Enterprise Text-to-SQL: Are Agents Ready to Scale?

Tarık Buğra Akbaş[†] Kemal Tarık Nehrozoğlu[†] Pinar Karagoz[†] Nesime Tatbul[*]
[†]METU, Türkiye [*]MIT, USA
{tarik.akbas,tarik.nehrozoglu,pinark}@metu.edu.tr,tatbul@csail.mit.edu

ABSTRACT

The integration of agentic architectures has significantly advanced the state-of-the-art in text-to-SQL tasks, enabling complex reasoning and self-correction through multi-step, modular workflows. However, the performance gains achieved by these systems, such as those utilizing ReAct-style loops, hierarchical delegation, and multi-agent collaboration, often come at a substantial computational and monetary cost that remains insufficiently quantified. This paper presents a comprehensive cost analysis of leading agentic text-to-SQL frameworks, evaluating the trade-offs between execution accuracy and operational expenditure. By profiling token consumption, GPU energy, and end-to-end SQL generation latency across the Spider, BIRD, and BEAVER benchmarks, we provide a comparative landscape of agentic efficiency.

VLDB Workshop Reference Format:
Tarık Buğra Akbaş[†] Kemal Tarık Nehrozoğlu[†] Pinar Karagoz[†] Nesime Tatbul[*] . Enterprise Text-to-SQL: Are Agents Ready to Scale?. VLDB 2026 Workshop: Novel Optimizations for Visionary AI Systems (NOVAS).

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/TarikBugraAkbas/text2sql.

1 INTRODUCTION

Text-to-SQL, the task of translating a natural language question into an executable SQL query, has long been considered an important benchmark of practical natural language understanding. Its industrial relevance is clear: enabling non-expert users to query databases directly would unlock data access across enterprises, scientific institutions, and public services alike. With the emergence of large language models (LLMs), the field has experienced remarkable progress, with state-of-the-art systems now achieving execution accuracy above 90% on established benchmarks such as Spider [17].

This progress, however, has not come evenly. The leaderboard gains of recent years have been driven by increasingly elaborate pipelines: multi-step schema linking, query decomposition, self-correction loops, and multi-agent collaboration [13, 15]. Each additional stage improves accuracy but also multiplies the number of LLM calls, the tokens consumed, the latency incurred, and the energy drawn from the underlying hardware. The field has converged on accuracy as the dominant yardstick while treating cost as an afterthought.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

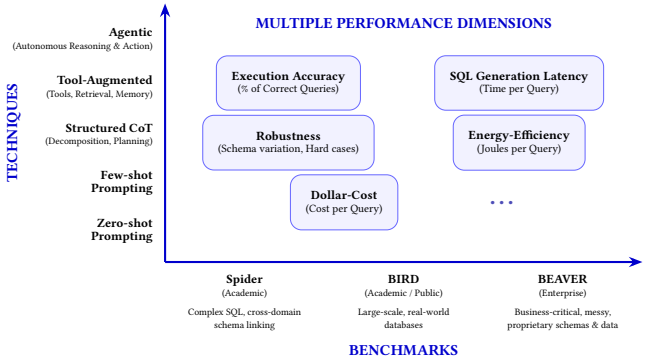

Figure 1: Overview of text-to-SQL technology landscape

Figure 1 illustrates the evolving landscape of text-to-SQL technology. As benchmarks transition from academic environments to enterprise-scale complexity, methodologies are increasingly shifting toward agentic architectures. Consequently, the primary evaluation metrics are expanding beyond mere accuracy to encompass a diverse range of cost and efficiency factors. For example, recent literature has begun evaluating reasoning latency alongside token boundaries across specialized domain schemas [2, 3, 18]. However, a systematic cross-benchmark analysis that profiles evolving architectures directly at the hardware-level energy boundary remains absent.

