

Article

Feedback-Driven SQL Optimization with Validated Query Rewrites

Martin Kostov *  and Kalinka Kaloyanova 

Faculty of Mathematics and Informatics, Sofia University “St. Kliment Ohridski”, 1164 Sofia, Bulgaria;
kkaloyanova@fmi.uni-sofia.bg

* Correspondence: martinkk@fmi.uni-sofia.bg

Abstract

Cost-based query optimizers are essential for relational database systems, but SQL formulation can still affect selected execution plans and runtime, especially in recurring analytical workloads and machine-generated queries. This paper proposes a feedback-driven lifecycle for SQL optimization, based on empirically validated query rewrites. The contribution of the presented approach is a persistent candidate-management process that covers the path from candidate intake to provenance recording, structural admissibility checks, empirical result-equivalence validation, paired runtime evidence, guarded activation, retention, rejection, and later deactivation. Candidate rewrites may come from deterministic rules, local large language models, manual alternatives, or external rewrite systems; the source is recorded but does not determine acceptance. The evaluation uses a controlled research implementation with deterministic-rule cases, repeated TPC-H SF1 runs, a real-world-style anti-pattern corpus, and JOB/IMDB. The results show conservative behavior on mature analytical templates, including mostly withheld TPC-H candidates with one held-out positive case, stable evidence for selected anti-patterns, and comparable but non-identical JOB/IMDB positives across runs. The findings support a source-neutral lifecycle in which alternative SQL formulations are admitted, measured, retained, activated, or rejected according to accumulated evidence rather than the generating mechanism.

Keywords: relational database systems; SQL optimization; SQL query rewriting; LLM-assisted query rewriting; query rewrite validation; execution feedback; semantic validation; paired benchmarking; rewrite candidates; candidate lifecycle



Academic Editors: Enrique Moguel and Sergio Laso

Received: 4 June 2026

Revised: 30 June 2026

Accepted: 1 July 2026

Published: 3 July 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Relational database management systems (DBMSs) remain a core infrastructure for operational data access, analytical reporting, extract-transform-load (ETL) workflows, and business intelligence (BI) applications. Structured Query Language (SQL) is the main declarative interface for these systems, and modern DBMSs rely on cost-based query optimizers to transform declarative statements into executable plans. The relational model provides the conceptual basis for these systems [1]. Early work on access-path selection introduced cost-based optimization, establishing principles still used in modern systems. Subsequent surveys expanded the understanding of physical operators, execution strategies, and large-scale query evaluation. System-level analyses highlighted practical challenges such as statistics management, robustness, and plan selection. Research on estimation theory emphasized the difficulty and importance of accurate cardinality prediction.

This dependence is particularly relevant for recurring workloads. BI dashboards, reporting jobs, object-relational mapping (ORM) layers, ETL pipelines, and data-transformation workflows often produce stable query templates whose parameter values change across executions. For such templates, the cost of searching for a better SQL formulation can be amortized over later executions. However, a plausible rewrite may be semantically invalid, unsupported by the target engine, unstable across parameter values, or slower than the original query. Automatic SQL rewriting is therefore useful only when candidate generation is separated from independent validation, measurement, and guarded activation.

1.1. Related Work and Research Gap

Classical optimizers rely on algebraic equivalences, access-path selection, decorrelation, subquery unnesting, predicate movement, projection pruning, and join reordering [2]. In [3], Graefe surveys query-evaluation techniques and physical operators for large databases. Chaudhuri summarizes relational query-optimization challenges at the system level [4]. Ioannidis discusses optimizer principles and estimation issues that motivate careful selection of plans [5]. Rule-based optimizer frameworks, such as the Volcano optimizer generator, demonstrate extensible and modular search architectures [6]. Modern systems like Apache Calcite provide an open-source example of modular query processing over heterogeneous data sources [7].

These mechanisms are powerful, but they are limited by the encoded rules, the supported SQL fragment, and the difficulty of preserving SQL semantics under NULL values, duplicate rows, aggregation, ordering, and dialect-specific behavior.

Recent systems expand the rewrite space in several ways. WeTune discovers and verifies query rewrite rules [8]. LearnedRewrite treats rewriting as a learned search process over rule sequences [9]. SlabCity uses program synthesis to search directly in SQL query space [10]. QueryBooster provides middleware-based rewriting for settings where the application and DBMS internals are black boxes [11]. These systems show that useful rewrites can exist outside the DBMS optimizer and that rule correctness depends on explicit preconditions rather than surface-level similarity.

Large language model (LLM) based systems further enlarge the candidate-generation space. GenRewrite uses natural-language rewrite rules and correction loops [12]. LITHE studies database-sensitive LLM prompting, semantic checks, and filters for brittle rewrites [13]. QUITE uses a feedback-aware multi-agent workflow with database tool use and hint injection [14]. R-Bot retrieves rewrite evidence and guides rule selection through step-by-step LLM reasoning [15]. E3-Rewrite frames rewriting around executability, equivalence, and efficiency through execution-guided learning [16]. LEO demonstrates learning from optimizer feedback inside DB2 [17]. Bao later shows practical learned steering of query-plan choices [18]. These adaptive systems demonstrate the value of execution feedback, but they operate mainly at the optimizer-estimate or plan-selection level, not at the SQL-formulation lifecycle level.

This paper, therefore, does not claim that LLM-based SQL rewriting, feedback-aware rewriting, or automatic rule discovery is new. Its focus is on the lifecycle after a candidate formulation has been generated. A rewrite candidate, regardless of source, is treated as a hypothesis. It must be recorded, structurally checked, empirically validated, benchmarked under paired conditions, and then activated, retained, or rejected. Deterministic rules and LLMs are candidate sources, not semantic authorities.

1.2. Scope and Positioning of the Paper

Against this background, the paper focuses on the evidence lifecycle that follows candidate generation rather than proposing another SQL rewrite generator. The proposed

approach separates the roles of candidate source, structural admissibility check, empirical result-equivalence validation, paired runtime measurement, provenance recording, and guarded activation. This separation allows deterministic rules, local LLM proposals, manual alternatives, learned rewrite systems, synthesis systems, or middleware-based tools to be treated as upstream sources whose outputs must pass the same downstream evidence path.

The methodological part of the paper formalizes recurring query templates, parameterized executions, candidate pools, observable SQL results, and empirical result-equivalence conditions under SQL bag semantics. The empirical part evaluates the lifecycle in a controlled execution setting over deterministic rule validation cases, TPC-H SF1, a controlled real-world-style anti-pattern corpus, and JOB/IMDB. The intended contribution is therefore a source-neutral, evidence-driven candidate-management process for recurring SQL templates, with an evaluation that reports positive, neutral, negative, and withheld outcomes rather than claiming broad benchmark-wide speedup.

The specific contributions of the research consist of: (1) a source-neutral evidence lifecycle that treats any rewrite candidate (deterministic-rule, local-LLM, manual, or external) as a hypothesis evaluated by the same downstream evidence path of structural checks, empirical result-equivalence validation, paired benchmarking, provenance recording, and guarded activation; (2) a persistent candidate pool that retains validation state, paired-measurement history, and decision state across repeated runs, enabling retention, activation, and deactivation on later regression; (3) conservative activation criteria combined with an independent held-out confirmation stage, so that activation is withheld for neutral, unstable, unsupported, or regressive alternatives; and (4) a controlled computational study over deterministic-rule cases, TPC-H SF1, a controlled real-world-style corpus, and JOB/IMDB that reports positive, neutral, negative, and withheld outcomes rather than a single speedup number. The remainder of the paper presents the method and protocol (Section 2), the empirical evidence (Section 3), and the interpretation, related-system positioning, and limitations (Section 4).

