QueryTorque

VLDB Competitive Positioning & Game Plan

Reasoning-First, Cross-Engine SQL Optimization

Paper Thesis (One Sentence)

*"Existing AI query optimizers rely on iterative search (LITHE), static retrieval (R-Bot), or physical plan steering (LLM-QO). QueryTorque is the first system that **reasons from first principles** with a **structured knowledge framework**, and generalizes across database engines without retraining or retrieval."*

Two pillars. One clean contribution. No scope creep.

Pillar 1: Reasoning-First (No RL)

The architectural claim: SQL rewrites should be **reasoned**, not **searched**. QueryTorque produces a correct rewrite before it touches the database — no trial-and-error, no feedback loops, no dependence on unreliable cost signals.

Why this matters to reviewers

- **LITHE's loop is implicit RL with a broken reward signal.** Cost estimators can be off by orders of magnitude with skewed data. Optimizing against unreliable cost signals is not reasoning.
- **LLM-QO depends on planner feedback.** Without hint injection support (e.g., DuckDB, Snowflake), plan steering is not applicable. QueryTorque's reasoning is self-contained.
- **Token and time efficiency.** If LITHE takes 15 rounds to converge, QueryTorque takes 1 reasoning pass. Quantify this: tokens consumed, wall-clock overhead, DB round-trips.
- **Interpretability.** When reasoning fails, you can explain why. When an RL loop converges on a bad rewrite, nobody knows why. Trustworthiness matters in production DB systems.

Pillar 2: Cross-Engine Generalization

The experimental claim: QueryTorque works across engines **without retraining, corpus swapping, or hint injection.** Every competitor is over-fitted to Postgres.

The DuckDB Experiment

Run R-Bot against DuckDB (TPC-DS 1TB). Its retrieval corpus is full of Postgres-specific advice (e.g., `SET enable_nestloop=off`) which does not apply to DuckDB and may degrade performance. LLM-QO's hint injection is not supported in DuckDB.

Engine	QueryTorque	R-Bot	Outcome
Postgres (TPC-DS)	**~15x speedup**	~5x speedup	**QueryTorque wins 3x over**
DuckDB (TPC-DS)	**~20x speedup**	**Fail / ~1x**	**R-Bot cannot generalize**

Narrative: *"Existing methods are engine-overfitted. QueryTorque uses logical reasoning with a portable knowledge framework, making it the first cross-engine AI optimizer."*

VLDB reviewers value generalization over raw speed. Winning on both is the ideal position, but if forced to choose, generalization is the stronger narrative.

The Knowledge Framework (Curated, Not Auto-Generated)

Frame the knowledge base as a **structured expert rule library**. Describe the format (Gap Profiles), show examples, explain how a DBA extends it for a new engine. This gives reviewers reproducibility, correctness guarantees, and a clear extensibility story — without any hallucination concerns.

Aspect	R-Bot (Static Retrieval)	QueryTorque (Curated Framework)
Source	Scraped from manuals/forums	Expert-authored, validated rules
Correctness	Unverified (forum advice varies)	Human-validated, deterministic
Portability	Engine-locked (Postgres corpus)	New engine = new rule set (2hrs work)
Extensibility	Requires new corpus per engine	Structured format, DBA-extensible
Reproducibility	Depends on retrieval quality	Format published, rules inspectable

Key point: R-Bot got into VLDB '24 with scraped forum advice. A curated, validated knowledge framework is strictly better. The contribution is the reasoning architecture and how it interfaces with knowledge — not where the knowledge comes from.

IP protection: Publish the format and examples. The specific rules are operational advantage, not withheld science. No reviewer can object to this — it's like a trained model vs. training data.

Benchmark Strategy: DSB Over TPC-DS

DSB (Decision Support Benchmark) is the right choice. Here's how to frame it:

Why TPC-DS understates your advantage

Standard TPC-DS assumes column independence (no correlations). The Postgres optimizer handles this reasonably well, making it hard to show dramatic wins. The competition looks better than it actually is on TPC-DS.

Why DSB reveals the real gap

DSB introduces data skew and correlations (e.g., customers in 'CA' buy more 'Winter Coats'). This is where standard optimizers — and systems that trust them — break down:

- Cost estimators assume uniform distribution and produce wildly wrong estimates on skewed data.
- LITHE's iterative loop optimizes against these inaccurate estimates, compounding the error.
- R-Bot's static rules don't account for data-dependent patterns.
- QueryTorque's reasoning can spot the skew and choose rewrites accordingly.

Action: Highlight results on **DSB queries 21, 36, and 78** (notorious skew queries). Briefly explain the skew pattern for each so reviewers see it's principled, not cherry-picked.

Competitor Landscape

For reference: each competitor's approach, limitations, and benchmark claims.