This paper aims to address this gap, by evaluating four representative text-to-SQL methods of increasing complexity: a one-shot baseline (*FewShot*), a similarity-driven few-shot selector (*DAIL-SQL* [4]), a four-stage decomposition pipeline (*DIN-SQL* [13]), and a three-agent collaborative framework (*MAC-SQL* [15]); across three benchmarks of distinct character: Spider [17] and BIRD [10] academic benchmarks, and the BEAVER [1] enterprise benchmark built based on private production data warehouses. Crucially, we evaluate cost dynamics from an infrastructure perspective. Rather than relying solely on estimated API pricing, we instrument queries using hardware-level analytics to record the actual GPU energy consumed per inference pass, mapping energy expenditure alongside token counts and latency metrics to build a comprehensive cost profile for each pipeline paradigm.

The main contributions of this work can be listed as follows:

- **Comprehensive evaluation methodology.** To provide a holistic view of the state-of-the-art in text-to-SQL performance, we present an evaluation methodology that can objectively assess a wide spectrum of techniques (few-shot prompting, decomposition pipelines, and multi-agent frameworks) across diverse benchmark complexities. Furthermore, we look beyond accuracy alone, mapping the tradeoffs between model performance

and operational costs across three key dimensions: energy footprint, latency, and token consumption.

- **Hardware-level energy profiling.** We instrument every query individually with NVIDIA DCGM, reporting per-query GPU energy in joules. To our knowledge, this is the first hardware-level per-query GPU energy measurement study in text-to-SQL.
- **Accuracy/cost tradeoff analysis for open-source and proprietary models.** Our evaluation methodology enables a detailed comparative study of text-to-SQL methods and benchmarks, revealing critical tradeoffs that span across both accuracy and cost dimensions. We show how to tailor this analysis when using proprietary models, where additional metrics such as dollar-cost and cache-utilization also become relevant.
- **Extensible and reusable evaluation framework.** The proposed evaluation methodology is designed as an extensible framework, allowing for seamless integration of new metrics as the text-to-SQL landscape evolves. To support reproducibility and future research, we made our implementation publicly available and easily adaptable to other text-to-SQL studies.

2 RELATED WORK

The text-to-SQL landscape has been in rise for both research and industry, with contemporary solutions increasingly centered on LLMs [8, 9, 13, 14, 16]. While the literature is extensive [5, 11, 12], evaluation benchmarks remain heavily concentrated on correctness metrics such as Exact Match (EM) and Execution Accuracy (EX) [17, 19]. Systematic analysis of the computational and economic overhead of query generation remains a secondary focus, though recent studies have begun addressing this imbalance. For example, Zhang et al. incorporated monetary cost reporting for closed-source models in a finance-specific context [18], while Li et al. introduced a framework measuring GPU memory and latency alongside standard accuracy for PLM-based methods [7]. Furthermore, integrated metrics have emerged to reconcile performance with efficiency. The BIRD dataset introduced the Valid Efficiency Score (VES) to account for execution time [10], a metric later refined by Germán et al. into VES* and VCES to capture reasoning latency and end-to-end costs in ReAct-style architectures [3]. Expanding this scope further, the BIRD-INTERACT framework simulates multi-turn CRUD operations, evaluating success rates alongside rewards and monetary expenditure [6]. The body of research represented by DIN-SQL [13], DTS-SQL [14], and MAC-SQL [15] illustrates a paradigm shift in Text-to-SQL from monolithic generation to modular, agentic workflows that prioritize both reasoning depth and operational transparency. While DIN-SQL and DTS-SQL established the efficacy of multi-step decomposition for complex schemas, MAC-SQL expanded this to parallel multi-agent collaboration with interactive error handling. In DIN-SQL [13], the authors provide a detailed cost-benefit analysis, highlighting that DIN-SQL's multi-step prompting significantly increases token cost compared to *vanilla* few-shot prompting. In DTS-SQL [14], the *cost per successful query* is explicitly quantified. Furthermore, Wang et al. [15], claim that MAC-SQL reaches a point of diminishing returns in accuracy versus token expenditure. While a growing body of research attempts to capture operational overheads, a holistic evaluation that simultaneously addresses benchmark complexity, methodological variations, and the accuracy/cost tradeoff remains absent. We bridge this gap by providing a comprehensive, multi-dimensional analysis.

