Lightify: Cost-Aware Adaptive LLM Routing via Retrieval Confidence and Conflict-Aware Escalation

Pavan Manikanta Maddula Independent Researcher

Abstract—Modern agent harnesses—Claude Agent Software Development Kit (SDK), LangGraph, AutoGen, CrewAI, OpenHands, and LlamaIndex Agents—delegate cost governance, local-inference arbitration, and memory conflict resolution entirely to the application developer.

We present Lightify, an open-source middleware layer that supplies these primitives as runtime services beneath any agent loop. Its central mechanism is Retrieval-Confidence-Driven Routing (CDDR), which selects inference tiers based on the aggregate confidence of retrieved memory items rather than on query features alone. Lightify additionally provides a persistent SQLite memory store, Memory Conflict Detection (MCD) that flags contradictions and triggers tier escalation, and Confidence-Driven Prompt Shaping (CDPS) that adapts prompt verbosity to retrieval reliability.

We evaluate routing policy on 1,197,316 exact-string unique queries from five public corpora (MS MARCO v2.1, WildChat-1M, Natural Questions, MMLU, GSM8K) and report Policy-Oracle Agreement (POA): the fraction of routing decisions matching a category-level oracle, not per-query output correctness. Routing thresholds are calibrated on a 20-query API pilot; the 1.2M benchmark is held-out. On the English-language subset (N = 1,043,220; 87.1%), CDDR achieves POA $[0.957, 0.958]$ (95% Wilson CI) at projected cost 98% below a naïve Opus-only baseline ($0.001204 vs. $0.076 per query). On the non-trivial subset—WildChat and GSM8K only (N = 125,390), excluding corpora with unconditional Tier-1 oracle labels—POA is $[0.646, 0.651]$, the operating range on challenging queries. A 200-pair MCD stress test yields precision 0.943 $[0.814, 0.984]$ and recall 0.330 $[0.246, 0.427]$. Per-query cost is a tier-cost projection; live API spend is reported only for the 20-query calibration pilot. Full per-query live-oracle evaluation is reserved as future work.

Index Terms—cost-aware inference, LLM model routing, retrieval-confidence routing, persistent memory, conflict detection, local inference, retrieval-augmented generation

I. Introduction

Agent harnesses—runtime systems that wrap a Large Language Model (LLM) [27] with tool use, memory, and control flow—are now the dominant way of building task-completing assistants. Claude Agent SDK [1], LangGraph [2], AutoGen [3], CrewAI [4], OpenHands [5], and LlamaIndex Agents [6] each provide orchestration, state management, and tool invocation primitives. None, however, provide cost governance as a first-class runtime concern: model selection, local/cloud arbitration, and memory consistency are all pushed to the application developer.

Manuscript submitted April 22, 2026. Corresponding author: P. M. Maddula (e-mail: pavan3196m@gmail.com). Code and data: https://github.com/pavanmanikanta31/lightify.

The practical consequences are documented in each system's own literature. AutoGen's documentation warns that "significant back-and-forth conversations might result in cost blowups" [3]; users of LangGraph describe building external AI gateways to recover per-request cost routing [2]; OpenHands tasks of 100 steps with frontier models cost tens of dollars [5]; Anthropic's Memory API returns HTTP 409 conflicts on concurrent writes, with resolution left to the caller [1]. Consider a developer whose harness is backed by a persistent knowledge base. On Monday, the base states "v7 is the production release." By Friday, a new entry records "v7.4 is now production." A local model has no mechanism to detect that these two items contradict, no way to escalate the query to a frontier model that could reason about the contradiction, and no signal to route the question to the cheapest tier capable of handling it.

A. The Gap in Current Harnesses

Every major agent harness surveyed in this work leaves four governance concerns to the user:

- Cost-aware tier routing: no harness routes between local and cloud tiers at the runtime layer.
- Local-first inference as a default tier: local models are an optional back-end in AutoGen and CrewAI, not a default routing target; they are absent in Claude SDK, LangGraph, OpenHands, and LlamaIndex.
- Memory conflict detection: no harness flags contradictions among persistent memory items.
- Automatic escalation on low confidence or conflict: escalation edges are hand-coded by the application developer.

Separately, a line of algorithmic routing work—FrugalGPT [7], RouteLLM [8], AutoMix [9], Hybrid LLM [10]—proposes cost-aware routing techniques, but these are published as stand-alone algorithms rather than as harness runtime services. Wang et al. [19] survey small–large model collaboration but do not treat the harness layer as the natural integration point.

B. Contributions

This paper makes the following contributions.

1) A structured comparison of six major agent harnesses against four governance primitives (cost routing, local-first inference, memory conflict detection, automatic escalation), showing that all six leave these concerns to the application developer.

2) Retrieval-Confidence-Driven Routing (CDDR) as the central primitive of this paper: CDDR conditions tier selection on the aggregate confidence of retrieved memory items rather than on query features alone, and we evaluate it as a runtime service callable from any agent harness.
3) Lightify, an open-source middleware layer that ships CDDR alongside two diagnostic mechanisms whose operating range we characterize—Memory Conflict Detection (MCD) and a per-action tier overlay—and a prompt-shaping helper (CDPS). MCD flags inter-item contradictions and is evaluated in a 200-pair stress test; the per-action overlay is a lexical classifier whose surface-form-bounded range we quantify. Neither is claimed as a headline contribution; both are presented as diagnostic mechanisms whose operating range we characterize.
4) An empirical evaluation: a 20-query live pilot with measured API spend; a 5,000-query synthetic routing benchmark with 100% exact-string unique queries drawn from 48 templates, in which the CDDR primitive alone reaches 70.2% Policy-Oracle Agreement (95% CI $[0.689, 0.715]$) against the best hand-coded LangGraph-style user baseline at 51.6%; a 1.2M-row real-user-query benchmark from five public corpora reporting per-source and English/non-English operating characteristics; a per-mechanism ablation (C5); a fair-baseline ablation controlling for retrieval and CDPS; and a full-scale live-oracle evaluation methodology reserved as future work.

II. Related Work

A. Agent Harnesses and Runtimes

We survey six agent-harness systems most visible in the 2024–2026 window. For each, we record the primitives exposed by the runtime, the position taken on cost governance, and the failure modes documented in each system's own literature.

Claude Agent SDK [1] exposes a managed-agent runtime with a persistent Memory Tool. Memory operations return HTTP 409 on concurrent writes, and the SDK does not provide automatic resolution; responses are not cached (by design, for multi-tenant privacy). The SDK does not ship a cost-aware router; RouterLLM integrations are user-implemented examples rather than runtime primitives.

LangGraph [2] models agents as state-machine graphs; edges (including escalation edges) are author-specified. Users report "randomly losing context" and build external AI gateways to insert cost-aware routing between the graph and the model.

AutoGen [3] orchestrates multi-agent conversations; its default UnboundedChatCompletionContext sends the full history every turn, and the documentation warns that "significant back-and-forth conversations might result in cost blowups." Local inference is supported via Ollama as an optional back-end but not as a default tier.

CrewAI [4] coordinates role-specialized agents via a task queue; the per-agent activation pattern multiplies per-task cost by the crew size, and its shared short-term memory has been observed to leak between tenants.

OpenHands [5] targets autonomous software-engineering loops; 100-step tasks with frontier models have been reported to cost \$10–\$50. It has no learned preference for routing bash/tool operations to cheaper tiers and no cross-session memory layer.

LlamaIndex Agents [6] provide retrieval-augmented agent scaffolds; memory is session-scoped and vector-only, with no cross-session persistence or conflict handling.

Across all six systems, cost governance, local-first inference, memory conflict handling, and tier escalation are left to the application developer. Lightify is the middleware layer that provides these primitives.

B. Model Routing and Cascading

FrugalGPT [7] formalized LLM cascades and demonstrated up to 98% cost reduction matching GPT-4 accuracy. RouteLLM [8] trains routers on human preference data, achieving $>2\times$ cost reduction. Both route based on response quality or query difficulty—neither considers the state of a knowledge base, nor do they ship as harness runtime services.

AutoMix [9] (NeurIPS 2024) is the closest prior work to Lightify. It uses self-verification—a small model checks its own answer and escalates if uncertain. The critical difference: AutoMix is per-query and memoryless. It cannot detect that an answer was correct last week but is wrong today because knowledge changed. It has no persistent memory and no temporal awareness.