2. Materials and Methods

This section presents the proposed feedback-driven method and the controlled experimental protocol used to evaluate it. The focus is on the method, validation procedure, paired measurement protocol, and empirical evidence. Software and environment details are included only where necessary for reproducibility.

2.1. Feedback-Driven Lifecycle

The method treats each alternative SQL formulation as a candidate hypothesis, not as an immediately trusted optimization. A recurring SQL template first enters candidate generation. Candidate SQL formulations may come from deterministic rules, a local LLM, manual input, or an external rewrite system. The source is recorded as provenance, but it does not determine acceptance.

Each alternative form must pass structural checks, empirical result-equivalence validation, paired runtime benchmarking, evidence recording, and a conservative decision stage. The possible outcomes are activation, retention for further evidence, rejection, or later deactivation when new measurements indicate regression. Figure 1 summarizes the lifecycle flow.

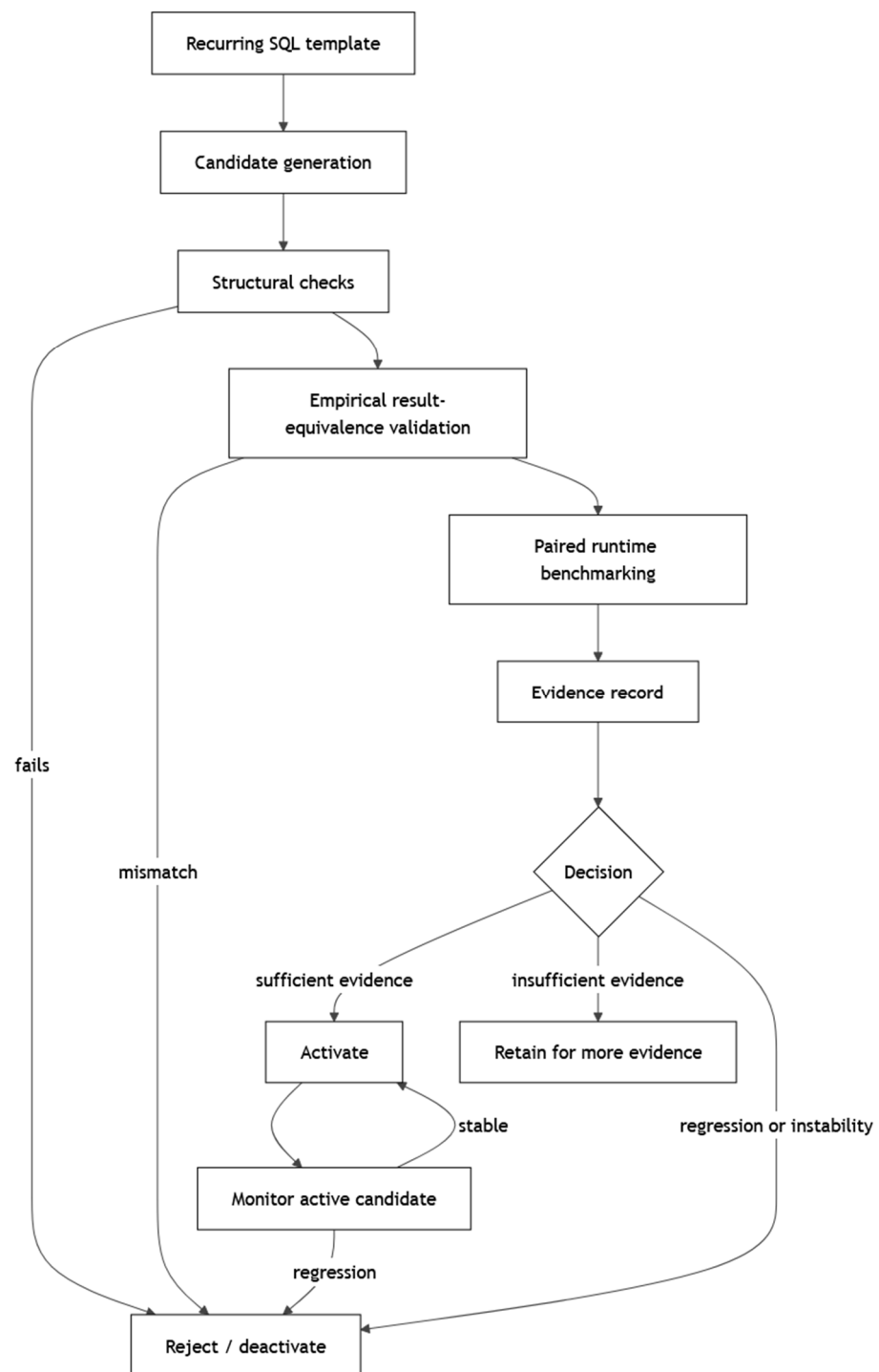


Figure 1. Feedback-driven lifecycle for validated SQL rewrite candidates.

The lifecycle states and transition labels in Figure 1 are defined as follows. *Candidate generation* records an alternative SQL formulation with its source and provenance. *Structural checks* verify that the candidate stays inside the supported SQL fragment, preserves parameter positions and the observable output shape, and introduces no unsafe constructs. *Empirical result-equivalence validation* compares baseline and candidate results on the configured parameter set. *Paired runtime benchmarking* measures the two formulations under the AB/BA protocol. The *Activate* state routes future executions to the candidate while preserving the baseline as a fallback. The *Retain for more evidence* state keeps a validated but not-yet-confirmed candidate in the pool for re-evaluation when more pairs, parameters,

or budget become available. The *Monitor active candidate* state periodically re-measures an activated candidate against the preserved baseline. The *Reject/deactivate* state removes a candidate, or an active candidate that later regresses. The transition labels are *mismatch* (empirical validation found a result difference), *regression* or instability (paired or monitoring evidence is negative or unstable), and *stable* (monitoring evidence continues to support the active candidate).

The lifecycle is intentionally conservative. A false positive activation can repeatedly route future executions to a slower or incorrect formulation. A false negative only misses a possible optimization opportunity. The rejection state also covers deactivation of a previously activated candidate when later evidence indicates regression.

2.2. Query Templates and Candidate Model

Let T denote a recurring SQL query template and let p denote a vector of parameter values. A concrete execution is represented as (T, p) . Let q_T be the baseline SQL formulation for template T , and let c be a candidate rewrite for the same template.

For a database instance D , the observed result of a query is written as:

$$R(q, D, p). \quad (1)$$

The result comparison uses the externally visible SQL result. If the query has an outer ORDER BY, the ordered sequence is compared. Otherwise, the result is compared as a multiset of rows, preserving duplicate counts. A candidate is empirically result-equivalent on the validation parameter set P_V when, for every p in P_V , $R(q_T, D, p)$ and $R(c, D, p)$ are observationally equivalent.

Here, observational equivalence is evaluated under the comparison procedure used in this study: ordered-sequence equality when an outer ORDER BY is present and multiset equality otherwise. This is not a universal proof of SQL equivalence. It is an empirical validation certificate for a concrete database instance, schema version, SQL fragment, and parameter set.

For each template T , the system maintains a candidate pool C_T :

$$C(T) = \{c_1, c_2, \dots, c_k\}. \quad (2)$$

Each candidate has a source type, source detail, SQL text, canonical hash, structural-validation state, empirical-equivalence state, benchmark records, and decision state.

2.3. Candidate Generation Sources and Deterministic Rule Families

The lifecycle is source-neutral. In the current evaluation, candidate SQL formulations are produced by deterministic guarded rules and by a local LLM path. The same intake model can also admit manually supplied alternatives or outputs from learned, synthesis-based, middleware, or hosted-model rewrite systems, provided that they are alternative SQL formulations for a known recurring template. The source is recorded for provenance, but it does not determine acceptance.

