-Greedy	0.396	0.419	0.382	297	99
SW-UCB	0.412	0.413	0.464	286	100
LQM-ONLY	0.563	0.575	0.588	454	95
ContextRoute	0.577	0.539	0.580	577	92
LQM-CONTEXTROUTE	**0.618**	**0.580**	**0.631**	645	90

Table 4: StrategyQA high-heterogeneity routing.

Component	search F1	search lat. (ms)	SQA acc	SQA lat. (ms)
Additive-CR	0.573	435	0.547	542
Score only	0.556	398	0.561	407
Score+defl	0.557	395	0.567	435
Score+ctx	0.581	435	0.591	515
Full router	**0.583**	431	**0.596**	565

Table 5: Component ablation; Full router is LQM-CONTEXTROUTE.

provide answers through the same routing interface, but the fast search-based arm is poorly matched to implicit multi-hop boolean reasoning, creating a 52 pp quality gap.

Table 4 reports the result. The load-aware bandit without quality matching drops to 0.412–0.464 accuracy because DDG's low $\tilde{\tau}$ compensates for its near-zero u and draws traffic toward the lower-quality provider. CONTEXTROUTE mitigates this partly (0.539–0.580), while LQM-CONTEXTROUTE reaches 0.580–0.631 and improves mean accuracy by +17.54 pp over SW-UCB. The LQM-ONLY ablation recovers 0.563–0.588, showing that the scoring objective accounts for most of the gain before context is added. Static-T0 has the highest pure-quality ceiling, but at 1238 ms mean latency and 70% SLA.

Companion pools without DDG show the same trend under smaller quality gaps: with a 12.3 pp gap, LQM-CONTEXTROUTE reaches 0.604 accuracy on No-DDG and 0.625 on Synth-3B (App. E).

Table 5 isolates the source of the gain. In the search API pool, adding latency-quality matching to the contextual router improves F1 from 0.573 to 0.583 without increasing mean latency; on StrategyQA, the full router reaches the highest accuracy. The renewal score handles latency-quality compensation, while context captures query-specific quality differences. The strongest additive contextual router from an α sweep remains below LQM-CONTEXTROUTE on both the search API pool (0.5806 vs. 0.5834 F1) and StrategyQA (0.5894 vs. 0.5978 accuracy; App. D).

5.4 Q3: Transfer and boundary cases

Q3 asks whether the latency-quality objective transfers across provider types and where its limits are. We evaluate heterogeneous retriever pools and an LLM-provider ladder, then report deployment checks for live ReAct routing, outage replay, and reward-proxy validation.

Transfer beyond search. We run two retriever-pool checks with provider-quality differences large enough for the objective to matter. On **SciFact** (Wadden et al., 2020), the providers are GTE-BASE (NDCG 0.762), BGE-BASE (0.740), and MINILM (0.645); on **NFCorpus**, they are a slower TF-IDF fusion index (0.302), a word TF-IDF index (0.295), and a fast title-only index (0.215). Table 1 re-

ports both aggregates: LQM-CONTEXTROUTE improves NDCG by +2.91 pp on SciFact and +3.22 pp on NFCorpus over the load-aware online baseline, while remaining above 95% SLA. The LQM-ONLY ablation is slightly stronger on these retriever pools (App. E), suggesting that the generic text context is the limiting component for retrieval routing rather than the latency-quality objective itself; the renewal score still avoids the latency-greedy failure. App. E adds an LLM-provider ladder where better answer providers are slower, and LQM-CONTEXTROUTE improves F1 by +8.89 pp over SW-UCB and +4.64 pp over CONTEXTROUTE by spending latency where the quality return is large.

Deployment checks. Finally, we evaluate several deployment-facing conditions. In a small live ReAct loop with Tavily/Brave/DDG (n=30, max_steps= 3, full fallback), Tavily is dominant and LQM-CONTEXTROUTE matches Static-T1 (0.3690 vs. 0.3681, p=0.33). Bootstrapping the benchmark from 270 live-measured calls gives LQM-CONTEXTROUTE +4.03 pp over Static-T1 (App. D). Outage replay shows that full agent fallback makes the static strong provider competitive, while without fallback all routers recover roughly twice static F1 (Table 8). The 1B judge reward recovers 88–98% of the F1 obtained from a 30B-oracle reward on K=2 (Table 7). Together, these checks define the operating regime: LQM-CONTEXTROUTE helps most when load pressure and quality heterogeneity coexist, remains neutral when one provider dominates, and depends on a usable deployment-time reward proxy.