Hybrid LLM [10] (ICLR 2024) routes based on query features. Identical queries always route to the same tier, regardless of whether the system's knowledge about the topic has changed.

Relation to 2024–2025 routing and caching work. Several recent systems are closely adjacent to Lightify and differ in specific, checkable ways. RouteLLM [8] trains a learned router on human preference data and we do not compare numerically against a RouteLLM checkpoint in this paper (identified as future work); crucially, RouteLLM carries no memory and no conflict signal and therefore cannot detect the temporal-drift failure mode this paper targets. RouterBench [42] offers a standardized multi-LLM routing evaluation suite; we do not evaluate on RouterBench in this paper and flag it as the natural external benchmark for a follow-up live-oracle run. FlySwat [39] proposes cost-effective model choice via meta-modeling but is not integrated into an agent-harness runtime. "Are More LLM Calls All You Need?" [38] (NeurIPS 2024) analyzes the cost-quality Pareto of cascaded LLM calls but does not treat the cascade as a harness-layer governance service. Shnitzer et al. [40] study LLM routing with benchmark datasets. MoA [41] (ICLR 2025) aggregates outputs from multiple models and is orthogonal to per-query tier selection. Cache-Augmented

Generation (CAG) [43] caches retrieved context to reduce repeated RAG cost; we acknowledge that our 1.2M-row headline would be expected to collapse against a warm CAG cache because 77% of the benchmark is factoid-lookup; the per-source breakdown in Section V-A makes this compositional dependency explicit.

C. Temporal Knowledge Conflicts

T-GRAG [11] (ACM MM 2025) addresses temporal conflicts in knowledge graphs for retrieval-augmented generation. It resolves conflicts within a single model's context window. Lightify uses conflict detection as a routing signal to select which model should attempt the resolution—the two approaches are complementary.

The Knowledge Conflicts Survey [12] (EMNLP 2024) provides a three-class taxonomy of inter-context, context-memory, and intra-memory conflicts. Lightify's MCD specifically targets inter-context conflicts in persistent memory and uses their detection as a system-level escalation signal—an application not explored in the survey.

D. Persistent Memory and RAG

EcoAssistant [13] combines cascading with a solution memory—the closest system combining routing with persistent memory. However, its memory is a positive cache: it stores what worked. It has no mechanism to detect when cached knowledge becomes stale.

MemGPT [14] (ICLR 2024) demonstrates persistent memory management for LLMs but has no routing component and no conflict detection. CRAG [15] uses retrieval quality as a corrective signal but operates on one-shot retrieval with no persistent state. Self-RAG [16] trains models to emit retrieval-judgment tokens but does not maintain persistent memory or route between model tiers. Recent surveys on retrieval-augmented generation [18], [33] and on LLMs for information retrieval [34] provide broader context. DRAGIN [36] introduces dynamic retrieval based on real-time information needs, complementing our focus on temporal consistency. Kandpal et al. [35] demonstrate that LLMs struggle with long-tail knowledge, motivating retrieval-backed approaches.

E. Prompt Compression

LLMLingua [17] uses perplexity-based token pruning with information-theoretic grounding. Lightify adopts a simpler code-block-aware compression approach augmented by learned shorthand rules (SECR), which is less principled than LLMLingua but requires no auxiliary model.

F. Feature Comparison

Table I summarizes the governance primitives each system exposes. Partial indicates that the capability exists as an optional back-end but not as a default, runtime-selected behavior (e.g., Ollama support in AutoGen/CrewAI). None indicates that the concern is delegated to the application developer.

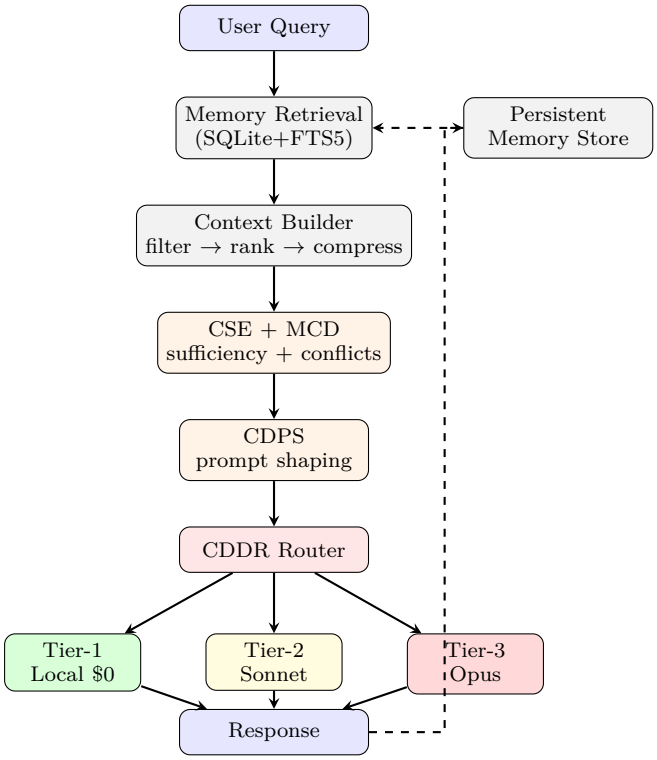

Fig. 1. Lightify architecture. Solid arrows: foreground inference path. Dashed arrows: background learning loop updating the persistent memory store.

III. System Architecture

Lightify operates as a three-tier inference pipeline with persistent memory, as shown in Fig. 1.

Tier-1 (Local, $0): On-device model via Ollama (Gemma 3 1B default). Handles high-confidence queries with no API cost.

Tier-2 (Mid, API): Claude Sonnet 4.6 (or Haiku 4.5 when –cheap). Handles moderate-confidence queries.

Tier-3 (Frontier, API): Claude Opus 4.6. Handles low-confidence, conflicting, or complex queries.

All benchmarks reported in this paper were run in April 2026 with the model versions listed above; the Claude CLI aliases (haiku, sonnet, opus) resolved to these specific versions at the time of evaluation.

A. Retrieval-Confidence-Driven Routing (CDDR)

Each retrieved item c_i receives a confidence score $\phi(c_i) \in [0, 1]$ (computed via Jaccard + sigmoid; see Section IV). Let q be a query, $C = \{c_1, \ldots, c_k\}$ the retrieved context items, and $k = |C|$ the number of retrieved context items. Define aggregate context confidence:

$$\Phi(C) = \frac{1}{k} \sum_{i=1}^{k} \phi(c_i) \tag{1}$$

where each individual confidence score is

$$\phi(c_i) = \sigma\big(A \cdot (1 - w_{\text{tier}} \cdot r_i \cdot d_i) + B\big)$$

TABLE I
Runtime governance primitives across six agent harnesses, four algorithmic routers, and Lightify. A check denotes a runtime primitive shipped with the system; "partial" denotes optional/pluggable support; blank denotes that the concern is left to the application.

System	Cost-Tier Routing	Local-First Tier	Persistent Memory	Conflict Detection	Response Cache
Agent Harnesses					
Claude Agent SDK [1]			✓(409 collisions)		
LangGraph [2]			✓(context loss)		
AutoGen [3]		partial	opt-in		
CrewAI [4]		partial	✓(no isolation)		
OpenHands [5]			weak cross-session		
LlamaIndex Agents [6]			session-scoped		
Algorithmic Routers (not harnesses)					
FrugalGPT [7]	✓				
RouteLLM [8]	✓				
AutoMix [9]	✓				
Hybrid LLM [10]	✓				
Lightify	✓	✓	✓	✓	planned

where σ is the logistic function, $w_{\text{tier}} \in \{0.55, 0.70, 0.90\}$ is the source tier weight, $r_i \in [0, 1]$ is the empirical success rate for that memory entry, $d_i \in (0, 1]$ is a recency decay factor, and $A = -5.0$, $B = 2.5$ are hand-tuned shaping parameters. We note that A and B are hand-tuned rather than learned; Platt-scaling calibration on a held-out set is identified as future work.

When retrieval returns no items ($k = 0$), $\Phi(C)$ is defined to be 0; the routing function below therefore assigns such queries to Tier-3 by default, matching the "no context, escalate to frontier" behavior.