The deterministic source is intentionally limited. It covers representative guarded rewrite families such as redundant predicate elimination, boolean and comparison simplification, positive 'IN' to 'EXISTS', guarded 'NOT IN' to 'NOT EXISTS', 'COUNT(*) > 0' to 'EXISTS', implicit-to-explicit join normalization, redundant outer 'GROUP BY' elimination, and subquery column pruning. These rule families are used to create auditable candidates for the lifecycle evaluation; they are not presented as a complete SQL rewrite catalog. Applying a deterministic rule, therefore, only creates a candidate. It does not acti-

vate the candidate and does not bypass structural validation, empirical result-equivalence validation, paired benchmarking, or conservative decision criteria.

2.4. Structural and Empirical Validation

The structural validation layer checks that the candidate remains inside the supported SQL fragment, preserves parameter positions, preserves the observable output shape, and does not introduce unsupported clauses or side effects. This layer is not a semantic proof. Its role is to reject clearly unsafe candidates before any empirical result comparison.

The empirical validation layer executes the baseline and candidate over controlled parameter sets and compares the results. Unordered outputs are compared as row multisets, preserving duplicates. Ordered outputs are compared as ordered sequences. This design reflects SQL's bag semantics and the fact that ORDER BY makes order observable.

The validation result is stored with the candidate and associated with the schema snapshot, parameter set, result-comparison mode, and, when available, the failure reason. Candidates that fail structural or empirical validation are not benchmarked.

The supported SQL fragment and the validation procedure are stated precisely so that the scope of the equivalence certificate is clear. Validation applies to read-only SELECT statements; results are compared as row multisets under SQL bag semantics, except when a complete observable outer ORDER BY makes order part of the result, in which case ordered sequences are compared. The validation stage uses recorded validation parameter vectors for each template, executed in the same read-only transaction with a configured row cap and timeout. Constructs that make result-set comparison unsafe or undefined are excluded or marked inconclusive rather than validated: LIMIT without a complete outer ordering, volatile or non-deterministic functions, and queries whose ties under a partial ordering make sequence comparison ambiguous. Empirical equivalence is therefore reported as a certificate over a concrete instance, schema version, supported SQL fragment, and validation parameter set rather than as formal SQL equivalence.

Held-out parameter vectors are used in the reported workflow for independent runtime confirmation of candidates that have already passed structural and empirical validation. The held-out stage, therefore, confirms performance behavior under fresh paired measurements; it does not re-certify equivalence for every possible parameter value or database state. This remaining risk is acknowledged as part of the empirical-equivalence limitation and is mitigated by structural guards, validation records, and retained provenance.

2.5. Paired Benchmarking and Activation Criteria

Candidates that pass validation are measured against the baseline under a paired protocol. The same parameter values are used for both formulations, and the execution order alternates between baseline-first and candidate-first pairs. In this paper, this alternating order is referred to as AB/BA paired benchmarking; the term is used as a local protocol label, not as the name of a general benchmark. This design reduces bias from cache warming and short-term system state.

For a benchmark pair i , let b_i be the baseline runtime and a_i the candidate runtime for the same template and parameter values. The relative improvement is:

$$\Delta_i = (b_i - a_i)/b_i, \quad (3)$$

Positive Δ_i indicates improvement and negative Δ_i indicates regression. The system preserves both cases as evidence.

A candidate is considered for activation only when all of the following conditions hold: $\text{Valid}(c)$, $n(c) \geq n_{\min}$, $\text{median}(\Delta_c) \geq \theta$, $p_c \leq \alpha$, and $\text{RegressionGuard}(c)$ is false. Here, $\text{Valid}(c)$ denotes structural and empirical validation, $n(c)$ is the number of paired

observations, θ is the minimum practical effect threshold, p_c is the paired statistical-test result, and RegressionGuard captures timeout, instability, error, or mismatch conditions. When statistical testing is used for activation, the paired nature of the observations is preserved, using a paired non-parametric test where appropriate [19].

The reported thresholds are conservative defaults for the controlled evaluation rather than universal deployment constants. The minimum of 30 paired observations gives each activation decision a non-trivial repeated-measurement basis, the 2% practical-effect threshold avoids promoting candidates for negligible timing differences, and $\alpha = 0.05$ is used as a conventional significance level for the one-sided paired test. These values are suitable for the reported local experimental setting, while workload-specific threshold tuning and wider sensitivity analysis are left for future operational studies.

Measurement is separated into a search stage and an independent held-out confirmation stage, and a candidate is reported as promoted only after it satisfies the activation criteria on the held-out stage. For parameterized templates, the two stages use disjoint parameter sets drawn from the same template (for TPC-H, 70 search and 30 held-out parameter vectors per query), so confirmation is measured on parameter values not used during search. For fixed-literal workloads such as JOB/IMDB, where a parameter split does not apply, the held-out stage is a separate set of confirmation repetitions: the candidate is re-measured against the baseline in fresh paired runs under the same AB/BA protocol, so confirmation is replication on independent executions rather than generalization across parameters. In either case, the held-out stage is evidence for the tested schema, data, and execution context; it is not a claim of equivalence or improvement across all possible parameters or database states.

2.6. Controlled Evaluation Setup

The lifecycle was evaluated in a controlled execution setting. The evaluation setup executes workload manifests, admits candidate SQL formulations, applies structural and empirical validation, performs paired runtime measurements, and records the resulting evidence. The target engine documentation is used for the behavior and configuration context [20]. Its EXPLAIN documentation is used for execution-plan interpretation [21]. The `pg_stat_statements` extension provides the optional monitoring context recorded in the public artifact [22]. Local model proposals are executed through an Ollama-compatible local runtime [23].

The main text keeps software details to the minimum required for reproducibility. Additional scripts, configuration files, run outputs, checksums, and monitoring exports are available in the public reproducibility materials.

2.7. Experimental Design, Workloads, and Reproducibility

The evaluation tests the evidence lifecycle rather than a single speedup number. The main questions are whether the proposed lifecycle can generate, validate, measure, record, and decide on SQL rewrite candidates; whether it can identify useful candidates for selected recurring templates; and whether it can withhold activation for neutral, unsupported, unstable, or regressive alternatives.

All reported measurements use a controlled Docker-based execution environment with a single target engine. The workloads are executed from explicit manifests, and the run outputs include provenance records, benchmark rows, validation outcomes, and monitoring files. The local LLM path uses the `qwen3.6:35b-a3b-q4_K_M` model tag through Ollama, where LLM-origin candidates are enabled. Appendix A and Table A1 summarize the execution environment and run protocol in a compact, self-contained form. Table 1 summarizes the role of each workload in the evaluation design.

Table 1. Workload roles in the evaluation.

Workload	Size/Source	Role in Evaluation	Interpretation Boundary
Deterministic-rule validation set	8 representative scenarios	Checks guarded rule-path feasibility.	Demonstrates end-to-end rule-path behavior; does not imply arbitrary SQL coverage.
TPC-H SF1	22 templates	Tests conservative behavior on a mature analytical benchmark [24].	No broad TPC-H speedup claim is made.
Controlled real-world-style corpus	10 targeted scenarios	Tests recurring SQL anti-patterns resembling reporting, BI, ORM, and ETL formulations.	Controlled corpus, not an external operational workload.
JOB/IMDB	113 fixed-literal public queries	Tests repeated behavior on join-heavy public queries [25].	Public-workload pilot evidence; promoted sets need not be identical across runs.

Operational details of the LLM candidate source, including prompt templates, model tags, run metadata, duplicate handling, invalid-SQL outcomes, and scripts used to launch the experiments, are preserved in the public artifact. The main text does not reproduce the full prompting configuration because the paper evaluates the evidence lifecycle rather than the behavior of a single prompt or model setting.