3 EVALUATION METHODOLOGY

3.1 Text-to-SQL Benchmarks

We use a representative set of text-to-SQL benchmarks in our evaluation, which span a spectrum of schema and query complexity. This includes a purely academic benchmark (Spider [17]), an academic benchmark based on real-world data (BIRD [10]), and a more recently released enterprise benchmark based on larger scale real-world data and query workloads (BEAVER [1]).

Spider [17] is a cross-domain text-to-SQL benchmark comprising 1,034 development questions across 20 SQLite databases. It is widely used as a standard benchmark and serves here as a representative of well-structured, academic-style schemas.

BIRD [10] contains 1,534 development questions across 11 real-world databases. Compared to Spider, BIRD features larger schemas, dirtier data, and domain-specific *evidence* annotations (free-text hints about value mappings and business rules). We evaluate on the full development set and stratify results by the three official difficulty levels: *simple*, *moderate*, and *challenging*.

BEAVER [1] is an enterprise-grade benchmark spanning three private, large-scale data warehouses: DW, an Oracle academic warehouse with 97 tables and 1,530 columns; NW, a research-laboratory MySQL cluster with 366 tables across five databases; and SP, a housing-management MySQL system with 349 tables across two databases. We evaluate on a balanced subset of 621 queries sampled from the full benchmark: 371 from DW (121 real user logs + 250 synthesized), and 125 each from NW and SP, providing roughly equal coverage across warehouses and query types. Our evaluation uses two BEAVER settings: *BEAVER S1* provides gold tables, column mappings, and join keys as oracle hints, isolating generation cost from retrieval difficulty; *BEAVER S0* is fully end-to-end: tables are retrieved with Qwen3-Embedding-8B (top-50) and re-ranked by Qwen3-Reranker-8B (top-15).

3.2 Text-to-SQL Methods

We implement and evaluate four Text-to-SQL generation strategies of increasing pipeline complexity. We use FewShot as a complexity baseline and have chosen the remaining three methods from top scoring standalone methods from BIRD and Spider benchmarks.

FewShot is a single-call baseline. The model receives the full database schema, one hand-crafted example question-SQL pair, and the target question (plus evidence for BIRD), and generates SQL in one pass. It establishes the cost lower bound and the accuracy floor.

DAIL-SQL [4] replaces the fixed example with dynamically selected few-shot demonstrations. For each query, we encode the question using `sentence-transformers/all-mpnet-base-v2` and retrieve the k=3 most similar questions from the training set by cosine similarity, excluding examples from the same database. On BEAVER, where no annotated Q-SQL training pairs exist, we use Qwen3-Embedding-8B over a hand-curated example pool instead. Like FewShot, only a single LLM call is made. When this paper was written, DAIL-SQL was in 2nd place of the Spider leaderboard and 61st place in the BIRD Validation Efficeincy Score (VES) leaderboard.

DIN-SQL [13] is a four-stage pipeline: *schema linking*, *query classification*, *SQL generation* with chain-of-thought reasoning, and *self-correction*. Each stage is a separate LLM call, yielding a minimum of four calls per query. When this paper was written, DIN-SQL was in 4th place of the Spider leaderboard and 63rd place in BIRD VES leaderboard.

MAC-SQL [15] is a three-agent collaborative framework. A *Selector* agent identifies relevant tables and columns; a *Decomposer* agent generates SQL with chain-of-thought reasoning over the filtered schema; and a *Refiner* agent self-corrects on execution errors up to three times. MAC-SQL makes two to five LLM calls per query. When this paper was written, MAC-SQL was in 52nd place in the BIRD VES leaderboard.

All methods were implemented following their respective original papers without benchmark-specific prompt tuning. For BEAVER, we used the official BEAVER benchmark scripts for FewShot, DAIL-SQL, and DIN-SQL; MAC-SQL was implemented uniformly across all benchmarks using the original paper's prompt templates. Schema exposure follows each method's original design: FewShot ingests the full schema, DAIL-SQL augments it with retrieved examples, MAC-SQL's Selector prunes it before generation, and DIN-SQL performs schema linking as its first stage. MAC-SQL's Refiner retries up to three times on execution failure, consistent with the original paper; we additionally retry on empty result sets, which are a common failure mode on enterprise schemas.