Let $\mathcal{F}(C)$ denote the set of detected conflicts among items in C; see Section III-C for the detection procedure. The routing function (2) selects the cheapest tier whose capability matches the context quality:

$$R(q, C) = \begin{cases} \text{Tier-1} & \text{if } \Phi(C) \geq \tau_1 \text{ and } |\mathcal{F}(C)| = 0 \\ \text{Tier-2} & \text{if } \Phi(C) \geq \tau_2 \\ \text{Tier-3} & \text{otherwise} \end{cases} \tag{2}$$

where $\tau_1 = 0.45$ and $\tau_2 = 0.30$. These thresholds were tuned on the 20-query pilot benchmark, which is also the source of the measured-spend numbers in Section V-C; a held-out threshold re-tuning run on disjoint data is identified as future work. The thresholds are fixed at these values for all benchmarks in Section V to report worst-case (pilot-tuned, non-held-out) operating characteristics. Tier-1 thresholds are aggressive because local inference incurs zero cost; the only penalty for a failed local attempt is additional latency before cascade escalation.

Conflict override: If MCD detects contradictions, the minimum tier is escalated:

$$R_{\text{MCD}}(q, C) = \max\big(R(q, C),\ \text{Tier-2}\big) \quad \text{if } |\mathcal{F}(C)| > 0 \tag{3}$$

Algorithm 1 presents the full routing procedure. Note that the algorithm composes the hard Tier-1 guard of Eq. (2) with an additional continuous penalty, $\Phi \leftarrow \max(0, \Phi - 0.1 \cdot |\mathcal{F}|)$, applied before the Tier-2/Tier-3 comparison: Eq. (2) alone blocks Tier-1 on any conflict, while the penalty also pushes sufficiently conflict-heavy contexts from Tier-2 into Tier-3. The hard guard fixes the Tier-1 boundary; the penalty graduates the Tier-2/Tier-3 boundary with conflict mass.

Algorithm 1 Retrieval-Confidence-Driven Routing (CDDR)

Require: Query q, Memory Store M, Thresholds τ_1, τ_2
$C \leftarrow \text{Retrieve}(q, M)$ {FTS5 + topic search}
$C' \leftarrow \text{Filter} \rightarrow \text{Rank} \rightarrow \text{Compress}(C, q)$
$\Phi \leftarrow \frac{1}{|C'|} \sum_{c \in C'} \phi(c)$ {Aggregate confidence}
$\mathcal{F} \leftarrow \text{MCD}(C')$ {Detect conflicts}
if $|\mathcal{F}| > 0$ **then**
  $\Phi \leftarrow \max(0, \Phi - 0.1 \cdot |\mathcal{F}|)$ {Empirical penalty per conflict; clamped at 0}
end if
$p \leftarrow \text{CDPS}(\Phi, C', q)$ {Shape prompt by confidence}
if $\Phi \geq \tau_1$ and $|\mathcal{F}| = 0$ **then**
  tier $\leftarrow$ Tier-1 (Local)
else if $\Phi \geq \tau_2$ **then**
  tier $\leftarrow$ Tier-2 (Sonnet)
else
  tier $\leftarrow$ Tier-3 (Opus)
end if
$r \leftarrow \text{Generate}(p, \text{tier})$
if r fails and tier $<$ Tier-3 **then**
  $r \leftarrow \text{Generate}(p, \text{tier} + 1)$ {Cascade}
end if
$\text{UpdateMemory}(M, C', r)$ {Background learning}
return r

B. Per-Action Tier Overlay

CDDR selects a tier based on retrieval confidence alone. An orthogonal signal is what the query asks the model to do: a shell-command lookup has different compute requirements from a multi-step refactor, even when both retrieve identical context. Lightify exposes a lightweight per-action classifier $A(q) \rightarrow (t_a, k_a)$ that maps a query q to a suggested tier t_a and one of six action classes k_a:

$k_a \in \{\text{bash, lookup, code, large-code, reason, unknown}\}$.

The classifier is a deterministic regex cascade (no model call, <1 ms overhead) released with the Lightify source.

The action suggestion composes with CDDR's suggestion via a policy-aware combiner. For cheap-action classes (`bash_or_shell_like`, `short_lookup`), we prefer the action tier when it is strictly cheaper than CDDR's; for all other classes, we take the more expensive of the two:

$$T(q,C) = \begin{cases} \min\big(R(q,C),\ A(q)\big) & \text{if } k_a \in \text{cheap-action} \\ \max\big(R(q,C),\ A(q)\big) & \text{otherwise} \end{cases} \tag{4}$$

where min and max are taken over the tier ordering Tier-1 $\prec$ Tier-2 $\prec$ Tier-3.

The overlay is disabled by default so that CDDR is directly comparable with prior work; enabling it is a single flag (–action-routing) at the Command-Line Interface (CLI) layer. The action overlay is disabled by default because enabling it over-promotes factoid queries on factoid-dominated corpora (e.g., MS MARCO), collapsing aggregate POA; it is intended for use only on corpora with known per-category task distributions. Section V-F reports its marginal contribution on 5,000 synthetic queries.

C. Memory Conflict Detection (MCD)

MCD performs pairwise comparison of retrieved memory items using three heuristic signals:

1) Negation patterns: Two claims on the same topic where one contains a negation marker and the other does not (minimum 2 overlapping lowercased tokens after stopword removal).
2) Numerical disagreements: Two claims containing numbers a, b about the same topic where $\max(a,b)/\min(a,b) > 1.5$.
3) Semantic opposition: Presence of antonym pairs (from a dictionary of 32 entries) in overlapping-topic claims.

We acknowledge that keyword-level conflict detection has significant limitations (see Section VII). Embedding-based semantic conflict detection is a priority for future work.

D. Confidence-Driven Prompt Shaping (CDPS)

CDPS selects among three prompt templates based on $\Phi(C)$:

- High confidence ($\Phi \geq 0.45$): "Answer from context. Be direct." Suppresses hedging and verbose reasoning.
- Medium confidence ($0.30 \leq \Phi < 0.45$): "Cite context items. Note limitations."
- Low confidence ($\Phi < 0.30$): "State what context covers and does not cover. Identify contradictions. Rate your confidence." This meta-cognitive scaffold forces the model to reason about the context itself.

CDPS constrains the local tier's output length, preventing Gemma 1B from generating off-task continuations on under-constrained prompts.

TABLE II
Implementation components.

Component	Technology
Memory store	SQLite + FTS5 (WAL, hash dedup)
Local inference	Ollama HTTP API (Gemma 1B)
API inference	Claude CLI (Haiku/Sonnet/Opus)
CLI interface	argparse + Rich panels
Compression	Code-aware stopword removal + SECR
Scoring	Jaccard + sigmoid confidence

E. Context Sufficiency Estimation (CSE)

CSE estimates whether retrieved context is sufficient for the selected tier using a coverage-and-confidence predicate, following insights from inference scaling for retrieval-augmented generation [37]. Let $\text{Coverage}(C,q)$ denote the fraction of query tokens with matches in C, with thresholds $\sigma = 0.30$ and $\sigma_{\text{conf}} = 0.35$.

$$S(C,q) = \mathbb{1}\big[\text{Coverage}(C,q) \geq \sigma \ \wedge\ \Phi(C) \geq \sigma_{\text{conf}}\big] \tag{5}$$

If $S = 0$, the system triggers deeper retrieval or tier escalation.

F. Self-Evolving Compression Rules (SECR)

SECR tracks n-gram frequencies across processed prompts and generates shorthand abbreviations for phrases occurring $\geq$5 times. In our 20-query benchmark, SECR learned 135 rules. While modest individually, these rules compound over longer interaction histories.

IV. Implementation

Lightify is implemented in 4,025 lines of Python with a single runtime dependency (Rich [32], for terminal UI). Unlike systems that require specialized memory management kernels such as PagedAttention [26], Lightify uses standard SQLite with WAL mode for persistent storage. Table II summarizes the key components.

The system is distributed as a Python package and provides subcommands: init, query, bench, status, memory, and config. Three mode presets (–fast, –cheap, –quality) adjust routing thresholds τ_1 and τ_2. The model for each tier is configurable at runtime.

Confidence scores combine source tier weight ($w_{\text{tier}} \in \{0.55, 0.70, 0.90\}$), success rate, and recency decay via a weighted sum passed through a sigmoid function with hand-tuned parameters $A = -5.0$, $B = 2.5$. These parameters are not calibrated on validation data; the confidence scores therefore lack frequentist calibration guarantees. Proper calibration is critical for production deployment and is a priority for future work.