The deterministic rule formulations are expressed over SQL and relational semantics where possible, so they are not conceptually limited to one database engine. All reported evidence, however, is validated only in the reported target engine, including its parser behavior, dialect, optimizer, execution engine, and recorded workload environment. The results are not presented as universal SQL speedups, formal equivalence guarantees, or empirical cross-DBMS claims.

2.8. Language Model and Coding Agent Use

This project studies local language models as candidate generators for SQL rewrites. Generated candidates have no semantic authority. They must pass structural safety checks, empirical result-set validation, and paired benchmarking before any promotion or activation decision. Model tag, prompt provenance, and run metadata are recorded where available.

This trust model is consistent with work on explainable, robust, and secure AI-assisted systems, where traceability and independent safeguards are important for using model outputs in technical workflows [26]. In this paper, these safeguards are SQL-specific: structural admissibility, empirical result-equivalence validation, paired benchmarking, and guarded activation are all recorded as evidence before an LLM-generated candidate can be used.

Coding-agent tools were also used as assistance for the experimental software and repository materials, including support for refactoring, code editing suggestions, debugging, verification planning, documentation wording, and analysis of experimental result data. The reported workflow is not presented as a one-shot generated result. Accepted changes were selected, revised, tested where appropriate, and committed as a source. Repro-

ducibility is based on the committed specifications, source code, dependency declarations, scripts, run protocols, and recorded evidence, not on replaying nondeterministic prompts.

3. Results

3.1. Overview of Reported Evidence

The evaluation produced positive, neutral, negative, and withheld outcomes. This is expected under a conservative lifecycle. A useful lifecycle should identify opportunities, but it should also refuse activation when evidence is weak, unstable, semantically invalid, unsupported, or regressive.

Tables 2–5 present the results as a candidate funnel. Table 2 gives the workload-level interpretation, Table 3 gives run-level counts, Table 4 reports held-out promotion evidence, and Table 5 traces generated, validated, and promoted candidates by source. Read together, they follow the lifecycle from generation to structural and empirical validation, paired measurement, and either promotion, retention, or withholding. Non-promoted candidates are not discarded silently; their validation and measurement records remain in the candidate pool and can support later retention, rejection, or deactivation decisions.

Table 2. Workload interpretation.

Workload	Main Reported Outcome	Main Interpretation
Deterministic-rule validation set	Eight representative rule scenarios were evaluated. The strongest COUNT(*) > 0 to EXISTS fixture showed a median runtime decrease from approximately 1981.6 ms to approximately 6.7 ms.	The deterministic rule path can support full lifecycle execution on controlled guarded cases.
TPC-H SF1	Four full runs completed all 22 templates. The first three runs promoted no candidate; the fourth promoted one held-out Q10 candidate with approximately 25% median improvement.	The lifecycle behaves conservatively on mature analytical templates and can still accept a supported candidate.
Controlled real-world-style corpus	Four repeated runs completed all 10 scenarios; promoted candidates per run were 2, 2, 1, and 2. The most stable correlated scalar-subquery case showed approximately 73–75% held-out median improvement.	Strong candidates can be found for selected anti-patterns, but promotion is withheld when stability or held-out evidence is insufficient.
JOB/IMDB	Four full runs over 113 queries produced 19, 23, 21, and 20 held-out positive candidates.	Public-workload pilot evidence supports selected recurring join-heavy templates, with non-identical promoted sets across runs.

Table 3. Run-level evidence.

Workload Group	Repeated Runs	Completed Templates/Cases	Validated Candidates or Valid Pairs	Promotions/ Positive Cases	Main Quantitative Evidence
TPC-H SF1	4	22 per run	Validated candidates: 11, 9, 10, 11; valid benchmark pairs: 330, 270, 300, 360	0, 0, 0, 1	Fourth run promoted one held-out Q10 candidate with approximately 25% median improvement.
Controlled real-world-style corpus	4	10 per run	Full completion in all runs	2, 2, 1, 2	Correlated scalar-subquery case confirmed in all four runs with approximately 73–75% held-out median improvement.
JOB/IMDB	4	113 per run	Full-corpus completion in all runs	19, 23, 21, 20 held-out positive candidates	Deterministic join normalization and local LLM proposals both contributed to promoting candidates.

Table 4. Promotion evidence.

Promotion (Source)	Runs Promoted	Baseline Median (ms)	Candidate Median (ms)	Held-Out Improvement %	<i>p</i>	95% CI %
TPC-H Q10 (LLM)	run 4	251.22	188.49	24.97	<0.001	18.03–28.31
Controlled RW-01, correlated scalar subquery (LLM)	all 4 runs	120.69–140.92	30.49–37.23	73.27–74.99	<0.001	57.80–79.94
Controlled RW-10, multiple aggregate passes (LLM)	3 of 4 runs	484.24–629.58	88.18–114.13	81.79–82.66	<0.001	73.45–87.00

Table 5. Candidate funnel by source.

Workload (Run)	Source	Valid	Promoted/ Total
TPC-H (run 4)	local LLM (all candidates)	11	1/18
TPC-H (run 4)	deterministic rules	0	0/0
Controlled (per run)	local LLM	8–9	1–2/9
Controlled (per run)	rules in_to_exists, redundant_group_by	2	0/2
JOB/IMDB (run 2)	rule implicit_to_explicit_join	67	9/67
JOB/IMDB (run 2)	rule implicit_to_explicit_join_alternate_order	67	0/67
JOB/IMDB (run 2)	local LLM	91	14/96

Table 4 reports the inferential statistics behind each promotion. Promotion uses a one-sided paired Wilcoxon signed-rank test on the paired improvement values together

with 95% percentile bootstrap confidence intervals; every promotion is supported by $n = 30$ held-out paired observations. The correlated scalar-subquery case (RW-01) was promoted in all four controlled-corpus runs and the multiple-aggregate-pass case (RW-10) in three of four, so their rows report the across-run ranges.

Table 5 reports, per workload and candidate source, how many candidates were generated, validated, and promoted, which quantifies the contribution of the local LLM relative to the deterministic rules.

Two observations follow. First, every TPC-H and controlled-corpus promotion was LLM-generated; the deterministic rules present in those workloads (`in_to_exists`, `redundant_group_by`) produced valid but never-promoted candidates. The LLM's promoted rewrites are structural transformations outside the limited deterministic catalog: eager aggregation (group-by pushdown) for TPC-H Q10, decorrelation of correlated scalar subqueries into left joins for the controlled RW-01 case, and collapsing three independent scalar-aggregate subqueries into a single scan with conditional aggregation for RW-10 (Listing 1). Second, in JOB/IMDB, the deterministic `implicit_to_explicit_join` rule does contribute (8–11 promotions per run), so rule-origin and LLM-origin candidates are complementary rather than redundant. This is the sense in which local LLMs add value: not as trusted authorities, but as a source of structurally different candidates that the rule catalog does not generate and that still must pass the same validation and measurement barriers.

Listing 1. Representative LLM-generated rewrite (controlled RW-10): three independent scalar-aggregate subqueries collapsed into a single scan with conditional aggregation. The candidate passed empirical result-equivalence validation and was promoted with approximately 82% held-out median improvement.