3.3 Evaluation Metrics and Cost Measurements

We report SQL generation accuracy in EX, where a generated SQL is counted correct if its result set matches the gold SQL result set after order-independent normalization.

We adopt EX as our primary accuracy metric since all benchmarks and baseline methods we compare against report it, enabling direct comparison. We acknowledge that EX has known limitations; for example, two semantically distinct queries returning an empty result set are treated as equivalent, while a query with a minor syntax error receives a score of zero despite being trivially correctable. Developing more robust accuracy metrics, such as fuzzy or semantic similarity measures, remains an open direction that we plan to explore in future work.

We measure cost along three dimensions. *Token cost* is the sum of input and output tokens reported by the Ollama API for each LLM call, aggregated per query. *Latency* is the wall-clock time from first LLM call to final SQL. *GPU energy* is measured using NVIDIA DCGM (`dcgmi dmon`) at a 100 ms sampling interval, bracketing each query from the first LLM call to the final SQL return. Energy is derived from the hardware cumulative counter (`TOTAL_ENERGY_CONSUMPTION`, field 156) as the delta between the first and last readings within the query window; no idle baseline is subtracted, as the delta naturally excludes energy consumed outside the measurement period. Process isolation is enforced by pinning each Ollama instance to a dedicated GPU via `CUDA_VISIBLE_DEVICES` and restricting the DCGM monitor to that physical device, so that concurrent workloads on the same machine do not pollute the readings. The measurement window covers the full query lifecycle, encompassing all LLM pipeline stages and any inter-call idle time, making it a measure of total per-query GPU cost rather than LLM inference alone. This hardware-level approach captures actual energy consumption independent of pricing models and is, to our knowledge, the first hardware-level per-query GPU energy measurement study in Text-to-SQL.

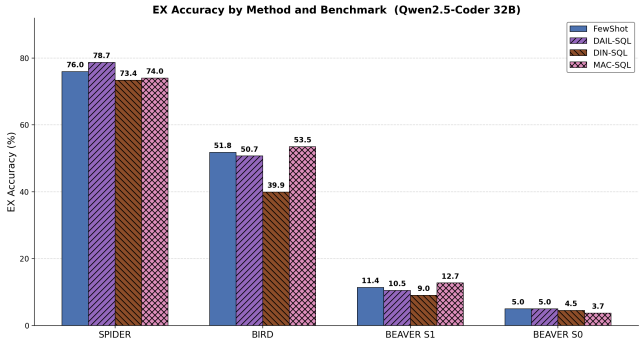

Figure 2: Execution accuracy across benchmarks

For the OpenAI API experiments, we additionally report *dollar cost per query* (billed input + output tokens at GPT-5-mini list prices) and *APC cache miss rate*: the fraction of input tokens *not* served from OpenAI's Automatic Prompt Cache, where a miss means the full prompt was re-evaluated on the server at full price.

4 EXPERIMENTAL ANALYSIS

4.1 Experiment Environment and Models

In our experiments, we used a single compute node with a dual-socket Intel® Xeon® Gold 6326 CPU with 2 x 16 cores at 2.90 GHz and 512 GB memory, and an NVIDIA RTX 6000 Ada (48 GB GDDR6). Our primary model is `qwen2.5-coder:32b`, served locally via Ollama on a single GPU; temperature is set to 0 throughout for deterministic outputs. Additionally, `GPT-5-mini` is used through the OpenAI API to measure dollar cost and prompt caching for our last set of experiments.

4.2 Execution Accuracy across Benchmarks

Figure 2 reports EX % across all three benchmarks, including both BEAVER settings (S_0 and S_1, 621 queries each). A few findings stand out:

First, **method ranking is not stable across benchmarks**. DAIL-SQL leads on Spider (78.7%) while FewShot (76.0%) finishes second; MAC-SQL is the top method on both BIRD (53.5%) and BEAVER S_1 (12.7%) yet ranks third on Spider (74.0%). DAIL-SQL (50.7% on BIRD) tracks MAC-SQL closely, finishing just 2.8 pp behind at one-quarter of its token cost. DIN-SQL trails all methods on Spider, BIRD, and BEAVER S_1, though MAC-SQL falls lowest on BEAVER S_0 (3.7% vs. DIN-SQL's 4.5%).