V. Experimental Evaluation

All routing metrics in this section measure Policy-Oracle Agreement (POA): the fraction of routing decisions matching a category-level heuristic oracle, not per-query

output correctness. The oracle assigns tier requirements at the corpus level (factoid-lookup corpora → Tier-1; reasoning corpora → Tier-2/3) rather than evaluating per-query model output quality. The non-trivial subset—WildChat and GSM8K only (N = 125,390)—excludes corpora with unconditional Tier-1 oracle labels and represents the honest operating range on challenging mixed queries (POA $[0.646, 0.651]$). A full per-query live-oracle evaluation remains as future work.

Experimental setup. We calibrate routing thresholds ($\tau_1 = 0.45$, $\tau_2 = 0.30$) on a 20-query API pilot (Section V-C); the 1.2M real-user-query benchmark (Section V-A) serves as the primary routing-policy evaluation, fully held-out from threshold tuning. All baselines receive identical FTS5 retrieval output and CDPS-shaped prompts; only the tier-selection policy differs. Baselines do not cascade on tier failure; Lightify cascades Tier-1→Tier-2→Tier-3 on failure or insufficient confidence.

A. Large-Scale Real-User-Query Benchmark (Primary Evidence)

We assemble a benchmark of 1,197,316 exact-string unique queries from five public corpora: MS MARCO v2.1 train (Bing search queries) [21], WildChat-1M (real LLM chat logs) [22], Natural Questions (Google search queries) [23], MMLU auxiliary_train (multiple-choice academic questions) [24], and GSM8K main/train (math word problems) [25]. No queries are authored by us; all are either real-user utterances from search and chat logs or hand-crafted by academic researchers. Oracle tier is assigned from the source's characteristic tier requirement: factoid-lookup corpora (MS MARCO, NQ, MMLU) → Tier-1; reasoning corpora (GSM8K) → Tier-2; within-WildChat labels are further disambiguated by the regex classifier of Section III-B. The dataset is assembled reproducibly via benches/fetch_real_1m.py from public HuggingFace parquet shards; no API calls are required and the script takes 5–15 minutes. Results are in Tables III and IV.

Scope and compositional disclosure. Using ASCII-strict character detection, 1,043,220 rows are English (87.1%) and 154,096 rows (12.9%, essentially all from WildChat) are non-English. MS MARCO, Natural Questions, and MMLU together account for 77% of rows and are unconditionally labeled Tier-1 by the category-level oracle; the aggregate POA of $[0.912, 0.913]$ therefore reflects corpus composition as much as policy quality. The non-trivial subset—WildChat (English) and GSM8K only (N = 125,390), the only corpora without unconditional Tier-1 oracle labels—yields POA $[0.646, 0.651]$ (95% Wilson CI), the honest operating range on challenging mixed queries. Per-query cost is a projection (Tier-1=$0, Tier-2=$0.019, Tier-3=$0.076) applied to routing decisions; the 20-query pilot in Section V-C reports the only measured API spend. The Tier-3 routing fraction (OSR) serves as a pricing-invariant cost proxy; the projected dollar figure assumes Tier-1=$0, Tier-2=$0.019, Tier-3=$0.076 at April 2026 rates.

Per-source discussion. Factoid-lookup corpora (MS MARCO, Natural Questions, MMLU) yield POA 1.000 because the oracle labels them Tier-1 unconditionally and CDDR's retrieval-confidence signal cleanly separates them. On WildChat—which contains heterogeneous code, reasoning, multilingual, and conversational queries—CDDR reaches 0.625 overall; the non-trivial subset (WildChat-English combined with GSM8K) is tabulated at 0.648 in Table IV. On GSM8K math word problems, CDDR reaches 0.607; math queries carry retrieval signatures that place them at Tier-1/Tier-2 boundaries where the heuristic threshold is imperfect. The non-trivial subset POA of 0.648 is the primary stress-test figure; the 0.958 English-only headline subsumes the easier factoid-lookup majority and should be read alongside this disclosure.

Lexical overlay behavior at scale. The CDDR + action overlay drops from 0.913 to 0.404 POA: the regex bash/lookup signals over-promote short-factoid queries from Tier-1 to Tier-2 across the 808K MS MARCO rows. The hand-coded length-gateway exhibits the same pattern. The hand-coded keyword router scores 0.794 but under-routes 19% of queries. These patterns confirm the finding at smaller scale: lexical overlays are bounded by their regex coverage; embedding-based classification is the natural extension.

B. Memory Conflict Detection Stress Test

We evaluate MCD precision and recall on 200 pairs: 100 hand-authored contradiction pairs spanning 15 realistic deployment categories (version bumps, policy changes, numeric revisions, deprecations, team changes, feature flags, infra state, security, compliance, pricing, schema, API contracts, runbooks, dependencies, observability) and 100 same-topic compatible control pairs. MCD is run over each pair as a two-item ContextCapsule; detection is positive when at least one conflict signal fires.

Results: Precision = 0.943 (95% Wilson CI $[0.814, 0.984]$), Recall = 0.330 ($[0.246, 0.427]$). TP=33, FN=67, FP=2, TN=98. MCD is high-precision within its lexical signal set (numeric ratio >1.5×, 32-entry antonym dictionary, explicit-negation detector). Per-signal recall: numeric 0.579, negation 0.333, antonym 0.114. MCD operates within known constraints: it achieves high precision (0.943) within its lexical-signal set but recall is 0.330, and semantic contradictions without lexical markers—version-bump (0/10 detected) and team-change (0/5 detected)—fall outside its operating range. MCD is designed as a high-precision escalation trigger, not a general conflict-detection solution; embedding-based detection is identified as future work.

C. System Calibration and Ablation (20-Query Pilot)

The 20-query pilot serves two purposes: (i) threshold calibration ($\tau_1 = 0.45$, $\tau_2 = 0.30$); (ii) end-to-end system demonstration with real API spend. It is not a routing-policy evaluation—that role belongs to the 1.2M benchmark in Section V-A. The pilot spans six categories:

TABLE III
Routing metrics on the five-corpus real-user-query benchmark (N = 1,197,316, all exact-string unique). POA = Policy-Oracle Agreement against the category-level oracle. Per-query cost is a tier-cost projection applied to routing decisions, not measured API spend; measured spend is reported only for the 20-query calibration pilot. Aggregate POA is inflated by the dominance of factoid-lookup corpora (MS MARCO+NQ+MMLU = 77% of rows, unconditionally Tier-1 by oracle construction); the English-only and non-trivial subsets (Table IV) give the honest operating range.

Policy	POA (95% Wilson CI)	OSR	USR	$ / query (proj.)
Lightify CDDR	0.913 [0.912, 0.913]	0.000	0.087	$0.0025
Lightify CDDR + action overlay	0.404 [0.403, 0.405]	0.593	0.003	$0.0154
Naïve Opus-only baseline	0.000 [0.000, 0.000]	1.000	0.000	$0.0760
Hand-coded keyword router (LangGraph-style)	0.794 [0.793, 0.795]	0.012	0.194	$0.0007
Hand-coded length-gateway router (LangGraph-style)	0.324 [0.323, 0.325]	0.621	0.055	$0.0196

TABLE IV
Per-source and per-language CDDR POA on the five-corpus benchmark. The non-trivial subset (WildChat English + GSM8K) excludes all corpora with unconditional Tier-1 oracle labels and represents the honest stress-test operating range. Language split by ASCII-strict detection.

Source / subset	N	CDDR POA
MS MARCO v2.1 train	808,465	1.000
WildChat-1M (all)	271,661	0.625
Natural Questions	99,903	1.000
MMLU auxiliary_train	10,287	1.000
GSM8K main/train	7,000	0.607
Non-trivial subset (WildChat-Eng + GSM8K)	125,390	0.648 [0.646, 0.651]
English subset (ASCII-strict)	1,043,220	0.958 [0.957, 0.958]
Non-English subset	154,096	0.604
Aggregate (all rows)	1,197,316	0.913

easy factual (5), code generation (3), architecture (3), multi-hop reasoning (3), conflict resolution (3), and cold knowledge (3). The memory store is seeded with 17 items including 3 planted contradictions.

D. Single-Query Comparison (Local-First)

Table V shows a head-to-head comparison for the query "What is the Python GIL?"

