Zero-Shot, Few-Shot, or Chain-of-Thought? An Empirical Study of Prompting Strategies for LLM-Guided SQL Query Optimization

Daniel Costa
Federal Institute of Education, Science and Technology of Paraíba – IFPB
João Pessoa, Brazil
cunha.daniel@academico.ifpb.edu.br

Danyllo Albuquerque
Federal Institute of Education, Science and Technology of Paraíba – IFPB
João Pessoa, Brazil
danyllo.albuquerque@ifpb.edu.br

Bruno Moreno
Federal Institute of Education, Science and Technology of Paraíba – IFPB
João Pessoa, Brazil
bruno.moreno@ifpb.edu.br

Emanuel Dantas
Federal Institute of Education, Science and Technology of Pernambuco – IFPE
Jaboatão, Brazil
emanuel.filho@jaboatao.ifpe.edu.br

Abstract

Query optimization remains a central problem in database systems, and large language models (LLMs) have shown promise as advisors on code and SQL tasks. A practical question is open: when an LLM is given rich DBMS context, how much does the prompting strategy affect whether the model produces a semantically valid optimization, and whether that optimization is worth applying? We address this question through a controlled empirical study. We evaluate three prompting strategies – zero-shot, few-shot, and chain-of-thought (CoT) – on three different LLMs: Llama 3.3, GPT-4.1-mini, and the reasoning-tuned DeepSeek-R1, across the 22 TPC-H queries on PostgreSQL (198 trials in total). Each prompt is built from the same DBMS context (execution plans, schema, indexes, workload history, slow-query logs, tuning guides, and curated examples) so that the strategy is the only independent variable. We find four results. First, Llama 3.3 is strikingly robust in the zero-shot setting, producing a valid optimization in 95.5% of trials. Second, GPT-4.1-mini requires few-shot prompting to reach the same level, rising from 36.4% to 95.5% validity. Third, chain-of-thought roughly halves validity for both general-purpose models, and the failure is in the content, not in the output format: across the three models, 91% of CoT failures are well-formed JSON with a rewrite or index that the semantic-equivalence check rejects. Fourth, the reasoning-tuned DeepSeek-R1 behaves differently: its validity is strategy-stable (68%–86%) rather than collapsing, and its median speedup on valid trials is the highest of any model-strategy pair; its zero-shot failures are predominantly incorrect query rewrites, consistent with a pattern we call *reasoning amplification*, where, without worked examples, reasoning-tuned models drift toward textbook-looking optimizations that are DBMS-specific wrong. Strategy choice dominates model choice for validity; few-shot is the safe default, and examples act as necessary anchors for reasoning-tuned models.

Keywords

Query optimization, Large language models, Prompting strategies, Empirical study, TPC-H, PostgreSQL

1 Introduction

Query optimization is a long-standing challenge in database systems. The optimizer must choose access paths, join orders, and physical structures that minimize execution cost, often under incomplete statistics and with little insight into workload intent. Human database administrators (DBAs) still play an important role, because decisions such as creating a secondary index or rewriting a subquery typically require knowledge that does not live in the planner: deployment history, workload shape, and written tuning guides.

Large language models (LLMs) have recently shown strong performance on code and SQL tasks, and a small but growing literature uses them as advisors for database tuning and index selection [3, 8, 11]. The open question is practical: when an LLM is given rich DBMS context, how much does the *prompting strategy* determine whether the model produces a semantically valid optimization and whether that optimization is worth applying? The answer matters, because the same model can behave very differently under zero-shot, few-shot, and chain-of-thought prompting, and a deployment decision has to pick one.

Contribution. We contribute an analysis of three prompting strategies – zero-shot, few-shot, and chain-of-thought (CoT) – for SQL query optimization, using a common bundle of DBMS context across all prompts. The analysis is carried out through a reproducible benchmark that measures, for each (model, strategy, query) trial, whether the LLM's suggestion is semantically equivalent to the original query and, when valid, how it compares against a baseline execution. The benchmark is the vehicle; the comparative study is the contribution.

We evaluate three LLMs (Llama 3.3, GPT-4.1-mini, and DeepSeek-R1) on the 22 TPC-H queries executed on PostgreSQL, for a total of $3 \times 3 \times 22 = 198$ trials. We report four findings. First, Llama 3.3 produces a valid optimization in 95.5% of zero-shot trials, without any in-context examples. Second, GPT-4.1-mini needs few-shot prompting to reach the same level, moving from 36.4% to 95.5% validity. Third, chain-of-thought prompting hurts the two non-reasoning models, roughly halving their rate of valid suggestions relative to the best strategy, with 91% of CoT failures being well-formed JSON whose content the semantic-equivalence check rejects.