6 Related Work

Tool and model routing. Tool-augmented agent work studies which tool an agent should call, how tool use is supervised, and how tool outputs enter reasoning (Schick et al., 2023; Patil et al., 2024; Qin et al., 2024; Yao et al., 2023; Xu et al., 2025; Jiang and Ferraro, 2026). Our setting begins after that decision: a gateway routes one selected tool type among functionally equivalent providers behind the same interface, a pattern made common by MCP-style tool interfaces (Anthropic, 2024). LLM-routing benchmarks and systems (Hu et al., 2024; Li et al., 2026; Ong et al., 2024; Jitkrittum et al., 2026) choose among model endpoints, often trading answer quality against inference cost. Routing has also been used for query-level teacher selection in data synthesis (Zhang et al., 2025); our router instead operates online over same-function service providers under runtime load.

Recent online and budgeted routers use bandit feedback, test-time compute, or constrained allocation (Zhang et al., 2024; Panda et al., 2025; Wei et al., 2025; Poon et al., 2025; Ding et al., 2025; Mei et al., 2025; Taberner-Miller, 2026). These lines of work motivate adaptive routing, but the provider-routing setting changes the action space and runtime signal: the choices are providers behind one selected tool interface, and the changing state is service load rather than only model cost. Production gateways such as LiteLLM and Portkey provide priority, cooldown, and fallback policies (BerriAI, 2026; Portkey, 2025b,a); LQM-CONTEXTROUTE studies the learning objective used to trade answer quality against service time within such same-function pools.

Cost-aware bandits. Non-stationary and contextual bandits adapt to changing rewards and query context (Garivier and Moulines, 2011; Kocsis and Szepesvári, 2006; Russac et al., 2019; Li et al., 2010). Bandits with knapsacks, constrained bandits, and CVaR bandits model resource limits or risk (Badanidiyuru et al., 2018; Agrawal and Devanur, 2019; Combes et al., 2015; Lim and Malik, 2022). In our setting, additive utility-cost rewards can rank a fast poor provider above a slow useful one. The renewal score $u/(1+\tau/L_{\text{ref}})$ follows the reward-per-service-cycle view (Ross, 1996), letting latency affect the service cycle rather than enter as an additive reward penalty.

7 Conclusion

We study routing among functionally equivalent tool providers for LLM agents under runtime load. Additive latency-quality rewards can prefer a fast low-quality provider over a slower useful one (Thm. 2). LQM-CONTEXTROUTE replaces reward mixing with the renewal-reward score $u_i/(1 + \tau_i/L_{\text{ref}})$ inside a contextual bandit with online judge feedback. Across a six-pool evaluation matrix spanning search, QA, retriever, and LLM-backend providers, the main gains appear when runtime pressure and provider-quality heterogeneity coexist: LQM-CONTEXTROUTE cuts latency and SLA misses, avoids additive-reward collapse, and remains neutral in stable-dominant pools.

Limitations

The search API pool records provider-response scores before the routing runs and then varies runtime load, so it isolates routing decisions rather than full production traffic. Non-search evidence comes from retriever pools and an LLM-provider ladder, not broad live traffic across many tool categories. The results should be read as evidence about when provider routing matters rather than as an estimate of average production lift.

The method assumes a same-function provider pool and a reliable online quality proxy; it does not address upstream tool selection, multi-tool planning, or provider schema mismatch. Retriever results indicate that the contextual head depends on the query representation, and the online judge validation covers a small K=2 response set.

References

János Aczél. 1966. *Lectures on Functional Equations and Their Applications*. Academic Press.

Shipra Agrawal and Nikhil R. Devanur. 2019. Bandits with concave rewards and convex knapsacks. In *ACM EC*.

Anthropic. 2024. Model Context Protocol Specification. https://modelcontextprotocol.io/specification/2024-11-05/. Protocol revision 2024-11-05; accessed: 2026-05-02.

Ashwinkumar Badanidiyuru, Robert Kleinberg, and Aleksandrs Slivkins. 2018. Bandits with knapsacks. *Journal of the ACM*, 65(3).

BerriAI. 2026. LiteLLM Routing and Load Balancing Documentation. https://docs.litellm.ai/docs/routing. Accessed: 2026-05-02.