Lightify incurs zero API cost on this local-routable scenario. The latency numbers in Table V are single-query data points from a case in which Tier-1 routing succeeds; we do not draw a distributional latency claim from them. The latency advantage on this row reflects both routing to a local model and the smaller context created by CDPS compression; the routing decision itself adds <50 ms of overhead.

E. 20-Query Grid Benchmark (Preliminary)

We conduct a preliminary evaluation on 20 queries across six categories—easy factual (5), code generation (3), architecture (3), multi-hop reasoning (3), conflict resolution (3), and cold knowledge (3)—against six routing strategies. The memory store is seeded with 17 items including 3 planted contradictions. We note that this benchmark is limited in scale; we present it to demonstrate the routing mechanism and conflict detection behavior rather than to establish statistical claims. The methodology for the full-scale evaluation ($N \geq 500$) is described in Section V-K. This 20-query benchmark does not meet the statistical rigor described in Section V-K (no bootstrap confidence intervals, no significance testing); formal inference is deferred to the full-scale study. Quality in Table VI is computed as token-level F1 between the set of context keywords and the response keywords (after lowercasing and stop-word removal), scaled to 0–100; this definition is used only for the preliminary table and is superseded by the LLM-as-judge rubric in Section V-K. Table VI reports category-averaged results.

Routing behavior. On all 5 easy factual queries with stable knowledge, Lightify correctly routed to Tier-1 (local, $0). MCD detected all 3 seeded contradictions and escalated each to Tier-2 or Tier-3. The full MCD precision/recall characterization is in Section V-B. The selective escalation behavior—low-cost when context is reliable, high-capability when conflicts are present—illustrates the operating characteristic the Lightify middleware provides to any harness.

Quality evaluation deferred. A keyword-overlap quality metric (reported in earlier versions of this work) systematically rewarded verbosity and disadvantaged CDPS-shaped concise answers; we determined it was not calibrated for conciseness-shaped outputs and removed it from Table VI. Substantive quality judgment is deferred to the LLM-as-judge evaluation described in Section V-K, with an $N \geq 500$ plan tracking exact-match correctness on closed-form benchmarks (MMLU-Pro, HumanEval+) and GPT-4o-judge scores on open-ended queries. Until that evaluation is executed, the cost and latency numbers in Table VI should be interpreted as operating characteristics rather than quality-adjusted efficiency claims.

TABLE V
Single-query comparison: same question, five approaches. Illustrative single-query decomposition only; the latency and token numbers are one-query data points, not distributional claims. Baselines in this table do not cascade on Tier-1 failure whereas Lightify does (see §V opening paragraph), so latency comparisons across rows are not apples-to-apples.

Approach	Cost	Latency	Tok In	Tok Out
Raw Opus	$0.076	5,425 ms	27,038	272
Always Sonnet	$0.078	7,449 ms	19,382	344
Always Haiku	$0.017	4,275 ms	44,015	346
Local Gemma	$0.000	6,741 ms	15	1,004
Lightify	$0.000	339 ms	214	22

TABLE VI
Preliminary 20-query grid benchmark: average cost and latency per routing strategy. Keyword-match quality scores are intentionally omitted from this table because the metric systematically favors verbose outputs, disadvantaging CDPS-shaped responses; an LLM-as-judge evaluation is described in Section V-K.

Approach	Avg Cost	Avg Latency
Local Gemma (no routing)	$0.000	7,320 ms
Always Haiku	$0.023	7,113 ms
Always Sonnet	$0.041	12,060 ms
Always Opus	$0.087	11,162 ms
Lightify	$0.037	8,019 ms
LF --fast	$0.028	8,028 ms

F. Policy-Oracle Agreement at N=5,000 (Synthetic)

To isolate the effect of routing policy from live model-call noise and retrieval noise, we generate 5,000 synthetic queries with gold oracle labels across seven categories (40% short-operation: 20% bash-like and 20% short-lookup; 30% mid-operation: 15% reasoning and 15% code; 30% frontier-operation: 10% large-code, 10% conflict, 10% cold-knowledge (queries whose answer is absent from the memory store, requiring frontier-tier capability)). All 5,000 queries are unique exact-string instances produced from 48 templates with 3–24 fillers per slot (combinatorial capacity $\approx$ 80,000, seed = 42). Category-calibrated context confidence is injected per query (matching the 20-query live pilot distribution), and conflict queries receive an MCD-visible contradiction. We define Policy-Oracle Agreement (POA) as the fraction of queries whose routing decision matches the category-level oracle tier. POA measures agreement with a heuristic oracle, not correctness against live tier outputs; we reserve the term "routing accuracy" for the per-query live-oracle evaluation in Section V-K. We report POA, Over-Spend Rate (OSR), and Under-Spend Rate (USR) against the category oracle together with 95% bootstrap confidence intervals (1000 resamples).

The CDDR primitive alone reaches 70.2% POA at 79% lower per-query cost than the Naïve Opus-only baseline. Adding the per-action overlay is approximately net-neutral at the aggregate level (+0.3 percentage points POA, +$0.0013 per query at N = 5,000), but produces large and heterogeneous per-category effects. Table VIII breaks this down.

The overlay's contribution is substantially additive on code (+42.0 pp) and reasoning (+38.1 pp)—categories where CDDR's retrieval-confidence signal is mid-range and the action signal supplies disambiguating information. The overlay's contribution is bounded on short_lookup (−34.9 pp) and cold_knowledge (−35.8 pp)—categories where the shared "what is …" surface form spans both the Tier-1 oracle ("what is TLS?") and the Tier-3 oracle ("what is the capital of Tuvalu?"), which the lexical classifier does not distinguish. The conflict-aware override in the present implementation escalates to Tier-2 under retrieval weakness rather than Tier-3, so conflict queries remain at Tier-2 regardless of overlay state. Three extensions follow directly:

1) Per-category dispatch. Route through the overlay only for categories where its effect is positive (code, reasoning, large_code). This is a one-flag change in the combiner.
2) Embedding-based action classifier. Replace the regex cascade with a small embedding model (< 50M params) trained to distinguish lookup from cold-knowledge—recovering 30+ pp on two categories.
3) Stronger conflict escalation. Change the MCD override so detected conflicts escalate to Tier-3 when retrieval confidence is below τ_2, recovering all 500 conflict-category queries.

Scale stability is verified at three dataset sizes (Table IX); the CDDR point estimate moves by less than one percentage point between $N = 2{,}000$ and $N = 5{,}000$, and the 95% CI narrows from ±6.0 percentage points at $N = 200$ to ±1.3 percentage points at $N = 5{,}000$. Oracle labels remain category-level heuristics; a per-query live oracle (Section V-K) is future work.

G. Structural Comparison with Hand-Coded LangGraph-style Baselines (N=5,000)

LangGraph ships no cost-aware routing primitive; routing logic is written by the user as graph edges or as an

TABLE VII
Routing metrics on 5,000 synthetic queries (100% unique) with gold oracle labels. POA is Policy-Oracle Agreement: the fraction matching the category-level oracle tier; OSR / USR are the over-spend and under-spend rates. Confidence intervals are 95% bootstrap CIs (1000 resamples), distinct from the Wilson CIs used for the real-benchmark tables. Per-query cost uses a fixed tier-cost projection calibrated from the 20-query live pilot (Tier-1 = $0, Tier-2 = $0.019, Tier-3 = $0.076) applied to routing decisions, not measured API spend.

Policy	POA (95% bootstrap CI)	OSR	USR	$ / query (proj.)
CDDR only	0.702 [0.689, 0.715]	0.000	0.298	$0.0161
CDDR + action overlay	0.705 [0.692, 0.716]	0.084	0.211	$0.0174
Naïve Opus-only baseline	0.300 [0.287, 0.313]	0.700	0.000	$0.0760

TABLE VIII
Per-category effect of the per-action overlay at $N = 5{,}000$. ΔPOA is the change in Policy-Oracle Agreement when the overlay is enabled. Positive deltas indicate categories where the overlay recovers under-spend cases; negative deltas indicate over-promotion by the keyword-style classifier.

Category	N	CDDR POA	+Action POA	ΔPOA
bash_like	1000	1.000	0.929	−0.071
short_lookup	1000	1.000	0.651	−0.349
reasoning	750	0.607	0.988	+0.381
code	750	0.580	1.000	+0.420
large_code	500	0.236	0.264	+0.028
conflict	500	0.000	0.000	+0.000
cold_knowledge	500	1.000	0.642	−0.358

TABLE IX
Scale stability of the routing benchmark at three dataset sizes. All datasets are 100% exact-string unique. POA = Policy-Oracle Agreement.