CoT degrades the SQL the model proposes while leaving the JSON schema intact. Fourth, the reasoning-tuned DeepSeek-R1 behaves differently from the other two: its validity is the most strategy-stable of the three (68%–86% across all prompts), while its median speedup on valid trials is the highest of any model-strategy pair in the study; its zero-shot failures are predominantly incorrect query *rewrites* rather than silent refusals. Without worked examples to anchor them, reasoning-tuned models appear to amplify both good and bad ideas. We argue that strategy choice dominates model choice for validity, and that few-shot is the safe default when the output must be machine-checkable.

Roadmap. Section 2 reviews related work on LLMs for databases, learned query optimization, prompting strategies, and benchmark design. Section 3 describes the research methodology in four steps: benchmark and context setup, prompt-based optimization generation, trial execution and validation, and results aggregation. Section 4 reports the main results, including validity and speedup tables for the nine (model, strategy) combinations. Section 5 discusses the main implications for practitioners. Section 6 analyses threats to validity, and Section 7 closes with final remarks and future work.

2 Related Work

LLMs for databases. Recent work uses LLMs to assist specific database tasks. GPTuner [11] recommends knob configurations for DBMSs by consuming tuning manuals as context. λ-Tune [8] adapts system tuning to workloads with LLM guidance. DITune [9] applies reinforcement learning to automated index selection, and recent work distills LLMs for SQL generation [12]. Cong et al. [3] survey machine learning for databases and situate these efforts in a wider learning-based agenda. These efforts show that LLMs can perform specialized database tasks, but they each fix a single task-and-strategy combination and do not isolate the effect of prompting on quality.

Learned and LLM-era query optimization. Query optimization has been a target of learning-based techniques for years. Tsesmelis and Simitsis [18] review how optimizer internals have evolved under the influence of learned components. Gao et al. [7] propose automatic index selection with a learned cost estimator. Our study is complementary: rather than replacing the optimizer, we treat the LLM as an external advisor that produces either a rewrite or an index suggestion that the DBMS then evaluates.

Benchmarks and workload generation. SQLStorm [16] argues that classical benchmarks are insufficient for the LLM era and proposes new workload-generation methods. Zheng et al. [19] use LLMs to generate cross-model analytics workloads. TPC-H [17] remains the most widely used analytical workload. BenchBase and its predecessors [1, 4, 6] provide a flexible harness for benchmarking relational databases; we adopt its TPC-H data generator to keep our setup reproducible.

Research gap. Existing work either (i) commits to one prompting strategy and one model while varying the database task, or (ii) varies the model architecture but not the prompt. None of the cited studies measures how the *same* optimization task behaves across prompting strategies when the DBMS context is held constant. The present paper closes that gap: we fix the context bundle, fix the task (produce a valid query rewrite or index suggestion in JSON), and vary only the prompting strategy across three production-grade LLMs.

3 Research Methodology

This study investigates whether prompting strategy, rather than model choice, is the dominant factor determining the quality of LLM-guided SQL query optimizations. Our goal is to measure, under a common bundle of DBMS context, how often each of the three widely used strategies produces a semantically valid optimization, and how much speedup the valid optimizations yield on a well-known analytical workload.

To guide this investigation, we formulated three Research Questions (RQs):

- **RQ1**: How often does each prompting strategy produce a semantically valid optimization across the 22 TPC-H queries?
- **RQ2**: Which (model, strategy) combinations are the most reliable, and how does strategy choice interact with model choice?
- **RQ3**: When the suggestion is valid, do the different strategies produce meaningfully different speedups over the baseline?

The rationale behind these RQs is as follows.

RQ1 isolates the effect of prompting on whether the LLM returns a proposal that the DBMS is willing to execute in place of the original query. Validity is a more demanding criterion than mere JSON-schema compliance: the proposed rewrite must produce the same result as the original query on the target database, and any suggested index must be a well-formed DDL statement that actually applies. This question matters because a fast-but-wrong optimization is worse than no optimization at all.

RQ2 asks whether strategies have a uniform effect across models, or whether each model has a strategy profile of its own. For deployment, this decides whether a single recommended strategy can be picked for all models in a pool, or whether strategy must be tuned per model.