-- Original

```
SELECT
  (SELECT SUM(l_extendedprice) FROM lineitem
   WHERE l_shipmode = $1 AND l_shipdate >= $4 AND l_shipdate < $5) AS
mode1_revenue,
  (SELECT SUM(l_extendedprice) FROM lineitem
   WHERE l_shipmode = $2 AND l_shipdate >= $4 AND l_shipdate < $5) AS
mode2_revenue,
  (SELECT SUM(l_extendedprice) FROM lineitem
   WHERE l_shipmode = $3 AND l_shipdate >= $4 AND l_shipdate < $5) AS
mode3_revenue;
```

-- Promoted candidate

```
SELECT
  SUM(l_extendedprice) FILTER (WHERE l_shipmode = $1) AS mode1_revenue,
  SUM(l_extendedprice) FILTER (WHERE l_shipmode = $2) AS mode2_revenue,
  SUM(l_extendedprice) FILTER (WHERE l_shipmode = $3) AS mode3_revenue
FROM lineitem
WHERE l_shipdate >= $4 AND l_shipdate < $5;
```

3.2. Deterministic-Rule Validation Results

The deterministic-rule validation set checks whether guarded rule-origin candidates can pass the complete lifecycle. The clearest case is the `'COUNT(*) > 0'` to `'EXISTS'` fixture: the candidate avoids counting all matching rows when only existence is required. In the reported deterministic setup, the median runtime decreases from approximately 1981.6 ms for the baseline to approximately 6.7 ms for the candidate. This is reported as an end-to-end workflow result on a controlled fixture, not as a broad benchmark claim.

This fixture is a hand-built guarded-rule scenario, constructed to exercise the deterministic rule path end to end on a case where the rewrite is unambiguously beneficial ('EXISTS' stops at the first matching row, whereas 'COUNT(*) > 0' scans all matching rows). The near-300-fold figure should therefore be read as a feasibility demonstration of the rule path on a favorable case, not as a representative or expected speedup. The representative evidence is the conservative behavior on the mature TPC-H templates (Section 3.3), where most validated alternatives were neutral or regressive and were withheld—including search-phase regressions as large as −133.40% (Q17) and −130.10% (Q18) that the activation criteria refused.

The same validation set also includes representative cases for redundant predicate elimination, Boolean simplification, comparison normalization, positive 'IN' to 'EXISTS', implicit-to-explicit join normalization, redundant outer 'GROUP BY' elimination, and subquery column pruning. These cases show that the deterministic rule path can generate structurally valid candidates across several guarded patterns. They do not imply coverage of arbitrary SQL, and they do not bypass the empirical validation and benchmarking stages.

3.3. TPC-H SF1 Results: Conservative Behavior on Mature Analytical Templates

Across four full TPC-H SF1 runs, all 22 templates were completed. The first three runs promoted no candidate, while the fourth run promoted one held-out candidate for the Q10 template. As shown in Table 3, the number of candidates that passed empirical result-equivalence validation varied modestly across runs, and the valid benchmark-pair counts were 330, 270, 300, and 360.

The TPC-H evidence is therefore interpreted as a conservative-behavior test on a mature analytical benchmark, not as a broad TPC-H speedup claim. Most generated and validated alternatives remained neutral or regressive and were not activated. One later run produced a held-out positive Q10 case. Both outcomes support the method: the lifecycle should withhold activation when evidence is weak, and it should allow a candidate to pass when independent validation and paired held-out measurements support it.

The recorded 'EXPLAIN (ANALYZE, BUFFERS)' plans make the Q10 improvement interpretable at the plan level. The original formulation joins the four relations ('customer', 'orders', 'lineitem', 'nation') and aggregates the joined rows grouped by customer, so the join and the final sort process every qualifying 'lineitem' row. The promoted candidate computes the per-customer revenue aggregate inside a derived table over 'orders' and 'lineitem'—a group-by pushdown, or eager aggregation—and collapses it to one row per customer before joining the small 'customer' and 'nation' relations, so far fewer rows reach the outer join and the 'ORDER BY... LIMIT'. On the recorded plan capture, the rewrite reduced execution time from 240.058 ms to 172.260 ms; this single annotated plan pair illustrates the mechanism, while the paired medians in Table 4 remain the activation evidence. The rewrite's correctness depends on the key constraints of 'customer' and 'nation'—exactly the class of schema knowledge that empirical validation confirmed on the configured instance.

The same plan-level reading also explains the withheld cases. For the Q17 and Q18 templates, structurally valid and result-equivalent candidates were measured but not promoted because their paired search-stage runtimes were large regressions (Section 3.2). At the plan level, the alternative formulation did not lead the planner to a cheaper execution strategy than the baseline: rather than removing work, it changed the statement in a way the optimizer realized with a more expensive plan, so the regression guard and the practical-effect threshold withheld activation. This is the intended asymmetry of the lifecycle: a candidate is activated only when paired and held-out evidence shows a real improvement, while an equivalent-but-slower rewrite is recorded and rejected rather than adopted. The

benchmark records for these withheld cases, including captured execution plans where available, are retained in the public artifact alongside those of the promoted candidates.

3.4. Controlled Real-World-Style Corpus Results: Selected Repeatable Positives

The controlled real-world-style corpus contains targeted SQL anti-patterns over generated relational data. Its role is to test whether rule-origin and LLM-origin candidates can survive the lifecycle on formulations that resemble recurring reports or machine-generated SQL. The scenarios include correlated scalar subqueries, ‘IN’ subqueries over relationship tables, redundant outer aggregation layers, duplicate producing joins hidden by ‘DISTINCT’, filter-placement variants, OR-predicate splitting opportunities, single-reference common table expression (CTE) cases, scalar aggregate subqueries, self-join lookup patterns, and multiple scalar aggregate passes over the same table.

The ten scenarios are synthetic anti-patterns over generated SF1 data rather than queries captured from a specific operational deployment. Each instantiates a documented recurring formulation problem from reporting, business-intelligence, ORM, and ETL practice, and the set was designed to cover the recurring shapes of machine-generated and handwritten reporting SQL rather than the distribution of any one workload. This is why the corpus is labeled controlled and interpreted as pilot evidence (Table 1) rather than as an external operational benchmark; its role is to test whether the lifecycle distinguishes strong candidates from neutral or negative ones on realistic formulations, not to estimate a population speedup.

Across the four repeated runs, the corpus consistently completed all 10 scenarios. The most stable promoted case is the correlated scalar-subquery scenario (RW-01): it was confirmed on 30 held-out paired observations in all four runs, with held-out median improvements of 73.27%, 73.58%, 74.74%, and 74.99% and 95% bootstrap confidence intervals contained in 57.80–79.94% (Table 4). A second case, multiple scalar-aggregate passes over lineitem (RW-10), was promoted and confirmed in three of the four runs, with held-out median improvements of 81.79–82.66% ($n = 30$ each); in the remaining run its search-stage effect was only 29.98% with a wide confidence interval and it did not reach held-out confirmation, which is why the per-run promotion counts are 2, 2, 1, 2. Another scalar-aggregate scenario (RW-09) repeatedly showed search-stage positive evidence (for example, a 42.53% search-stage improvement in one run) but did not consistently satisfy the conservative promotion criteria and was withheld.

The controlled corpus also gives direct evidence that the empirical check discriminates rather than merely passing already-correct candidates. In one run, an LLM-generated candidate that passed the structural checks was rejected by empirical result-equivalence validation as a row-multiset mismatch: it returned 27 rows where the baseline returned 28, with two rows present only in the baseline and one present only in the candidate, a genuine semantic divergence on the data instance, caught before any benchmarking. In the TPC-H runs, the validation similarly rejected an ordered-row mismatch (identical rows in a different order under ORDER BY) and a non-executable candidate (an aggregate function placed in WHERE). Discrimination also occurs at the held-out stage, in the performance dimension: the RW-10 candidate that showed a positive search-stage effect in the run noted above failed to confirm on held-out measurement and was withheld. The held-out stage in the current workflow re-measures runtime rather than re-running equivalence, so it confirms performance rather than catching new semantic mismatches; extending equivalence re-checking to held-out parameters is left to future work.