Nihir Chadderwala. 2025. Optimizing life sciences agents in real-time using reinforcement learning. *arXiv preprint arXiv:2512.03065*.

Richard Combes, Chong Jiang, and R. Srikant. 2015. Bandits with budgets: Regret lower bounds and optimal algorithms. In *ACM SIGMETRICS*.

Digital Applied. 2026. 100 MCP Servers Stress-Tested: Reliability Findings. https://www.digitalapplied.com/blog/mcp-server-reliability-100-server-stress-test-study. Accessed: 2026-05-02.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian Del Carmen Hipolito Garcia, Menglin Xia, Laks V. S. Lakshmanan, Qingyun Wu, and Victor Rühle. 2025. BEST-Route: Adaptive LLM routing with test-time optimal compute. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 13870–13884.

Aurélien Garivier and Eric Moulines. 2011. On upper-confidence bound policies for switching bandit problems. In *Algorithmic Learning Theory*, pages 174–188.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Aayush Gupta. 2026. ReliabilityBench: Evaluating LLM agent reliability under production-like stress conditions. *arXiv preprint arXiv:2601.06112*.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. 2024. RouterBench: A benchmark for multi-LLM routing system. *arXiv preprint arXiv:2403.12031*.

Yuxuan Jiang and Francis Ferraro. 2026. SCRIBE: Structured mid-level supervision for tool-using language models. *Preprint*, arXiv:2601.03555.

Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Congchao Wang, Zifeng Wang, Alec Go, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. 2026. Universal model routing for efficient LLM inference. In *International Conference on Learning Representations*.

Levente Kocsis and Csaba Szepesvári. 2006. Discounted UCB. In *2nd PASCAL Challenges Workshop*.

Hao Li, Yiqun Zhang, Zhaoyan Guo, Chenxu Wang, Shengji Tang, Qiaosheng Zhang, Yang Chen, Biqing Qi, Peng Ye, Lei Bai, Zhen Wang, and Shuyue Hu. 2026. LLMRouterBench: A massive benchmark and unified framework for LLM routing. *arXiv preprint arXiv:2601.07206*.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *WWW*.

Shiau Hong Lim and Ilyas Malik. 2022. Distributional reinforcement learning for risk-sensitive policies. In *Advances in Neural Information Processing Systems*.

Kai Mei, Wujiang Xu, Minghao Guo, Shuhang Lin, and Yongfeng Zhang. 2025. OmniRouter: Budget and performance controllable multi-LLM routing. *ACM SIGKDD Explorations Newsletter*, 27(2):107–116.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. 2024. RouteLLM: Learning to route LLMs with preference data. *arXiv preprint arXiv:2406.18665*.

Pranoy Panda, Raghav Magazine, Chaitanya Devaguptapu, Sho Takemori, and Vishal Sharma. 2025. Adaptive LLM routing under budget constraints. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 23934–23949. Association for Computational Linguistics.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37.

Manhin Poon, Xiangxiang Dai, Xutong Liu, Fang Kong, John C. S. Lui, and Jinhang Zuo. 2025. Online multi-LLM selection via contextual bandits under unstructured context evolution. *arXiv preprint arXiv:2506.17670*.

Portkey. 2025a. Failover Routing Strategies for LLMs in Production. https://portkey.ai/blog/failover-routing-strategies-for-llms-in-production/. Accessed: 2026-05-02.

Portkey. 2025b. The Most Reliable AI Gateway for Production Systems. https://portkey.ai/blog/the-most-reliable-ai-gateway-for-production-systems/. Accessed: 2026-05-02.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*.

Sheldon M. Ross. 1996. *Stochastic Processes*, 2nd edition. Wiley. Renewal-reward theorem (Theorem 3.6.1).

Yoan Russac, Claire Vernade, and Olivier Cappé. 2019. Weighted linear bandits for non-stationary environments. In *NeurIPS*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.

Annette Taberner-Miller. 2026. ParetoBandit: Budget-paced adaptive routing for non-stationary LLM serving. *arXiv preprint arXiv:2604.00136*.

Nandan Thakur and 1 others. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *NeurIPS Datasets and Benchmarks*.

David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. 2020. Fact or fiction: Verifying scientific claims. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7534–7550. Association for Computational Linguistics.

Wang Wei, Tiankai Yang, Hongjie Chen, Yue Zhao, Franck Dernoncourt, Ryan A. Rossi, and Hoda Eldardiry. 2025. Learning to route LLMs from bandit feedback: One policy, many trade-offs. *arXiv preprint arXiv:2510.07429*.