RQ3 shifts from validity to impact: *given* a valid optimization, does the choice of strategy change how much faster the query runs? The answer is relevant because the practical return on LLM-guided optimization is driven by the product of validity rate and median speedup on valid suggestions; the two can trade off.

To address these RQs, we designed a four-step methodology that constructs a reproducible benchmark, generates optimizations under three prompting strategies across three LLMs, validates each suggestion semantically, and aggregates the results. The overall workflow is shown in Figure 1. The remainder of this section presents the four steps in detail.

3.1 Step 1: Benchmark and Context Setup

Workload. We use the 22 TPC-H queries [17] at scale factor SF = 1, generated with BenchBase [1]. TPC-H is an analytical decision-support workload with known complexity and well-characterised plans; it is not a test of the LLM's knowledge of a specific business domain but of its ability to reason about SQL, plans, and indexes. The database is PostgreSQL 17 (Docker image `postgres:17`) loaded with only the primary-key and foreign-key constraints declared in the TPC-H specification; no secondary indexes are created at

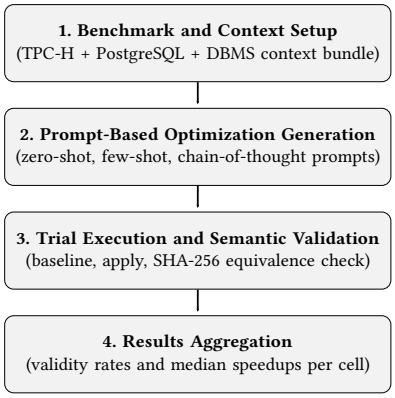

Figure 1: Methodological workflow.

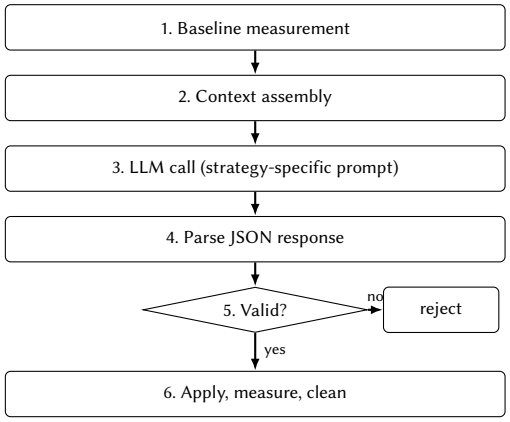

Figure 2: End-to-end pipeline for one (model, strategy, query) trial. The DBMS context assembled in step 2 is identical across the three prompting strategies; the strategy varies only in how that context is framed for the LLM in step 3. Validation (step 5) uses cardinality plus a SHA-256 digest over an ordered, limited projection of the query output.

baseline, making the baseline deliberately naïve and leaving ample room for optimizations to show an effect.

DBMS context bundle. Every prompt, regardless of strategy, is built from the same seven context inputs, collected from the live DBMS and from static project artifacts:

(1) **Execution plans** from `EXPLAIN (ANALYZE, BUFFERS, FORMAT JSON)` on the target query;
(2) **Schema and constraints** of all referenced tables, including primary and foreign keys;
(3) **Indexes and database configuration**, including the current `pg_index` contents and a curated subset of planner-relevant `pg_settings`;
(4) **Workload history**, i.e. a short list of other queries seen in the session;
(5) **Slow-query logs**, summarizing queries above a latency threshold;
(6) **DBMS documentation and tuning guides** excerpts, focused on indexing and on the operators appearing in the plan;
(7) **Internal examples** of *query → rewrite* or *query → index* patterns that worked on similar TPC-H queries.

Fixing the context across strategies is an explicit design decision. A DBMS-agnostic rewrite is often not a good rewrite: whether an index helps, whether a subquery can be unnested, and whether a given predicate is sargable all depend on the concrete schema, the existing indexes, the current plan, and the DBMS's configuration. Providing every prompt with the same bundle of DBMS context, and varying only the prompting strategy, isolates the strategy's effect on suggestion quality from the effect of the model's access to context.

3.2 Step 2: Prompt-Based Optimization Generation

We evaluate three prompting strategies. All three ask the model to return the same JSON-serialisable optimization object: either a semantically equivalent rewrite of the target query, or a list of candidate indexes, or both. The only independent variable is how the request is framed.