3.5. JOB/IMDB Results: Repeated Public-Workload Pilot Evidence

The JOB/IMDB experiments evaluate the lifecycle on 113 join-heavy fixed-literal queries. Four full runs were recorded. They produced 19, 23, 21, and 20 held-out positive candidates, respectively. The candidate-source split varied by run, but both deterministic join normalization and local LLM proposals contributed to promoted candidates. The deterministic ‘implicit_to_explicit_join’ rule produced 8, 9, 11, and 10 promotions across the four runs, while the local LLM path produced 11, 14, 10, and 10 promotions. The alternate explicit-join-order candidate was validated but not promoted in these runs.

The exact set of promoted JOB/IMDB queries is not identical across runs. This is treated as controlled pilot evidence for selected recurring join-heavy templates, not as a claim that the same JOB/IMDB queries will always be promoted. The variation is also informative: join-heavy workloads are sensitive to runtime noise, plan choices, candidate formulation, and thresholding behavior. Repeated evidence is therefore more useful than a single promoted query list.

The overlap across the four runs makes “comparable but non-identical” precise. Of the queries promoted at least once, 12 were promoted in all four runs (a stable core), three in exactly three runs, six in exactly two, and 14 in exactly one; the union over the four runs is 35 distinct queries, and the pairwise Jaccard similarity of the promoted sets ranges from 0.48 to 0.63 (mean 0.54). This stable core supports the intended behavior of the lifecycle, while most run-to-run variability is concentrated among marginal candidates. That variability has several identifiable sources: local-LLM non-determinism (the model proposes different candidates across invocations at sampling temperature 0.7, so the candidate set itself varies by run), short-term runtime and cache noise on a single host, target engine plan-choice sensitivity for join-heavy queries, and threshold-boundary effects in which a candidate with a near-threshold effect passes in one run and misses held-out confirmation in another. The two-stage search-then-held-out design absorbs part of this: the search stage proposes candidates and the disjoint held-out stage independently confirms them, so a marginal candidate that does not replicate is withheld rather than promoted.

3.6. Consolidated Result Interpretation

The evaluation supports four focused claims. First, the reported evidence demonstrates the full candidate lifecycle: candidates can be generated, structurally checked, empirically validated, measured in paired baseline/candidate runs, stored with provenance, and accepted or rejected through an explicit decision path.

Second, the lifecycle is conservative but still able to accept supported candidates. The TPC-H runs are dominated by non-activation, validation rejections, neutral candidates, and regressions, but the fourth run also shows that a candidate can pass the full evidence path when held-out paired measurements support it.

Third, selected positive cases are repeatable but not universal. The correlated scalar-subquery case is stable across repeated controlled-corpus runs; the multiple-aggregate-pass case is strong but not promoted in every repetition; JOB/IMDB produces comparable positive counts across repeated full runs but not the same promoted set every time.

Fourth, deterministic rules and LLM-generated candidates are complementary candidate sources. Deterministic rules provide controlled and explainable transformations. LLM proposals expand the candidate space. Neither source is trusted automatically, and both are subject to the same validation and measurement barriers.

The activation thresholds were not finely tuned, and the promoted set is insensitive to the practical-improvement threshold over a wide range. The promoted candidates clear the 2% threshold by a large margin, approximately 25% for TPC-H Q10 and 73–82% for the controlled-corpus cases (Table 4), so varying the threshold between 1% and 10% does

not change which candidates are promoted; only borderline search-phase candidates near the threshold are affected. The minimum of 30 promotion pairs and the alpha of 0.05 are conventional choices for this controlled evaluation and are reported explicitly to make the decision rule reproducible. Crucially, the conservative outcome is produced by the combination of a practical-effect threshold, a paired statistical test, a stability guard, and an independent held-out confirmation, not by a precise threshold value: false activations are limited by requiring agreement across all of these, and missed opportunities (candidates withheld at the margin) are the deliberate cost of that conservatism. The held-out confirmation stage, rather than the 2% value, is what removes the marginal cases (Section 3.5). A full sensitivity study that sweeps the practical-effect threshold, the minimum-pairs requirement, and the significance level jointly is beyond the scope of this evaluation and is left to future work.

4. Discussion

4.1. Interpretation of the Evidence

The results support the feasibility of the proposed evidence-driven lifecycle under controlled evaluation. The evidence is strongest at the process level: candidates are generated, screened by structural checks, checked for empirical result equivalence, measured in paired runs, stored with provenance, and advanced only when the decision criteria are satisfied. This framing is important because the evaluation contains positive, neutral, negative, and withheld outcomes rather than uniform benchmark-wide speedups.

The repeated runs sharpen the interpretation. TPC-H is mostly a conservative-decision workload, with one fourth-run held-out positive case rather than a broad benchmark-wide improvement. The controlled real-world-style corpus reproduced a strong held-out positive case and showed that another high-effect case can remain unpromoted when stability or held-out criteria are not satisfied. JOB/IMDB produced comparable positive counts across four full runs, but not an identical promoted set. The appropriate claim is therefore engine-specific pilot evidence for selected recurring templates, not broad benchmark dominance.

4.2. Relation to Prior Systems

Recent LLM-based rewriting systems can be interpreted as upstream sources of candidate SQL formulations. GenRewrite can contribute candidates derived from natural-language rewrite rules and correction loops [12]. LITHE can contribute candidates produced by database-sensitive prompting and advisor-style exploration [13]. QUITE can provide candidates generated in a feedback-aware multi-agent workflow [14]. R-Bot can contribute evidence-guided rewrite candidates [15]. E3-Rewrite can contribute alternatives produced under executability, equivalence, and efficiency objectives [16]. Their outputs could enter the proposed lifecycle whenever they produce an executable alternative for a known recurring template.

The same interpretation applies to non-LLM systems. WeTune can provide rule-derived candidates [8]. LearnedRewrite can provide search-derived candidates [9]. SlabCity can provide synthesized alternatives [10]. QueryBooster can provide middleware-derived or human-authored candidates [11]. The lifecycle is therefore deliberately source-neutral: it controls what happens after a candidate exists, rather than prescribing one preferred generator.

The contribution is restricted to persistent candidate management for recurring SQL templates. The managed evidence includes provenance, structural admissibility, empirical equivalence, paired measurement, statistical state, guarded activation, and deactivation when regressions appear.

Table 6 summarizes these distinctions as a capability comparison. The research systems concentrate on candidate generation and, in several cases, a per-candidate semantic or performance check. Compared with these systems, the present work emphasizes persistent candidate management after generation: provenance, empirical validation, paired-measurement evidence, guarded activation, and post-activation deactivation across repeated runs at the SQL-formulation level.

Table 6. Capability comparison with representative rewrite and learned-optimization systems (Y = present, P = partial, N = absent or not applicable).

System	Candidates	Semantic Check	Paired Measure	Persistent Pool	Guarded Activation	Deactivation
WeTune [8]	Y	Y	N	N	N	N
LearnedRewrite [9]	Y	P	N	N	N	N
SlabCity [10]	Y	N	N	N	N	N
QueryBooster [11]	Y	N	N	N	N	N
GenRewrite [12]	Y	Y	P	N	N	N
LITHE [13]	Y	Y	Y	N	N	N
QUITE [14]	Y	P	P	N	N	N
R-Bot [15]	Y	P	N	N	N	N
E3-Rewrite [16]	Y	Y	Y	N	N	N
Bao [18]	N	N	Y	N	Y	P
This work	Y	Y	Y	Y	Y	Y

4.3. Practical Applicability

The approach is most relevant for recurring workloads where the cost of candidate generation, validation, and benchmarking can be spread over later executions. Typical cases include BI dashboards, reporting jobs, ETL pipelines, dbt-style transformations, ORM-generated recurring queries, and heavy analytical templates.