Second, **BEAVER reveals an enterprise performance floor**. All four methods collapse into a narrow 9.0–12.7% EX band on the enterprise warehouses under S_1 regardless of pipeline complexity, and further degrade to a 3.7–5.0% band under the end-to-end S_0 setting. Enterprise schema complexity and retrieval constitute qualitative barriers that academic benchmarks do not expose.

To better understand what drives BEAVER's enterprise performance floor, Table 1 breaks down failures under both settings

Table 1: Failure mode breakdown on BEAVER, gold-schema (S_1) vs. end-to-end (S_0) settings (Qwen2.5-Coder-32B, N=621 per method per setting). Each method's BEAVER S_1 and BEAVER S_0 columns each sum to 100% (including correct results).

	BEAVER S_1				BEAVER S_0			
Error Type	FewShot	DAIL-SQL	DIN-SQL	MAC-SQL	FewShot	DAIL-SQL	DIN-SQL	MAC-SQL
Correct	11.4	10.5	9.0	12.7	5.0	5.0	4.5	3.7
Wrong result (semantic)	53.1	49.1	50.1	63.6	38.8	37.0	17.9	36.7
Wrong column	11.3	12.6	11.0	7.9	34.9	25.0	11.4	23.3
Syntax error	16.7	18.0	21.7	2.7	15.5	13.4	60.9	7.7
Empty result	–	–	–	11.4	–	–	–	19.6
Wrong table	0.2	0.5	–	0.2	1.6	0.2	0.5	2.9
Other SQL error	7.2	9.3	8.2	1.4	4.2	19.5	4.8	6.0

(Qwen2.5-Coder-32B). Under S_1 (gold schema hints), wrong-result errors, semantically incorrect but syntactically valid SQL, dominate across all four methods (49.1–63.6%), while syntax and schema errors stay comparatively rare. The picture changes sharply under S_0: wrong-column and syntax-error rates rise for most methods as retrieval replaces gold hints, and DIN-SQL is the clearest outlier, with its syntax-error rate jumping from 21.7% under S_1 to 60.9% under S_0, four times the next-highest method. MAC-SQL is the only method with a separate empty-result category (11.4% under S_1, 19.6% under S_0): its Refiner explicitly retries when a query returns zero rows, so these cases are logged and re-attempted rather than being folded into the wrong-result bucket as they are for the other three methods. DIN-SQL's syntax-error spike is consistent with its schema-linking-first design: its first stage commits to a specific set of tables and columns before generation, so under S_0's retrieved (rather than gold) schema, an early linking mistake can propagate through the three downstream stages, including self-correction, as SQL referencing non-existent or mislabeled columns. The other methods either expose the full schema (FewShot) or re-select it at each generation call (DAIL-SQL's retrieval, MAC-SQL's Selector), making them less exposed to a single point of schema-linking failure.

Key Insight: *No single method dominates across all three benchmarks. As benchmark complexity grows, agentic methods gain an accuracy edge (MAC-SQL leads on BIRD and BEAVER S_1); but on truly enterprise end-to-end workloads (BEAVER S_0), no method yet achieves meaningful accuracy. The failure-mode breakdown (Table 1) points to a concrete driver: DIN-SQL's schema-linking-first design is uniquely fragile under retrieved (non-gold) schemas, where an early linking mistake cascades into syntax errors that self-correction cannot fix.*

4.3 Cost Profile across Benchmarks

Figure 3 shows per-metric absolute cost (GPU energy, latency, tokens) across all benchmark/method combinations as a color-normalized heatmap.