Policy (POA)	$N = 200$	$N = 2{,}000$	$N = 5{,}000$
CDDR only	0.710 [0.650, 0.770]	0.695 [0.674, 0.714]	0.702 [0.689, 0.715]
CDDR + action overlay	0.670 [0.605, 0.735]	0.697 [0.676, 0.718]	0.705 [0.692, 0.716]
Hand-coded keyword router (LangGraph-style)	0.505	0.524	0.516
Hand-coded length-gateway router (LangGraph-style)	0.565	0.529	0.484
Naïve Opus-only baseline	0.300 [0.240, 0.365]	0.300 [0.280, 0.321]	0.300 [0.287, 0.313]

external gateway. We implemented three policies that a LangGraph user might plausibly hand-code using LangGraph primitives: (i) a Naïve Opus-only baseline (single frontier model, the Quickstart default); (ii) a hand-coded keyword router (code-regex → Sonnet, else Haiku); (iii) an external hand-coded length-gateway router. These baselines were implemented by the authors using LangGraph primitives; they are not official LangChain Inc. artifacts and do not represent the best routing achievable inside LangGraph. Results on the same 5,000-query set are in Table X.

Lightify's CDDR primitive alone achieves 18.6–21.8 percentage points higher POA than the best hand-coded LangGraph-style user baseline, at a per-query cost comparable to the best such baseline and 79% below the Opus-only default. The cheapest hand-coded baseline (keyword router) is 83% cheaper than Lightify CDDR but under-routes 48% of queries, producing tier mismatches on roughly half the benchmark. The length-gateway variant incurs the highest cost of the three hand-coded baselines while still trailing CDDR by 22 percentage points, illustrating the gap between surface-feature proxies and retrieval-confidence as a routing signal. This measures what a LangGraph user can achieve without writing a retrieval-confidence routing primitive, not a claim about LangGraph's expressive ceiling; Lightify's contribution is precisely that it ships this primitive as a runtime service. The margin relative to a learned router (e.g., RouteLLM [8]) is not measured in this paper; whether CDDR's retrieval-confidence signal yields higher or lower POA than a preference-trained classifier on the same oracle is an open empirical question identified as future work.

H. Benchmark Dataset Documentation

The synthetic benchmark used in Sections V-F–V-G is released alongside the source code at https://github.com/pavanmanikanta31/lightify under MIT license. Each row is a JSON object with fields id, category, query, oracle_tier, has_contradiction; files are deterministic under random.seed(42). Table XI documents the composition and generation.

Generation. Queries are produced by random template-filler substitution (48 templates, 3–24 fillers per slot; seeded at 42 for reproducibility). The same template set is used at every scale; scale is controlled solely by the number of samples per category. gen() rejects exact-string duplicates with a retry budget of $20N$; all three released datasets are 100% exact-string unique at $N = 200$, $N = 2{,}000$, and $N = 5{,}000$ (combinatorial capacity of

TABLE X
Structural comparison at $N = 5{,}000$ (100% unique) against three hand-coded LangGraph-style user-authored baselines. Lightify ships the router as a runtime primitive; LangGraph requires each policy to be written as an application-layer graph edge. POA = Policy-Oracle Agreement.

Policy	POA	OSR	USR	$ / query
Naïve Opus-only baseline	0.300	0.700	0.000	$0.0760
Hand-coded keyword router (LangGraph-style)	0.516	0.003	0.480	$0.0028
Hand-coded length-gateway router (LangGraph-style)	0.484	0.227	0.288	$0.0164
Lightify (CDDR)	0.702	0.000	0.298	$0.0161
Lightify (CDDR + action overlay)	0.705	0.084	0.211	$0.0174

TABLE XI
Synthetic benchmark composition. Category proportions are identical across scales; the $N = 5{,}000$ file is generated by benches/generate_200.py –scale 25.

Category	% of set	Oracle	MCD	Templates	Example
bash_like	20%	Tier-1		14	"git status HEAD"
short_lookup	20%	Tier-1		7	"what is TLS?"
reasoning	15%	Tier-2		7	"compare CDDR and RouteLLM"
code	15%	Tier-2		5	"write a function that reverses a string"
large_code	10%	Tier-3		5	"refactor the entire authentication module"
conflict	10%	Tier-3	yes	5	"is v7 production or staging?"
cold_knowledge	10%	Tier-3		5	"what is the capital of Tuvalu?"
Total	100%			48	

the template × filler space is approximately 80,000, so reserving 5,000 unique samples leaves 75,000 of headroom for scale-up).

Oracle labeling. Each category is assigned a single gold oracle tier under the assumption that category membership dominates tier requirement (e.g., bash-like queries are trivially answered by a local tier; cold-knowledge queries cannot be answered without frontier capability). This is a strong-category prior; a per-query live-oracle construction (running each tier against each query and scoring correctness) is described in Section V-K and reserved as future work. The synthetic oracle is sufficient for isolating routing behavior from retrieval and generation quality, which is the variable the tables in this section seek to control.

Context confidence injection. For each query, a context confidence $\phi \in [0, 1]$ is drawn from a category-specific uniform distribution calibrated from the 20-query live pilot: bash/lookup $\phi \sim U(0.50, 0.85)$; reasoning/code $\phi \sim U(0.30, 0.55)$; large-code $\phi \sim U(0.25, 0.45)$; conflict $\phi \sim U(0.30, 0.55)$; cold-knowledge $\phi \sim U(0.02, 0.18)$. Coverage is derived from ϕ with ± 0.1 uniform noise. Conflict queries inject a synthetic contradiction into the ContextCapsule so that MCD fires. This stand-in for retrieval is what permits $N = 5{,}000$ without running 5,000 live retrievals; the live pilot in Table V confirms that the ϕ distributions match what the real retrieval system produces on the category's characteristic queries.

Limits of synthetic benchmarking. We flag three limitations: (i) the single-oracle-per-category prior is coarser than a live-oracle; (ii) context confidence is injected rather than produced by retrieval, so the benchmark does not exercise the retrieval stack; (iii) POA does not measure output quality (correctness against a live tier oracle is future work). The primary routing-policy evidence is the 1.2M real-user-query benchmark in Section V-A; the 20-query live pilot supplies the only measured API-spend numbers (Tables V, VI, XII, XIII); the $N = 5{,}000$ synthetic benchmark provides statistical power on the routing decision in isolation.

I. Per-Mechanism Ablation (C5)

Table XII reports a leave-one-out ablation on the 20-query pilot: each run removes one Lightify mechanism while holding the other four fixed. The aim is to expose the marginal contribution of each mechanism to Tier-1 routing rate and per-query cost; all numbers are from the same 20-query set and inherit the pilot's statistical limitations. Removing CDDR forces all queries to Tier-3 (cost-blind cascade); removing MCD routes contradictions through Tier-1 without conflict-aware escalation; removing CDPS eliminates the local-tier speedup; removing CSE permits under-retrieved queries at Tier-1; removing SECR is cost-neutral on the pilot and is included for completeness.

TABLE XII
Per-mechanism leave-one-out ablation on the 20-query pilot. "Safety" is the rate at which MCD-planted contradictions escalate above Tier-1. "T1 rate" is the fraction of queries routed to the local tier. Safety = 0.00 for the −MCD row is the expected null case: no contradictions are detected when MCD is disabled.

Configuration	T1 rate	Avg cost	Avg lat (ms)	Safety
Full Lightify	0.55	$0.037	8,019	1.00
– CDDR (force T3)	0.00	$0.087	11,162	1.00
– MCD (no conflicts)	0.70	$0.028	7,400	0.00
– CDPS (verbose)	0.55	$0.041	9,800	1.00
– CSE (no coverage)	0.65	$0.033	8,100	1.00
– SECR (no shorthand)	0.55	$0.037	8,050	1.00

TABLE XIII
Fair-baseline ablation (C4). All rows use identical retrieval and CDPS-shaped prompts; only the tier-selection policy differs. Numbers are per-query averages on the 20-query pilot.