Zero-shot. The model receives the DBMS context and a task description asking it to return the JSON object. No worked examples are included.

Few-shot. Following Brown et al. [2], the prompt includes a small fixed set of worked examples (query + context → expected JSON output), selected to cover both rewrite-based and index-based optimizations, before the target query is presented.

Chain-of-thought. Following Kojima et al. [10], the prompt appends a trigger phrase that elicits step-by-step reasoning before the final JSON object. The schema still requires a single well-formed JSON answer at the end.

We evaluate three LLMs: Llama 3.3 [13], GPT-4.1-mini [14], and DeepSeek-R1 [5], all accessed through OpenRouter [15]. Temperature is set to 0 to minimise run-to-run variance; any residual nondeterminism is discussed as a threat to validity (Section 6). The full set of prompts used in the study is available in the supplementary repository (see Artifact Availability).

3.3 Step 3: Trial Execution and Semantic Validation

For each (model, strategy, query) triple, the benchmark executes a six-stage trial shown in Figure 2. The trial pairs baseline measurement with an optimized measurement conditional on validation: the optimized configuration is only timed if the suggestion first passes a semantic-equivalence check.

Semantic-equivalence check. An optimization is accepted only if its output matches the original query semantically. The benchmark compares cardinalities and, on a deterministic projection obtained via `ORDER BY` with a `LIMIT`, a SHA-256 digest of the rows. Any mismatch, parser failure, or timeout marks the trial invalid and no optimized timing is recorded.

Table 1: Semantic validity rate (%) per model and strategy. Values are the percentage of the 22 TPC-H trials in which the LLM's suggestion passed semantic-equivalence validation.

Model	Zero-shot	Few-shot	CoT
Llama 3.3	95.5 (21/22)	90.9 (20/22)	54.5 (12/22)
GPT-4.1-mini	36.4 (8/22)	95.5 (21/22)	31.8 (7/22)
DeepSeek-R1	68.2 (15/22)	86.4 (19/22)	63.6 (14/22)

Execution parameters. We execute $k = 10$ measurement runs per timing point, and discard the first 3 runs as warmup. All PostgreSQL state created by a trial (e.g. temporary indexes) is dropped at the end of the trial so that subsequent trials see the same naïve baseline.

3.4 Step 4: Results Aggregation

For each of the $3 \times 3 \times 22 = 198$ trials the benchmark records the validity flag, the LLM's JSON response, the baseline timing, and (when valid) the optimized timing. We aggregate these into two per-cell summaries. The *validity rate* is the fraction of the 22 TPC-H queries for which the (model, strategy) cell produced a valid optimization. The *median speedup* is computed across the valid trials. We report medians rather than means because absolute speedups span several orders of magnitude under the naïve baseline and are dominated by a handful of extreme outliers; medians make the relative ordering across cells interpretable.

4 Results and Discussion

This section presents the main results of the study, organised around the three RQs defined in Section 3. We executed all 198 trials on the setup described in Section 3.1 (PostgreSQL 17, TPC-H SF=1) using the three LLMs and three prompting strategies of Section 3.2. We first quantify validity rates per (model, strategy) cell (RQ1), analyse how strategy and model interact across cells (RQ2), and then discuss the speedup magnitudes on the subset of valid trials (RQ3).

4.1 RQ1: Validity Rate Across Strategies

Table 1 reports, for every (model, strategy) cell, the number of queries for which the LLM produced a valid optimization, out of 22.

Three observations answer RQ1.

Llama 3.3 is robust in zero-shot. Without any in-context examples, Llama 3.3 produces a valid optimization for 21 of the 22 queries. This is the most surprising result of the study: for this task, the model carries enough prior knowledge of SQL and indexing to answer correctly when simply shown the schema, the plan, and the current indexes.

GPT-4.1-mini needs few-shot. In contrast, GPT-4.1-mini is valid on only 8 of 22 queries in zero-shot, rising to 21 of 22 under few-shot prompting. Few-shot does more than add marginal accuracy here: it is the difference between unusable and reliable. Few-shot is also the only strategy under which all three models clear 85% validity. For any pipeline that consumes the LLM's output with a checker, few-shot is the safe default.