Ningning Xu, Yuxuan Jiang, Shubhashis Roy Dipta, and Hengyuan Zhang. 2025. Learning how to use tools, not just when: Pattern-aware tool-integrated reasoning. *MATH-AI @ NeurIPS 2025*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Bowen Zhang, Gang Wang, Qi Chen, and Anton van den Hengel. 2024. How do we select right LLM for each query? MAR: Multi-armed recommender for online LLM selection. OpenReview preprint. ICLR 2025 withdrawn submission; OpenReview ID AfA3qNY0Fq.

Hengyuan Zhang, Shiping Yang, Xiao Liang, Chenming Shang, Yuxuan Jiang, Chaofan Tao, Jing Xiong, Hayden Kwok-Hay So, Ruobing Xie, Angel X. Chang, and 1 others. 2025. Find your optimal teacher: Personalized data synthesis via router-guided multi-teacher distillation. *arXiv preprint arXiv:2510.10925*.

A LQM-ContextRoute online algorithm

This appendix gives the full online loop corresponding to Fig. 5 and Eq. 3. The pseudo-code makes explicit the two-stage score computation: quality estimates are computed for all active providers before the cross-provider gap term is formed.

Algorithm 1 LQM-ContextRoute online routing

input ridge λ_r, deflation λ, latency scale L_{ref}
init $A_i \leftarrow \lambda_r I$, $A_i^{-1} \leftarrow \lambda_r^{-1} I$, $\mathbf{b}_i \leftarrow \mathbf{0}$, $\hat{\tau}_i \leftarrow \tau_0$ for each provider i
for $t = 1, \ldots, T$ **do**
  Embed query q_t as $\mathbf{x}_t$
  for each active provider i **do**
    $\hat{u}_i \leftarrow \mathbf{x}_t^\top A_i^{-1} \mathbf{b}_i$
    $c_i \leftarrow \alpha_{\text{ucb}} \sqrt{\mathbf{x}_t^\top A_i^{-1} \mathbf{x}_t}$
  end for
  $\hat{u}_{\max} \leftarrow \max_j \hat{u}_j$
  for each active provider i **do**
    $\Delta_i \leftarrow \max(0, \hat{u}_{\max} - \hat{u}_i)$
    $s_i \leftarrow \hat{u}_i/(1 + \hat{\tau}_i/L_{\text{ref}}) + c_i/(1 + \lambda \Delta_i)$
  end for
  Select $i_t \leftarrow \arg\max_i s_i$ and call provider i_t
  Observe latency τ_t and quality score u_t
  Update $\hat{\tau}_{i_t}$ by EMA
  Add $(\mathbf{x}_t, u_t)$ to provider i_t's window; evict old samples if needed; update $A_{i_t}^{-1}, \mathbf{b}_{i_t}$ by Sherman–Morrison
end for

B Positioning vs. prior LLM-routing work

Table 6 compares LQM-ContextRoute with representative routing work along the axes that determine whether a method directly addresses same-function provider routing: the routed unit, the available deployment feedback, the runtime state being adapted to, and the quality-latency objective.

Work	route unit	deploy fb.	runtime state	u–τ objective
RouteLLM	LLM endpoint	pref.	fixed	additive cost
RouterBench	LLM endpoint	gold	fixed	additive cost
PILOT	LLM endpoint	pref.	budget	additive cost
BaRP	LLM endpoint	bandit	fixed	additive reward
MAR	LLM endpoint	judge	fixed	additive reward
Chadderwala	tool choice	user	latency	additive reward
Ours	**tool provider**	**judge**	**load**	**renewal rate**

Table 6: Positioning by the problem axes used in this paper.

Constrained, knapsack, and Pareto routing. Constrained bandits and bandits with knapsacks model resource limits as budgets over a sequence of decisions (Badanidiyuru et al., 2018; Agrawal and Devanur, 2019; Combes et al., 2015). This view is related to our use of a service-time budget, but it answers a different question. A knapsack formulation controls aggregate resource consumption; it does not specify how latency and answer quality should be ranked for two same-function providers on a single request. An additive Lagrangian or scalarized reward can still prefer a fast low-quality arm when latency compensates for low utility, which is the score-level failure in Theorem 2.