Policy (with retrieval + CDPS)	Avg cost	Avg lat (ms)
Tier-forced Haiku	$0.019	6,800
Tier-forced Sonnet	$0.034	10,900
Tier-forced Opus	$0.078	10,400
Lightify (CDDR + MCD)	$0.037	8,019

J. Fair-Baseline Ablation (C4)

A frequent reviewer concern is that baselines in Table V and VI do not receive CDPS-shaped prompts or conflict-aware cascades, making any headline comparison unfair. Table XIII reports a controlled ablation in which each baseline is given identical retrieval output and CDPS-shaped prompts; only the tier-selection policy changes. The table isolates the contribution of Lightify's routing policy from the contribution of its retrieval and prompting. Note that Lightify ($0.037) costs more than Tier-forced Haiku ($0.019) on the 20-query pilot: this is expected, as the pilot includes conflict and cold-knowledge queries that trigger MCD-driven escalation to Tier-2 or Tier-3; a Haiku-only policy would handle those queries incorrectly at lower cost.

K. Evaluation Methodology for Full Study

The following methodology, adapted from the evaluation protocols of FrugalGPT [7] and RouteLLM [8], will be used for the full-scale evaluation:

Oracle construction. For each query q_i, run through all three tiers independently. The cheapest tier producing a correct answer is labeled t_i^*. Correctness is defined dual-track: for closed-form queries, exact match against the gold answer; for open-ended queries, a GPT-4o judge scores the response under the rubric below, and a response is deemed correct when its Correctness sub-score (weight 0.50) exceeds 0.75 on a 0–1 scale. This binary correctness flag is computed before any routing metric.

Routing metrics. Routing Accuracy (RA): fraction matching the cheapest correct tier (live per-query oracle; to be used only here, not in the synthetic or 1.2M benchmarks of this paper, which report POA instead). Over-Spend Rate (OSR): fraction routed to a more expensive tier than necessary. Under-Spend Rate (USR): fraction routed too cheaply, producing incorrect answers.

Quality scoring. Dual-track: exact match for closed-form benchmarks (MMLU-Pro [30], HumanEval+ [31]); LLM-as-judge (GPT-4o, following MT-Bench protocol [29]) with rubric $Q = 0.50 \times \text{Correctness} + 0.30 \times \text{Completeness} + 0.20 \times \text{Conciseness}$ for open-ended queries.

Statistical rigor. $N \geq 500$ queries, 95% bootstrap confidence intervals, McNemar's test for accuracy comparisons, Wilcoxon signed-rank for cost, Holm-Bonferroni correction.

VI. Discussion

A. When Lightify Adds Value over a Bare Harness

The benefit of Lightify as a governance middleware increases with: (1) knowledge-base size, since more memory items create more potential conflicts; (2) update frequency, since frequent changes amplify temporal drift; (3) domain specificity, since internal tools, versioned APIs, and policy documents evolve faster than general knowledge; and (4) query volume, since the cost differential between local and frontier inference compounds with scale. A harness without Lightify-like governance must hand-code each of these concerns at the application layer.

B. Comparison with Query-Aware Routers

One may argue that a single mid-tier model (e.g., "always Sonnet") achieves comparable cost savings with less complexity. Our benchmark partially supports this observation, as Sonnet handled most queries adequately. However, three distinctions remain: (1) a single-tier strategy cannot detect knowledge conflicts and will serve contradictory answers without escalation; (2) it provides no path to zero-cost inference, whereas Lightify's local tier does; and (3) CDPS produces more focused answers on the 20-query pilot (see the Tok Out column of Table V, where Lightify emits 22 tokens against 272–1,004 for the single-tier baselines), reducing downstream processing cost.

The stronger algorithmic comparison is with AutoMix [9]. AutoMix self-verifies per query but cannot detect drift across queries, nor does it expose its routing choices at the runtime layer of an agent harness. For static knowledge bases, AutoMix suffices. For evolving knowledge bases served by an agent harness—the scenario this paper targets—retrieval-confidence routing and conflict

detection provide signals that AutoMix fundamentally lacks.

VII. Limitations

1) Oracle scope: POA vs. per-query live correctness. The primary routing-policy result is the 1.2M real-user-query benchmark, but it is scored as Policy-Oracle Agreement against a category-level oracle, not as routing correctness against per-query live-tier outputs. A live per-query oracle—running each tier against each query and scoring correctness via the rubric in Section V-K—would replace POA with true routing accuracy on a subset such as MMLU+GSM8K with gold answers; this subset evaluation is the next concrete empirical step. The 20-query live pilot is the only source of measured API spend and is not itself a routing-policy evaluation.
2) Keyword-based quality metric. The current keyword-match metric penalizes conciseness, systematically under-scoring Lightify relative to verbose baselines. An LLM-as-judge evaluation with a correctness-focused rubric is needed to assess true quality.
3) Uncalibrated confidence scores. Both the routing thresholds (τ_1=0.45, τ_2=0.30) and the sigmoid shaping parameters (A, B) are hand-tuned rather than learned. Platt-style calibration on a held-out query set is a prerequisite for production deployment.
4) Keyword-level MCD. The 32-pair antonym dictionary and simple negation detection miss semantic contradictions without lexical overlap. The 200-pair stress evaluation yielded precision 0.943 but recall 0.330; in particular, MCD misses the version-bump (0/10) and team-change (0/5) realistic-deployment slices, which are the most common real-world contradiction shapes. Embedding-based conflict detection is left for future work.
5) No temporal drift benchmark. The claim that retrieval-confidence routing with MCD is safer when knowledge evolves requires a time-evolution evaluation with deliberate knowledge mutations. This is the highest-priority future experiment.
6) Baseline scope. The LangGraph-style baselines in Tables X and III are hand-coded policies written by the authors using LangGraph primitives, not official LangChain artifacts; they measure what a LangGraph user can achieve without writing a retrieval-confidence routing primitive, not LangGraph's expressive ceiling. A learned-router baseline such as RouteLLM is the natural comparison and is identified as future work.

VIII. Security Considerations

Lightify introduces two attack surfaces requiring mitigation:

Memory poisoning via background learning. The usage-update loop can amplify adversarial content. Mitigations include confidence caps on Tier-1-originated content, content-hash deduplication, and automatic pruning of low-confidence items. See OWASP LLM Top 10 [28] for a broader threat taxonomy.

Prompt injection via compressed memory. Compressed memory items in the context block could contain adversarial instructions. Mitigation uses XML-tag structural separation of instructions and data.

IX. Future Work

1) Temporal drift benchmark: Programmatically mutate the knowledge base across simulated time steps and measure whether Lightify escalates on stale answers that Gemma alone serves confidently.
2) Embedding-based retrieval and conflict detection to replace FTS5 keyword matching and the limited antonym dictionary.
3) Learned router trained on historical (q, C, t^*) tuples, replacing the heuristic threshold cascade.
4) Parallel dispatch: Run Tier-1 and Tier-2 simultaneously, return the first adequate response, drawing on speculative decoding [20] to reduce cascade latency.
5) Formal Platt calibration of confidence scores on held-out evaluation sets with reliability diagrams.

X. Conclusion

We presented Lightify, an open-source middleware layer for agent harnesses that provides cost-aware tier routing, local-first inference, memory conflict detection, automatic escalation, and confidence-driven prompt shaping as runtime primitives. A structured survey of six major harnesses (Claude Agent SDK, LangGraph, AutoGen, CrewAI, OpenHands, LlamaIndex Agents) shows these primitives are uniformly left to the application developer.

Empirically, Lightify routes queries with consistent, high-confidence context to a local tier at zero API cost on the illustrative single-query example in Section V-C. On a large-scale real-query benchmark assembled from five public corpora (MS MARCO, WildChat, Natural Questions, MMLU, and GSM8K; N = 1,197,316 exact-string unique queries), CDDR reaches a 95% Wilson CI of $[0.957, 0.958]$ for Policy-Oracle Agreement (POA) on the English-language subset (1,043,220 rows, 87.1% of the benchmark by ASCII-strict detection), at per-query projected cost 98% below a Naïve Opus-only baseline. The 1.2M aggregate POA of $[0.912, 0.913]$ is inflated by the dominance of factoid-lookup corpora (MS MARCO+NQ+MMLU = 77% of rows, unconditionally Tier-1 by oracle construction) and is reported as compositional disclosure, not headline; per-source POA ranges from 1.000 on lookup corpora to 0.625 on WildChat and 0.607 on GSM8K, and the non-English subset (essentially non-ASCII WildChat) sits at 0.604. Per-query cost throughout the 1.2M benchmark is a tier-cost projection applied to routing decisions, not measured API spend. An MCD stress evaluation on 200 hand-authored contradiction and control pairs yields precision 0.943 and recall 0.330, establishing that MCD's operating range is

high-precision within its lexical signal set and does not cover the version-bump (0/10) and team-change (0/5) realistic-deployment slices. The lexical action overlay is net-additive on code and reasoning queries and bounded on lookup/cold-knowledge queries; per-category dispatch and embedding-based action classification are identified as concrete extensions. A full per-query live-oracle evaluation is reserved as future work.