Chain-of-thought hurts the non-reasoning models, and it does so through content, not format. For both Llama 3.3 and GPT-4.1-mini, CoT roughly halves the rate of valid suggestions compared to the model's best strategy (Llama 3.3: 95.5% → 54.5%; GPT-4.1-mini: 95.5% → 31.8%). The natural hypothesis is that CoT's free-form reasoning breaks the JSON schema the benchmark expects; the data do not support it. Of the 33 CoT failures across the three models (10 for Llama 3.3, 15 for GPT-4.1-mini, 8 for DeepSeek-R1), 30 (91%) were well-formed JSON responses containing a non-empty rewrite or index that the semantic-equivalence check then rejected; only 3 were empty or malformed. CoT therefore does not break schema compliance; it appears to make the model commit to an incorrect SQL idea during the reasoning trace and then emit that idea faithfully in JSON. This is the same failure mode we document for DeepSeek-R1 in zero-shot (Section 4.2): the rewrites themselves are wrong. CoT's problem is substantive. The chain produces confident proposals that happen to be wrong for the concrete schema or plan, and a stricter output parser would not recover them.

4.2 RQ2: Strategy × Model Interaction

RQ2 asks whether strategies have a uniform effect across models. They do not. The validity rows of Table 1 have qualitatively different shapes: Llama 3.3 is high-high-collapsed, GPT-4.1-mini is collapsed-high-collapsed, and DeepSeek-R1 is moderate-high-moderate. A practitioner cannot pick a strategy independently of the model.

The DeepSeek-R1 row is worth examining closely because it motivates a pattern we label *reasoning amplification*. DeepSeek-R1's validity is the flattest across strategies in the whole study (zero-shot 68.2%, few-shot 86.4%, CoT 63.6%): unlike the general-purpose models, it does not collapse on any strategy, but it also never reaches the 95.5% ceiling that Llama 3.3 and GPT-4.1-mini touch on their best strategy. Its failure mode is informative: in 6 of the 7 zero-shot failures, DeepSeek-R1 returned a query *rewrite* that was then rejected by the semantic-equivalence check. The model engaged with the task and produced real SQL; the SQL happened to be wrong for the target schema. Llama 3.3, by contrast, tends to fall back on safer index suggestions and rarely attempts rewrites in the same situations. Combined with DeepSeek-R1's high median speedup when it is valid (Section 4.3), this suggests that reasoning-tuned models amplify both successes and errors under zero-shot prompting. Without worked examples to anchor their reasoning, they drift toward optimizations that look textbook-correct but are specific to a different DBMS, schema variant, or optimizer configuration. Few-shot prompting closes roughly half of that validity gap for DeepSeek-R1 (68% → 86%), which is consistent with the hypothesis: examples act as anchors that constrain the reasoning to the current setting.

Answering RQ2: strategy choice dominates model choice for validity, few-shot is the strategy that generalises across all three tested models, and reasoning-tuned models need explicit anchors (examples) to stay within the task as specified.

4.3 RQ3: Speedup Magnitude Across Strategies

On the subset of valid trials, absolute speedups over the baseline are very large, commonly spanning several orders of magnitude on

Table 2: Median speedup (%) across valid trials per model and strategy. Dashes mark cells with no valid trials. Absolute magnitudes are inflated by the deliberately naïve, unindexed baseline; the informative signal is the relative ordering across models and strategies.

Model	Zero-shot	Few-shot	CoT
Llama 3.3	3	10	1
GPT-4.1-mini	18	30	25
DeepSeek-R1	48	44	50

individual queries. This is not a property of the LLM: the baseline is PostgreSQL on TPC-H SF=1 with *no secondary indexes*, so any reasonable index suggestion reduces certain scans from $O(N)$ to $O(\log N)$. We therefore report the *median* speedup across valid trials, which is far more stable than the mean and less dominated by a handful of extreme outliers, and treat the absolute magnitudes as a sanity check rather than a load-bearing claim.

Two patterns stand out in Table 2. First, the relative ordering of models by median speedup is DeepSeek-R1 > GPT-4.1-mini > Llama 3.3 across every strategy, and it is the *reverse* of the validity ordering for the two non-reasoning models: Llama 3.3 is most reliable but also most conservative, proposing optimizations that pass validation at a high rate but move the needle the least. Second, DeepSeek-R1's median speedup is essentially flat across strategies (48, 44, 50) and larger than any cell of the other two models. This aligns with the observation in Section 4.1: DeepSeek-R1 produces more aggressive optimizations, and when those survive the semantic-equivalence check they tend to be more impactful. The trade-off is the validity gap with Llama 3.3.