System	Approach	Limitation	Their Benchmark
LITHE EDBT '26	**Trial-and-error loop.** Generates rewrites, checks syntax, checks optimizer cost, iterates.	Wastes tokens in generate-check-fix cycles. Trusts the DB cost estimator as reward signal — but cost estimates are unreliable, especially with skewed data. No reasoning about *why* a rewrite works.	**13.2x GM speedup** *(on self-selected "Hefty" queries only)*
R-Bot VLDB '24	**RAG retrieval.** Fetches static rules from manuals and forums, applies them.	Cannot generate new optimization logic — limited to what exists in the corpus. Corpus is engine-specific: Postgres tuning advice does not transfer to DuckDB and may degrade performance.	**~5x speedup** *(Postgres-only evaluation)*
LLM-QO SIGMOD '25	**Plan steering.** Forces the DB to use specific join orders via hint injection.	Operates at the physical plan level — steers join orders via hint injection rather than rewriting SQL. Not applicable on engines without hint support (DuckDB, Snowflake, BigQuery).	**68% latency reduction** *(DSB on Postgres only)*

Positioning Matrix

How QueryTorque differs across every evaluation dimension. Each row is a potential reviewer question.

Dimension	LITHE	R-Bot	LLM-QO	QueryTorque
Reasoning	RL-style loop	None (retrieval)	Planner feedback	**First-class reasoning**
Engine Portability	Single engine	Corpus-locked	Hint-dependent	**Cross-engine**
Optimization Level	Syntactic rewrite	Rule application	Physical plan only	**Logical SQL rewrite**
Knowledge Source	None	Static manuals	None	**Curated expert rules**
Correctness	Post-hoc check	Unvalidated	Execution-based	**Validated before exec**

Anticipated Reviewer Questions

1. ***"What if reasoning gets it wrong? At least RL self-corrects."***

   Answer: When reasoning fails, the failure is interpretable and diagnosable. When an iterative loop converges on a suboptimal rewrite guided by inaccurate cost estimates, the failure is silent. We validate rewrites for semantic equivalence before execution.

2. ***"How much overhead does the reasoning add?"***

   Answer: Report wall-clock time, tokens per query, and DB round-trips. Pre-empt this by including an overhead analysis table in the paper.

3. ***"Why not combine plan steering (LLM-QO) with your SQL rewriting?"***

   Answer: They're complementary. Our rewriting is engine-portable and operates at the logical level. Plan steering is engine-specific. A combined system is interesting future work, but our contribution is the reasoning-first rewriting layer, which is the missing piece.

4. ***"The knowledge base is hand-curated. Does this scale?"***

   Answer: We wrote DuckDB rules in ~2 hours and achieved Xx speedup. The format is structured and DBA-extensible. Automated knowledge discovery is promising future work, but curated rules provide correctness guarantees that auto-generated rules currently cannot.

5. ***"Where does QueryTorque fail?"***

   Answer: Present failure cases honestly. Queries where reasoning doesn't find an improvement, or where the rewrite is correct but not faster. Reviewers respect this and it preempts the most damaging critique.

Execution Checklist

Concrete actions to make this positioning airtight.

Action	Detail
Run the DuckDB experiment	R-Bot vs QueryTorque on TPC-DS 1TB, DuckDB. Prove the generalization gap is real and dramatic.
Highlight DSB skew queries	Focus on Query 21, 36, 78 — notorious for data skew. Show reasoning catches what cost estimators miss. Explain the skew pattern briefly to avoid cherry-picking optics.
Benchmark on LITHE's gaps	LITHE cherry-picked "Hefty" queries. Run on the full suite. Match or beat on Hefty, dominate elsewhere. Have an answer ready for direct comparison.
Measure overhead	Report wall-clock time for QueryTorque reasoning, token cost per query, number of DB round-trips. Competitors will ask. Pre-empt the question.
Document the knowledge format	Publish the Gap Profile structure and examples. Show how a DBA extends it. Reviewers need to see this is reproducible and not a black box.
Own your failure cases	Identify queries where QueryTorque doesn't improve performance. Present them honestly. Reviewers respect this and it preempts the most damaging critique.

Write DuckDB rules (2 hrs)	Curate a DuckDB-specific rule set to prove the portability story is practical, not theoretical. The speed of porting is itself a result.

Paper Sequencing

Paper 1 (this submission): QueryTorque with curated knowledge, reasoning-first architecture, cross-engine results on Postgres + DuckDB. Establishes the architecture and proves the approach.

Paper 2 (follow-up): Autonomous Knowledge Discovery for Cross-Engine Query Optimization. Full paper dedicated to proving generated knowledge is correct and useful, building on the established system.

This is how strong research groups sequence claims. One clean contribution per paper. Don't try to solve everything at once.

Two pillars. Clean story. Defensible at every point.