The approach is less suitable for lightweight one-off queries. It is also not intended to replace the native optimizer. In the reported evaluation, each candidate SQL formulation is submitted to the target engine, whose optimizer selects the physical plan.

A practical deployment would likely begin as an advisory workflow rather than a fully automatic query-routing component. The evidence path can help a database administrator or data-engineering team distinguish between supported candidates, neutral alternatives, regressions, and cases where a generator produced invalid SQL.

4.4. Limitations and Threats to Validity

All reported measurements target one database engine implementation. Several deterministic rules are stated over general SQL semantics and are not conceptually specific to the evaluated engine, but parser behavior, dialect details, optimizer transformations, planner feedback, and runtime plans may differ across DBMSs. Cross-engine evaluation is therefore future work rather than a current empirical claim.

Empirical result-equivalence validation is not formal SQL equivalence. The validation result is bounded by the selected database instance, parameter sets, SQL fragment, and comparison procedure. The deterministic rules use guards and structural restrictions, but they are still treated as candidate sources rather than final proof of equivalence.

The evaluation is controlled and repeated, but it remains a pilot evaluation. TPC-H, the controlled corpus, and JOB/IMDB play different roles and should not be combined into

a single global speedup number. The repeated JOB/IMDB runs show comparable positive counts, but the exact promoted set varies.

The LLM path uses local model execution. This supports privacy and reproducibility of the evaluation, but it limits claims about model-scale generality or superiority over hosted models.

The current work does not validate a full multi-armed bandit (MAB) strategy. The recorded evidence structure can support future budget-aware candidate selection, but the reported experiments use conservative activation rather than a comparative bandit-policy evaluation.

Resource monitoring is contextual only. The reported measurements do not support claims about energy efficiency, graphics processing unit (GPU) efficiency, or carbon impact.

Integration into an advisory workflow is outside the current evaluation. The evaluation is conducted as a controlled research study.

5. Conclusions

This paper presented a feedback-driven lifecycle for SQL optimization through empirically validated query rewrites. The main result is not a new rewrite generator, but a persistent candidate-management process in which alternative SQL formulations are treated as hypotheses. A candidate is trusted only after provenance recording, structural checks, empirical result-equivalence validation, paired benchmarking, and conservative activation criteria support it.

The controlled evaluation demonstrates this lifecycle over deterministic-rule examples, repeated TPC-H SF1 runs, repeated controlled real-world-style runs, and repeated full JOB/IMDB runs. All reported measurements were obtained in the reported single-engine execution environment, so the evidence is DBMS-specific; however, the lifecycle itself is formulated at the SQL-candidate management level. The results support feasibility and selected positive cases while also showing conservative non-activation, validation rejection, neutral outcomes, regressions, and run-to-run variability. This combination is central to the contribution: a useful lifecycle must identify opportunities, but it must also refuse activation when evidence is insufficient or unstable.

Future work should extend the supported SQL fragment, compare multiple database engines and model configurations, add stronger plan analysis, evaluate budget-aware candidate-selection policies, and study advisory workflows under realistic operational constraints. These extensions require additional experiments and are intentionally kept outside the present lifecycle-validation paper. Formal equivalence support can also be expanded for restricted SQL fragments, while preserving the distinction between candidate generation and semantic authority.

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

Funding: K.K. gratefully acknowledges the support provided under Project UNITE BG16RFPR002-1.014-0004, funded by PRIDST.

Data Availability Statement: The public benchmark resources used in this study are available from their respective sources. The experimental software source code, specifications, scripts, Docker configuration, SQL schema, tests, curated result summaries, checksummed result bundles, and reproducibility metadata are available in the public GitHub release artifact-v0.1: <https://github.com/KostovMartin/mk-feedback-driven-sql-optimization/releases/tag/artifact-v0.1> (accessed on

30 June 2026). The manuscript refers to commit 36b913b12ca1a7435cc66e15c3ffa272f77907e8. Later repository revisions may differ from the version used for the reported results. Raw database volumes, external raw datasets, generated data volumes, and local model files are not redistributed because of size, licensing, and reproducibility constraints. The public artifact supports independent re-execution of the reported workflows as follows. A lightweight verification entry point (run-fast-check.ps1) checks the toolchain. The main runs are produced with run-main-run.ps1 -Model <model-tag> -Corpus tpch | real-world | job-imdb, where the model tag is always explicit and recorded in the run metadata. The deterministic-rule cases, TPC-H SF1, and the controlled corpus generate their data locally; the JOB/IMDB corpus additionally requires externally staged IMDB CSV files prepared with prepare-job-imdb-resources.ps1. Each run writes machine-readable exports and a generated post hoc Markdown summary under experiment-runs/<corpus>/<run-id>/, together with a checksummed raw run-data bundle under experiment-artifacts/. The following inputs are not redistributed and therefore reproduce structurally but not byte-for-byte: raw and generated database volumes, the external IMDB CSV files, and the local model weights. Re-running regenerates equivalent data and measurements rather than identical bytes, and the run-to-run numbers vary for the reasons discussed in Section 3.5.

Acknowledgments: During the preparation of this manuscript and supporting experimental materials, the authors used local language models through Ollama v0.20.7 (local runtime; model tags recorded where available in run metadata) for SQL-candidate generation experiments and OpenAI Codex v5.4 (cloud service accessed in March–May 2026) for coding-agent assistance, including refactoring suggestions, debugging support, verification planning, documentation wording, and analysis of experimental result data. The authors reviewed and edited the outputs and take full responsibility for the content of this publication. Candidate SQL generated by language models was not accepted without structural validation, empirical result-equivalence validation, paired benchmarking, and decision checks.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

AB/BA	Alternating baseline/candidate and candidate/baseline paired benchmark order
BI	Business intelligence
CBO	Cost-based optimizer
CTE	Common table expression
DBMS	Database management system
ETL	Extract, transform, load
IMDB	Internet Movie Database
JOB	Join Order Benchmark
LLM	Large language model
MAB	Multi-armed bandit
GPU	Graphics processing unit
ORM	Object-relational mapping
SQL	Structured Query Language
TPC-H SF1	Transaction Processing Performance Council H decision-support benchmark at scale factor 1

Appendix A. Experimental Environment and Run Protocol

Appendix A gives the execution context needed to interpret the reported runtime measurements. Following the usual reporting practice for empirical systems papers, the table includes the main hardware class, execution mode, software versions, workload

scale, benchmark protocol, and reproducibility-material status. The article does not claim hardware-independent performance or GPU-efficiency results; the environment is reported to make the measurements auditable and reproducible.

Table A1. Execution stack and host-monitoring information recorded for the public artifact.