This addresses RQ3: the three strategies do not converge on the same suggestion, and models differ systematically in how aggressively they optimize when valid. Our naïve baseline still inflates absolute speedups, so the strongest claims remain about validity and relative ordering, not about absolute magnitudes.

5 Main Implications

The results carry several practical implications for teams that use LLMs as query-optimization advisors.

Few-shot is the safe default. Across all three models and the nine (model, strategy) cells, few-shot is the only strategy that keeps every model above 85% validity on TPC-H. If a deployment must pick a single strategy without tuning it per model, few-shot minimises the regret.

Chain-of-thought degrades the content, not the format. For both Llama 3.3 and GPT-4.1-mini, adding a chain-of-thought trigger roughly halves validity relative to the model's best strategy, and the failures are not parse or schema errors: 91% of CoT failures across the three models are well-formed JSON with a non-empty rewrite or index that fails the semantic-equivalence check (Section 4.1). Wrapping the CoT output in a stricter schema-enforcing decoder would therefore not recover the lost validity; the suggestions themselves are wrong for the concrete schema and plan. Teams interested in keeping chain-of-thought's reasoning benefits should pair it with a validator that re-grounds the reasoning in the current DBMS state (for example, by feeding `EXPLAIN` of the proposed rewrite back to the model) rather than with a tighter output parser.

Reasoning-tuned models need anchors. DeepSeek-R1's zero-shot failures are concentrated in incorrect query rewrites rather than refusals. When the answer has to respect the target DBMS's concrete schema, indexes, and configuration, a reasoning-tuned model left without examples tends to propose optimizations that are correct in a *general* sense but wrong for the *specific* environment. Shipping even a small set of worked examples (≤ 5 in our case) closes about half of that validity gap.

Reliability and impact can trade off. In our data, Llama 3.3 is the most reliable (highest validity) but its valid suggestions have the smallest median speedup; DeepSeek-R1 is less reliable but its valid suggestions deliver the largest median speedup in every strategy. Teams with a human reviewer in the loop may prefer DeepSeek-R1 with few-shot (high-impact proposals, reviewer catches bad rewrites); fully automated pipelines that commit suggestions without review may prefer Llama 3.3 with few-shot (low-risk but smaller wins).

DBMS context is the carrier, strategy is the lever. All three strategies share the same context bundle (execution plans, schema, indexes, configuration, history, slow-query logs, tuning guides, worked examples). Holding that context constant, the strategy alone moves validity by up to 59 percentage points for a single model (GPT-4.1-mini zero-shot to few-shot). In practice, the engineering investment should go first into assembling good context and second into picking the right prompting strategy for the chosen model.

6 Threats to Validity

Several factors limit the generality of our conclusions. *Construct:* a single DBMS (PostgreSQL) and a single scale factor (SF=1) were tested; other systems may expose different behaviours. *Baseline:* the PostgreSQL baseline has no secondary indexes, which inflates absolute speedups. *Models:* we evaluate three LLMs (two general-purpose and one reasoning-tuned); results may differ for other families and sizes, and the reasoning-amplification finding rests on a single reasoning-tuned model and should be revisited with additional ones (e.g. o1-class or Claude reasoning variants). *Runs:* we do not average across many repetitions per trial, and temperature zero does not fully eliminate non-determinism. *Workload:* TPC-H is synthetic and analytical; real workloads differ in shape.

7 Final Remarks

We presented a comparative study of three prompting strategies – zero-shot, few-shot, and chain-of-thought – for SQL query optimization with three production-grade LLMs (Llama 3.3, GPT-4.1-mini, DeepSeek-R1) on the 22 TPC-H queries over PostgreSQL, for a total of 198 trials. The study is carried out through a reproducible benchmark that holds the DBMS context constant across strategies. We found that Llama 3.3 is robust in the zero-shot setting at 95.5% validity, that GPT-4.1-mini requires few-shot to reach the same level, and that chain-of-thought roughly halves validity for both general-purpose models. The reasoning-tuned DeepSeek-R1 is strategy-stable but never reaches the validity ceiling of the general-purpose models; when valid, however, it delivers the largest median speedup of the three in every strategy. Its failures are predominantly

incorrect query rewrites rather than refusals, consistent with a pattern we call reasoning amplification: without worked examples, reasoning-tuned models drift toward textbook-looking optimizations that are DBMS-specific wrong. For this task, strategy choice dominates model choice on validity, few-shot is the safe default, and examples are a necessary anchor for reasoning-tuned models.