Pareto routing methods and budget-paced LLM routers expose a quality-cost frontier or allocate traffic under a global budget (Mei et al., 2025; Taberner-Miller, 2026). LQM-ContextRoute instead provides a single online selection rule for a gateway that has already selected the tool type and must choose a provider under current load. The renewal-rate score $u/(1+\tau/L_{\text{ref}})$ can be viewed as one operating point on the frontier, with L_{ref} setting the latency scale. This makes the method compatible with budgeted routing layers while keeping the provider-level objective focused on avoiding latency-quality compensation.

C Proofs and regret sketch

This section collects the formal score-level statements used in the method section. The regret sketch applies to the unmodulated optimistic form of the renewal score; the separation and characterisation results concern the score itself and are independent of a particular exploration rule. In the stationary K-armed setting, let $u_i \in [0,1]$ be sub-Gaussian with proxy σ_u^2 and $\tau_i \geq 0$ be sub-Gaussian with proxy σ_τ^2. Define $V_i^\star = u_i^\star/(1 + \tau_i^\star/L_{\text{ref}})$ and $\Delta_i^V = V_{i^\star}^\star - V_i^\star$.

Concentration and regret. For the plug-in estimator $\hat{V}_i = \hat{u}_i/(1 + \hat{\tau}_i/L_{\text{ref}})$, a first-order Taylor expansion gives, with probability at least $1 - \delta$,

$$|\hat{V}_i - V_i^\star| \leq (1 + L_{\text{ref}}^{-1}) \sqrt{\frac{2\sigma^2 \log(1/\delta)}{n_i}} + O(1/n_i),$$

where $\sigma^2 = \max(\sigma_u^2, \sigma_\tau^2)$. The unmodulated optimistic version of the scoring ablation (Eq. 4 with $\lambda = 0$) satisfies

$$R_T \leq \sum_{i:\Delta_i^V > 0} \frac{C(1 + L_{\text{ref}}^{-1})^2 \sigma^2 \log T}{\Delta_i^V} + o(\log T).$$

Sliding-window concentration gives the usual non-stationary additive term $O(\sqrt{T \log T \cdot V_T})$. The guarantee is stated for the unmodulated optimistic form; the $\lambda > 0$ gap modulation used in Eq. 3 is an empirical exploration control for arms whose current quality estimate is dominated.

Characterisation under linear-cycle calibration. Let $S(u, \tau) = T(u, z)$ with $z = \tau/L_{\text{ref}}$. Consider score families that satisfy non-compensation ($T(0, z) = 0$), monotonicity in quality and service time, scale invariance in the normalized latency z, and boundedness for $u \in [0,1]$ and $z \geq 0$. These

assumptions restrict the admissible scores but do not uniquely specify one; for example $u(1+z)^{-\alpha}$ satisfies them for any $\alpha > 0$. The additional structural calibration is that a call is a renewal cycle of duration $1+z$, so the reward rate is $u/(1+z)$ by the renewal-reward theorem (Ross, 1996, Thm. 3.6.1).

Theorem 1 (Characterisation under linear-cycle calibration)**.** *Under the assumptions above plus the linear-cycle calibration, the selected member of the admissible score family (up to monotone transformation of the quality scale) is* $V_i = u_i/(1+\tau_i/L_{ref})$.

Formally, the linear-cycle calibration makes the latency ratio $T(u,z_1)/T(u,z_2)$ depend only on $(1+z_2)/(1+z_1)$; standard multiplicative functional-equation arguments yield $T(u,z) = u(1+z)^{-\alpha}$ (Aczél, 1966, Ch. 3), and the renewal cycle fixes $\alpha = 1$.

Separation from additive composites.

Theorem 2 (Renewal-reward vs. additive separation)**.** *Fix* $\alpha \in (0,1)$ *and let* $r_i^{\text{add}}(\alpha) = \alpha u_i - (1-\alpha)\tilde{\tau}_i$ *with* $\tilde{\tau} = \min\{\tau/L_{ref}, 1\}$. *There exists a two-arm instance where the additive score chooses the lower-quality faster arm while* $V_i = u_i/(1+\tilde{\tau}_i)$ *chooses the higher-quality arm whenever*

$$\frac{u_2\Delta_{\tilde{\tau}}}{1+\tilde{\tau}_2} < \Delta_u < \frac{1-\alpha}{\alpha}\Delta_{\tilde{\tau}}.$$

The interval is non-empty when $u_2/(1+\tilde{\tau}_2) < (1-\alpha)/\alpha$.