Aspect	Reported/Recorded Information
Host machine	Single local workstation used for the reported controlled runs.
Operating environment	Windows 11 Build 26200 host with PowerShell-based run scripts.
Processor	AMD Ryzen 9 5950X 16-Core Processor, 3401 MHz, 16 physical cores, and 32 logical processors.
System memory	Kingston HyperX Fury Black, 64.0 GB DDR4 memory at 3000 MHz.
GPU/local-model memory	Gainward NVIDIA RTX 5090 with 32 GB VRAM, used for the local Ollama-compatible model path where LLM-origin candidates are enabled.
Containerization	Docker v4.78.0 and Docker Compose v5.1.4.
Target DBMS	PostgreSQL v18.3.
SQL-candidate validation and rewrite service	Python v3.14 FastAPI service used in the evaluation workflow.
Workload execution controller	.NET 10 runner used to execute manifests and paired benchmark runs; repository requirements list .NET SDK 10.0.300.
Package/runtime utilities	Python v3.14, uv, Docker/Docker Compose, Windows PowerShell, and an Ollama-compatible local model runtime.
Evaluated workloads	Deterministic-rule validation set; TPC-H SF1; controlled real-world-style anti-pattern corpus; JOB/IMDB fixed-literal queries.
TPC-H scale	Scale factor 1.
TPC-H parameterization	70 search parameter sets per query and 30 held-out parameter sets per query in the reported TPC-H configuration.
Benchmark pairing	Baseline and candidate formulations are measured on the same parameter values; execution order alternates between baseline/candidate and candidate/baseline pairs.
Promotion settings	Minimum promotion pairs: 30; promotion alpha: 0.05; practical improvement threshold: 2.0%.
Repeated runs	TPC-H SF1, the controlled real-world-style corpus, and JOB/IMDB are each reported over four repeated full runs.
Recorded evidence	Candidate provenance, validation outcomes, benchmark rows, activation decisions, interpreted post hoc summaries, checksums, and monitoring files.
Redistribution limits	Raw database volumes, external raw datasets, generated data volumes, and local model files are not redistributed because of size, licensing, and reproducibility constraints.

Additional per-run monitoring details are preserved in the raw run-data bundles, including host, GPU, container, and rewrite-service monitoring exports. These records are used for contextual reproducibility and are not used to support energy-efficiency or carbon-impact claims.

References

1. Codd, E.F. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* **1970**, *13*, 377–387. [\[CrossRef\]](#)
2. Selinger, P.G.; Astrahan, M.M.; Chamberlin, D.D.; Lorie, R.A.; Price, T.G. Access Path Selection in a Relational Database Management System. In Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, Boston, MA, USA, 30 May–1 June 1979.
3. Graefe, G. Query Evaluation Techniques for Large Databases. *ACM Comput. Surv.* **1993**, *25*, 73–170. [\[CrossRef\]](#)
4. Chaudhuri, S. An Overview of Query Optimization in Relational Systems. In Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Seattle, WA, USA, 1–3 June 1998.
5. Ioannidis, Y.E. Query Optimization. *ACM Comput. Surv.* **1996**, *28*, 121–123. [\[CrossRef\]](#)
6. Graefe, G.; McKenna, W.J. The Volcano Optimizer Generator: Extensibility and Efficient Search. In Proceedings of the Ninth International Conference on Data Engineering, Vienna, Austria, 19–23 April 1993; pp. 209–218. [\[CrossRef\]](#)
7. Begoli, E.; Camacho-Rodríguez, J.; Hyde, J.; Mior, M.J.; Lemire, D. Apache Calcite: A Foundational Framework for Optimized Query Processing over Heterogeneous Data Sources. In Proceedings of the 2018 International Conference on Management of Data (SIGMOD), Houston, TX, USA, 10–15 June 2018; pp. 221–230. [\[CrossRef\]](#)
8. Wang, Z.; Zhou, Z.; Yang, Y.; Ding, H.; Hu, G.; Ding, D.; Tang, C.; Chen, H.; Li, J. WeTune: Automatic Discovery and Verification of Query Rewrite Rules. In Proceedings of the 2022 International Conference on Management of Data (SIGMOD), Philadelphia, PA, USA, 12–17 June 2022. [\[CrossRef\]](#)
9. Zhou, X.; Li, G.; Wu, J.; Liu, J.; Sun, Z.; Zhang, X. A Learned Query Rewrite System. *Proc. VLDB Endow.* **2023**, *16*, 4110–4113. [\[CrossRef\]](#)
10. Dong, R.; Liu, J.; Zhu, Y.; Yan, C.; Mozafari, B.; Wang, X. SlabCity: Whole-Query Optimization Using Program Synthesis. *Proc. VLDB Endow.* **2023**, *16*, 3151–3164. [\[CrossRef\]](#)
11. Bai, Q.; Alsudais, S.; Li, C. QueryBooster: Improving SQL Performance Using Middleware Services for Human-Centered Query Rewriting. *Proc. VLDB Endow.* **2023**, *16*, 2911–2924. [\[CrossRef\]](#)
12. Liu, J.; Mozafari, B. GenRewrite: Query Rewriting via Large Language Models. *Proc. ACM Manag. Data* **2026**, *4*, 70. [\[CrossRef\]](#)
13. Dharwada, S.; Devrani, H.; Haritsa, J.R.; Doraiswamy, H. LITHE: A Query Rewrite Advisor using LLMs. In Proceedings of the 29th International Conference on Extending Database Technology (EDBT 2026), Tampere, Finland, 24–27 March 2026; pp. 233–246. [\[CrossRef\]](#)
14. Song, Y.; Yan, H.; Lao, J.; Wang, Y.; Li, Y.; Zhou, Y.; Wang, J.; Tang, M. QUITE: A Query Rewrite System Beyond Rules via LLM Agents. *arXiv* **2026**, arXiv:2506.07675v3.
15. Sun, Z.; Zhou, X.; Li, G.; Yu, X.; Feng, J.; Zhang, Y. R-Bot: An LLM-based Query Rewrite System. *Proc. VLDB Endow.* **2025**, *18*, 5031–5044. [\[CrossRef\]](#)
16. Xu, D.; Cui, Y.; Shi, W.; Ma, Q.; Guo, H.; Li, J.; Zhao, Y.; Zhang, R.; Di, S.; Zhu, J.; et al. E3-Rewrite: Learning to Rewrite SQL for Executability, Equivalence, and Efficiency. *arXiv* **2025**, arXiv:2508.09023v1.
17. Stillger, M.; Lohman, G.M.; Markl, V.; Kandil, M. LEO—DB2’s LEarning Optimizer. In Proceedings of the 27th International Conference on Very Large Data Bases (VLDB), Roma, Italy, 11–14 September 2001; pp. 19–28.
18. Marcus, R.; Negi, P.; Mao, H.; Tatbul, N.; Alizadeh, M.; Kraska, T. Bao: Making Learned Query Optimization Practical. In Proceedings of the 2021 International Conference on Management of Data (SIGMOD), Xi’an, China, 20–25 June 2021; pp. 1275–1288. [\[CrossRef\]](#)
19. Wilcoxon, F. Individual Comparisons by Ranking Methods. *Biom. Bull.* **1945**, *1*, 80–83. [\[CrossRef\]](#)
20. PostgreSQL Global Development Group. PostgreSQL Documentation. Available online: <https://www.postgresql.org/docs/> (accessed on 27 May 2026).
21. PostgreSQL Global Development Group. PostgreSQL Documentation: EXPLAIN. Available online: <https://www.postgresql.org/docs/current/sql-explain.html> (accessed on 27 May 2026).
22. PostgreSQL Global Development Group. PostgreSQL Documentation: pg_stat_statements. Available online: <https://www.postgresql.org/docs/current/pgstatstatements.html> (accessed on 27 May 2026).
23. Ollama. Ollama Documentation. Available online: <https://ollama.com/> (accessed on 27 May 2026).
24. Transaction Processing Performance Council. TPC-H Benchmark Specification. Available online: <https://www.tpc.org/tpch/> (accessed on 27 May 2026).

25. Leis, V.; Gubichev, A.; Mirchev, A.; Boncz, P.; Kemper, A.; Neumann, T. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* **2015**, *9*, 204–215. [[CrossRef](#)]
26. ForouzeshNejad, A.A.; Arabikhan, F.; Taheri, R. The Role of Explainable AI (XAI) in Enhancing the Security of Machine Learning Systems Against Adversarial Attacks. In *Adversarial Example Detection and Mitigation Using Machine Learning*; Nowroozi, E., Taheri, R., Cordeiro, L., Eds.; Springer: Cham, Switzerland, 2026. [[CrossRef](#)] [[PubMed](#)]

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