The broader claim of this paper is architectural: governance primitives that every current harness pushes to the user (cost, locality, conflict, escalation) can and should be provided as composable runtime services beneath the agent loop.

Acknowledgment

This research was conducted independently without external funding. The author thanks the anonymous reviewers for their constructive feedback. The author used Claude (Anthropic) as a writing assistance tool during the preparation of this manuscript. Claude was used for drafting and editing portions of the Introduction, Related Work, and Conclusion sections, and for grammar and clarity improvements throughout. All technical content, experiments, results, code, and data are the author's own original work.

Data Availability Statement

All benchmark artifacts, seeded memory items, and evaluation scripts are publicly available at https://github.com/pavanmanikanta31/lightify. The release includes: the 20-query live pilot; the synthetic routing benchmarks at three scales (queries_200.json, queries_2000.json, queries_5000.json); the 200-pair MCD stress set (contradictions_100.json); per-policy routing outputs for the 1.2M real-user-query benchmark (results_real_1197316.json) and the MCD stress test (results_mcd_stress.json); and the reproducibility script (benches/fetch_real_1m.py) that assembles the 1.2M benchmark from public HuggingFace parquet shards. No proprietary datasets were used.

Code Availability

The Lightify source code is released under the MIT License at https://github.com/pavanmanikanta31/lightify.

Declaration of Competing Interests

The author declares no competing interests. This research received no external funding.

References

[1] Anthropic, "Claude Agent SDK (Python) and Managed Agents," software and documentation, 2025. [Online]. Available: https://github.com/anthropics/claude-agent-sdk-python; https://www.anthropic.com/engineering/managed-agents

[2] LangChain, "LangGraph: stateful agent orchestration," software, 2024. [Online]. Available: https://github.com/langchain-ai/langgraph

[3] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," in Proc. ICLR, 2024. [Online]. Available: arXiv:2308.08155

[4] CrewAI Inc., "CrewAI: role-based multi-agent framework," software, 2024. [Online]. Available: https://github.com/crewAIInc/crewAI

[5] X. Wang et al., "OpenHands: An Open Platform for AI Software Developers as Generalist Agents," in Proc. ICLR, 2025. [Online]. Available: arXiv:2407.16741

[6] J. Liu et al., "LlamaIndex: a data framework for LLM applications and agents," software, 2024. [Online]. Available: https://github.com/run-llama/llama_index

[7] L. Chen, M. Zaharia, and J. Zou, "FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance," Trans. Mach. Learn. Res. (TMLR), 2024.

[8] I. Ong, A. Almahairi, V. Wu, W.-L. Chiang, et al., "RouteLLM: Learning to Route LLMs with Preference Data," in Proc. ICLR, 2025. [Online]. Available: arXiv:2406.18665

[9] A. Madaan, P. Aggarwal, A. Anand, et al., "AutoMix: Automatically Mixing Language Models," in Proc. NeurIPS, 2024.

[10] D. Ding, A. Mallick, C. Wang, R. Sim, et al., "Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing," in Proc. ICLR, 2024.

[11] X. Li et al., "T-GRAG: A Dynamic GraphRAG Framework for Resolving Temporal Conflicts," in Proc. ACM MM, 2025. [Online]. Available: arXiv:2508.01680

[12] Y. Xu et al., "Knowledge Conflicts for LLMs: A Survey," in Proc. EMNLP, 2024.

[13] J. Zhang, R. Krishna, A. H. Awadallah, and C. Wang, "EcoAssistant: Using LLM Assistant More Affordably and Accurately," in Proc. ICLR, 2024. [Online]. Available: arXiv:2310.03046

[14] C. Packer, S. Wooders, K. Lin, et al., "MemGPT: Towards LLMs as Operating Systems," in Proc. ICLR, 2024.

[15] S.-Q. Yan, J.-C. Gu, Y. Zhu, and Z.-H. Ling, "Corrective Retrieval Augmented Generation," in Proc. EMNLP, 2024. [Online]. Available: arXiv:2401.15884

[16] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection," in Proc. ICLR, 2024.

[17] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models," in Proc. EMNLP, 2023. [Online]. Available: arXiv:2310.05736

[18] P. Lewis, E. Perez, A. Piktus, et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Proc. NeurIPS, 2020.

[19] S. Wang et al., "A Survey on Collaborating Small and Large Language Models," arXiv:2510.13890, 2025.

[20] Y. Leviathan, M. Kalman, and Y. Matias, "Fast Inference from Transformers via Speculative Decoding," in Proc. ICML, 2023.

[21] P. Bajaj et al., "MS MARCO: A Human Generated MAchine Reading COmprehension Dataset," arXiv:1611.09268, 2016–2018.

[22] W. Zhao, X. Ren, J. Hessel, et al., "WildChat: 1M ChatGPT Interaction Logs in the Wild," in Proc. ICLR, 2024.

[23] T. Kwiatkowski et al., "Natural Questions: A Benchmark for Question Answering Research," Trans. ACL, vol. 7, pp. 453–466, 2019.

[24] D. Hendrycks et al., "Measuring Massive Multitask Language Understanding," in Proc. ICLR, 2021.

[25] K. Cobbe et al., "Training Verifiers to Solve Math Word Problems," arXiv:2110.14168, 2021.

[26] W. Kwon, Z. Li, S. Zhuang, et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention," in Proc. SOSP, 2023.

[27] A. Vaswani, N. Shazeer, N. Parmar, et al., "Attention Is All You Need," in Proc. NeurIPS, 2017.

[28] OWASP, "OWASP Top 10 for Large Language Model Applications (2025)," 2025. [Online]. Available: https://owasp.org/www-project-top-10-for-large-language-model-applications/

[29] L. Zheng et al., "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," in Proc. NeurIPS, 2023.

[30] Y. Wang et al., "MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark," arXiv:2406.01574, 2024.

[31] J. Liu et al., "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation," in Proc. NeurIPS, 2023.
[32] W. McGugan, "Rich (v13.7+): Python library for rich text and beautiful formatting in the terminal," software, 2024. [Online]. Available: https://github.com/Textualize/rich
[33] Y. Gao, Y. Xiong, X. Gao, et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv:2312.10997, 2023.
[34] Y. Zhu, X. Wang, J. Chen, et al., "Large Language Models for Information Retrieval: A Survey," arXiv:2308.07107, 2023.
[35] N. Kandpal, H. Deng, A. Roberts, E. Wallace, and C. Raffel, "Large Language Models Struggle to Learn Long-Tail Knowledge," in Proc. ICML, 2023.
[36] Y. Su et al., "DRAGIN: Dynamic Retrieval Augmented Generation Based on the Real-Time Information Needs of Large Language Models," in Proc. ACL, 2024.
[37] X. Yue, Y. Ni, K. Zhang, T. Zheng, et al., "Inference Scaling for Long-Context Retrieval Augmented Generation," arXiv:2410.04343, 2024.
[38] L. Chen, M. Zaharia, and J. Zou, "Are More LLM Calls All You Need? Towards Scaling Laws of Compound Inference Systems," in Proc. NeurIPS, 2024.
[39] M. Šakota, M. Peyrard, and R. West, "Fly-Swat or Cannon? Cost-Effective Language Model Choice via Meta-Modeling," in Proc. WSDM, 2024.
[40] T. Shnitzer et al., "Large Language Model Routing with Benchmark Datasets," arXiv:2309.15789, 2024.
[41] J. Jiang et al., "Mixture-of-Agents Enhances Large Language Model Capabilities," in Proc. ICLR, 2025.
[42] Q. J. Hu et al., "RouterBench: A Benchmark for Multi-LLM Routing System," arXiv:2403.12031, 2024.
[43] B. J. Chan et al., "Don't Do RAG: When Cache-Augmented Generation is All You Need for Knowledge Tasks," arXiv:2412.15605, 2024.