Proof sketch: in a two-arm pool with $\tilde{\tau}_1 > \tilde{\tau}_2$ and $\Delta_u = u_1 - u_2$, the additive score chooses the faster lower-quality arm 2 iff $\Delta_u < \frac{1-\alpha}{\alpha}\Delta_{\tilde{\tau}}$. The renewal score chooses arm 1 iff $\Delta_u > u_2\Delta_{\tilde{\tau}}/(1+\tilde{\tau}_2)$. Hence the two rankings disagree exactly when

$$\frac{u_2\Delta_{\tilde{\tau}}}{1+\tilde{\tau}_2} < \Delta_u < \frac{1-\alpha}{\alpha}\Delta_{\tilde{\tau}},$$

which is non-empty whenever $u_2/(1+\tilde{\tau}_2) < (1-\alpha)/\alpha$.

D Deployment and robustness details

The following checks provide the deployment-facing details summarized in Q3: whether the online reward proxy is sufficient for closed-loop learning, how routing behaves under agent fallback and outage replay, how sensitive the method is to L_{ref}, and how the measured latency replay is constructed.

LLM-as-judge closed-loop validation. On K=2 (Brave + DDG, the two providers with stored raw responses), CONTEXTROUTE trained on the 1B LLM-judge reward recovers 88–98% of the F1 obtained when training on the 30B-oracle reward (Table 7). Because this response set is Brave-dominant, the experiment is used to measure reward-proxy recovery rather than policy ranking.

Pattern	30B oracle	1B judge	recovery
step	0.533	0.468	87.8
rotation	0.508	0.496	97.6
spike	0.535	0.496	92.8
gradual	0.496	0.460	92.6

Table 7: Closed-loop reward-proxy validation for CONTEXTROUTE on the K=2 Brave/DDG response set. Recovery is the F1 obtained with the 1B judge reward as a percentage of the F1 obtained with the 30B-oracle reward.

ReAct/outage replay. For each of 200 HotpotQA dev questions and each of {Brave, DDG}, we ran the Qwen-2.5-7B agent with up to 3 ReAct steps and stored the final answer, F1, and tool latency. The forced-search replay in §5.4 uses $n = 100$ questions, 30 seeds, and a sustained Brave outage at $T/2$. Table 8 reports the aggregate F1. With full fallback, the static strong-provider policy remains competitive; without fallback, online routing substantially improves recovery.

Regime	Static	SW	CR	LQM-CONTEXTROUTE
fallback	0.400	0.295	0.311	0.304
no fallback	0.134	**0.261**	0.256	0.254

Table 8: ReAct outage replay: aggregate F1 under a sustained Brave outage, with and without the agent's own fallback behavior.

Reference-latency sensitivity. The experiments use $L_{\text{ref}} = 1500$ ms, matching the SLA threshold. To check whether this choice drives the result, we rerun only LQM-CONTEXTROUTE on the HotpotQA+TriviaQA search API pool with $L_{\text{ref}} \in \{750, 1500, 3000\}$ ms. Mean F1 varies by at most 0.29 pp and SLA remains within 97.9–98.0% (Table 9).

Additive-reward alpha sweep. To test additive tuning, we sweep the quality weight $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ for both SW-UCB and the contextual additive router. Table 10 reports the strongest additive variants by quality. On the search API pool, the best additive contextual router nearly

L_{ref}	F1	lat. (ms)	SLA
750	0.581	432	98.0
1500	0.579	439	97.9
3000	0.582	436	98.0

Table 9: LQM-CONTEXTROUTE sensitivity to the reference latency on the HotpotQA+TriviaQA search API pool.

matches LQM-CONTEXTROUTE but has slightly lower F1 and higher latency. On StrategyQA, where the low-latency provider has much lower quality, increasing α helps but does not remove the additive-compensation failure.

Dataset	Router	quality	lat. (ms)	Δqual.
Search	LQM-CONTEXTROUTE	0.5834	431	–
Search	CR($\alpha = 0.9$)	0.5806	444	-0.28
Search	SW($\alpha = 0.5$)	0.5592	337	-2.43
StrategyQA	LQM-CONTEXTROUTE	0.5978	566	–
StrategyQA	CR($\alpha = 0.9$)	0.5894	568	-0.84
StrategyQA	SW($\alpha = 0.9$)	0.5652	424	-3.26

Table 10: Best additive baselines under an α sweep. Quality is F1 for the search API pool and accuracy for StrategyQA; Δquality is relative to LQM-CONTEXTROUTE.