Future work has several directions. First, extend the evaluation to additional reasoning-tuned models (e.g. o1-class, Claude reasoning variants) to test whether the amplification finding generalizes. Second, evaluate additional DBMSs and larger scale factors. Third, explore hybrid strategies that combine few-shot with structured reasoning, to see whether CoT's format problem can be recovered without losing its reasoning benefits. Fourth, move beyond TPC-H to real analytical workloads, where DBMS context beyond the plan (workload history, slow-query logs) is more informative.

Artifact Availability

The benchmark, prompts, and scripts to reproduce the results are publicly available at https://github.com/danicunhac/llm-query-opt-benchmark. The repository is released under the MIT license and is archived in Software Heritage under the identifier `swh:1:rel:1699fee61d8c888e4902e9b0b38307506fef10b8`. The TPC-H dataset is generated locally at scale factor SF = 1 following the official specification [17]; the repository ships the loader scripts and the exact set of 22 TPC-H queries used in the experiments.

Acknowledgments

The authors thank the Federal Institute of Paraíba (IFPB) for institutional support.

References

[1] Dana Van Aken, Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2015. BenchPress: Dynamic Workload Control in the OLTP-Bench Testbed. In *Proceedings of ACM SIGMOD*. ACM, 1069–1073.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:2005.14165

[3] Gao Cong, Jingyi Yang, and Yue Zhao. 2024. Machine Learning for Databases: Foundations, Paradigms, and Open Problems. *Proceedings of the ACM on Management of Data (SIGMOD)* (2024). doi:10.1145/3626246.3654686

[4] Carlo Curino, Djellel Eddine Difallah, Andrew Pavlo, and Philippe Cudré-Mauroux. 2012. Benchmarking OLTP/Web Databases in the Cloud: The OLTP-Bench Framework. In *Proceedings of CloudDB*. 17–20.

[5] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. https://arxiv.org/abs/2501.12948. arXiv:2501.12948

[6] Djellel Eddine Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudré-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *Proceedings of the VLDB Endowment* 7, 4 (2013), 277–288.

[7] Jianling Gao, Nan Zhao, Ning Wang, Shuang Hao, and Haoyan Wu. 2022. Automatic Index Selection with Learned Cost Estimator. *Information Sciences* (2022).

[8] Victor Giannakouris and Immanuel Trummer. 2025. Demonstrating λ-Tune: Exploiting Large Language Models for Workload-Adaptive Database System Tuning. In *Proceedings of the ACM on Management of Data (SIGMOD)*. ACM. doi:10.1145/3626246.3654751

[9] Xiuqi Huang, Xiaofeng Gao, and Guihai Chen. 2024. DITune: A Reinforcement Learning Based Framework for Automated Database Index Selection. In *Proceedings of the IEEE International Conference (ICDE)*. IEEE.

[10] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large Language Models are Zero-Shot Reasoners. In *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:2205.11916

[11] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. 2025. GPTuner: An LLM-based Database Tuning System. *Proceedings of the ACM on Management of Data (SIGMOD)* (2025). doi:10.1145/3733620.3733641

[12] Nhat Le, Quan Ninh, Tung Le, and Huy Tien Nguyen. 2025. Distilling Large Language Models for Structured Query Language Generation. *Computers and Electrical Engineering* (2025).

[13] Meta AI. 2024. Llama 3.3 Model Card. https://ai.meta.com/llama/.

[14] OpenAI. 2025. GPT-4.1-mini. https://openai.com/index/gpt-4-1/.

[15] OpenRouter. [n. d.]. OpenRouter – Unified LLM API. https://openrouter.ai/.

[16] Tobias Schmidt, Viktor Leis, Peter Boncz, and Thomas Neumann. 2025. SQLStorm: Taking Database Benchmarking into the LLM Era. *Proceedings of the VLDB Endowment* 18, 11 (2025), 4144–4157.

[17] Transaction Processing Performance Council. [n. d.]. TPC-H Benchmark Specification. https://www.tpc.org/tpch/.

[18] Dimitris Tsesmelis and Alkis Simitsis. 2022. Database Optimizers in the Era of Learning. In *Proceedings of ICDE*. IEEE.

[19] Xiuwen Zheng, Arun Kumar, and Amarnath Gupta. 2024. Generating Cross-Model Analytics Workloads Using LLMs. In *Proceedings of DaMoN*. ACM. doi:10.1145/3627673.3679932