Live latency profile and real-bootstrap setup. We profile Tavily, Brave, and DuckDuckGo on 30 HotpotQA queries under three conditions: idle (sequential), moderate (200 ms inter-call sleep), and stressed (3 concurrent threads with 800 ms inter-batch sleep). All 270 calls succeeded. The real-bootstrap replay samples latencies with replacement from the corresponding empirical pool and maps the load bins to idle, moderate, or stressed without interpolation.

Tool	condition	mean (ms)	p50 (ms)	p95 (ms)
Tavily	idle	1650	1477	2817
Tavily	moderate	149	76	87
Tavily	stressed	175	79	159
Brave	idle	680	715	809
Brave	moderate	342	316	405
Brave	stressed	320	305	482
DDG	idle	2790	2737	3415
DDG	moderate	2584	2483	2927
DDG	stressed	2151	1955	2903

Table 11: Live latency profile used for the real-bootstrap replay. Latency statistics are in milliseconds over 30 calls per provider-condition pair.

On HotpotQA K=3 ($T = 200$, 50 seeds, 4 load patterns), the real-bootstrap run gives LQM-CONTEXTROUTE F1 0.5823 (Static-T1 0.5420, $\Delta = +4.03$ pp), SW-UCB 0.5809, CONTEXTROUTE 0.5739, Latency-Oracle 0.5418, and F1-Oracle 0.6750. This replay combines measured latency samples with the same provider-response scores used in the search API pool, strengthening the load evidence without claiming a full production deployment.

E Additional diagnostics

These diagnostics complement the main evaluation by varying the degree of provider-quality heterogeneity and by reporting additional provider-pool breakdowns. They are not separate task formulations; each keeps the same same-function routing setup used in the main experiments.

StrategyQA no-DDG companions. This companion analysis reduces the provider-quality gap while preserving the same routing interface. Table 12 reports mean accuracy over four load patterns. Per-pattern results follow the same trend: on Pool-synth-3B, LQM-CONTEXTROUTE wins all four patterns over CONTEXTROUTE; on Pool-noddg, it wins all four over CONTEXTROUTE and three of four over SW-UCB.

Pool	SW	CR	LQM-ONLY	LQM-CONTEXTROUTE
No-DDG	0.586	0.577	0.581	**0.604**
Synth-3B	0.587	0.595	0.594	**0.625**

Table 12: StrategyQA companion pools with no weak DDG arm. Scores are mean accuracy over four load patterns.

SciFact retriever breakdown. Table 13 gives the load-pattern breakdown for the SciFact heterogeneous retriever pool. Per-retriever NDCG@10 on 300 test claims is from BEIR (Thakur et al., 2021); routing then selects among these retrievers under the same non-stationary load patterns used in the main evaluation.

	STEP		ROTATION		GRADUAL	
Router	NDCG	lat. (ms)	NDCG	lat. (ms)	NDCG	lat. (ms)
Static-T0	0.692	909	0.678	1129	0.631	1665
EMA-Greedy	0.672	46	0.686	134	0.672	46
SW-UCB	0.672	46	0.691	181	0.672	46
LQM-ONLY	0.728	205	**0.735**	255	**0.718**	240
ContextRoute	**0.732**	260	0.699	449	0.715	330
LQM-CONTEXTROUTE	0.717	102	0.707	365	0.717	102

Table 13: SciFact heterogeneous retriever pool. Each cell reports NDCG and mean latency under the corresponding load pattern.

LLM-provider ladder. As a non-search same-function pool, we route among 1B, 7B, and 30B

pre-run answer providers over the same 200 questions. This setting reverses the web-search trade-off: the higher-quality providers are slower. LQM-CONTEXTROUTE improves F1 by $+8.89$ pp over SW-UCB and $+4.64$ pp over CONTEXTROUTE, with $+526$ and $+90$ ms higher mean latency, respectively. This pool tests the same objective when higher-quality providers require spending additional latency.

Reactive-Cooldown. We also compare against a LiteLLM-style priority+cooldown router in a K=10 stress setting. On K=10 STEP, its F1 varies by 0.605 across operator priority orderings; CONTEXTROUTE matches the priority-tuned setting without manual ordering. This supports the claim that quality-aware online routing complements reactive fallback mechanisms.