The heatmap reveals a cost split on BEAVER S_1: in latency, multi-step methods pay a heavy overhead—MAC-SQL takes 59 s/q and DIN-SQL 86 s/q due to sequential agent calls (vs. FewShot's 17 s/q). In tokens, however, **both DAIL-SQL and MAC-SQL are cheaper than FewShot**: DAIL-SQL retrieves only a short hand-curated example (2.6K tok/q) and MAC-SQL's Selector prunes the schema (5.4K tok/q) vs. FewShot's full 97-table schema (7,286 tok/q average).

On BIRD and Spider the picture is reversed: every method above FewShot costs more in both latency and tokens, with DIN-SQL reaching 27 s/q and 3.9K tok/q on BIRD (vs. FewShot's 1.6 s/q and 0.6K tok/q).

Three patterns stand out in the heatmap. First, costs generally scale with benchmark difficulty, but DIN-SQL is the notable exception: its GPU energy and latency are lower on BIRD (7.1 kJ, 27 s) than on Spider (11 kJ, 44 s), suggesting that schema structure rather than query count drives its overhead. Second, DAIL-SQL is consistently more efficient than FewShot, most starkly on BEAVER where retrieved examples replace the full 97- table schema (2.6K vs. 7.3K tok/q under S_1). Third, MAC-SQL's costs are lower on BIRD than on Spider (3.4 kJ, 13 s vs. 4.7 kJ, 18 s) before jumping sharply on BEAVER (S_0: 32 kJ, 176 s/q).

Key Insight: *Costs scale with benchmark difficulty as a general rule, but schema exposure overrides this trend. Methods that limit schema input (DAIL-SQL via example retrieval, MAC-SQL via its Selector) consistently beat the trend, while DIN-SQL's anomaly (higher cost on Spider than BIRD) reveals that schema structure matters more than query count.*

4.4 Accuracy/Cost Tradeoff

Figure 4 breaks down cost by query outcome across all benchmarks. Each method has two independent bars: the solid bar is the average cost over correct queries and the hatched bar is the average cost over incorrect queries. A consistent pattern emerges: **incorrect queries always cost more than correct ones**, across every method and every cost metric. This is not explained by extra retries. DIN-SQL issues exactly four LLM calls per query regardless of outcome, yet the gap persists. The explanation is *selection bias*: the queries a model fails on tend to be structurally harder (larger schemas, more complex joins, longer prompts), and those same properties inflate GPU energy, latency, and token count independently of the method's control flow. MAC-SQL additionally issues extra refinement calls on failed queries, compounding the gap.

Figure 5 plots EX% against each cost dimension across all benchmarks. On BEAVER S_1, DAIL-SQL occupies the Pareto efficiency frontier: at 2.4 kJ/q, 9.3 s/q, and 2.6K tok/q, it is the only method both cheaper and faster than FewShot (4.5 kJ/q, 17.3 s/q, 7.3K tok/q).

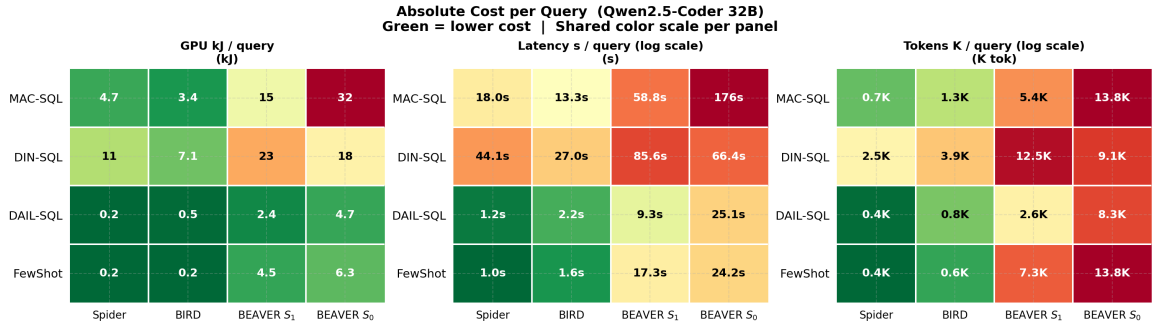

Figure 3: Absolute cost heatmap across benchmarks and methods. Each panel shares a single color scale (log-normalized for latency and tokens). Green = lower cost.

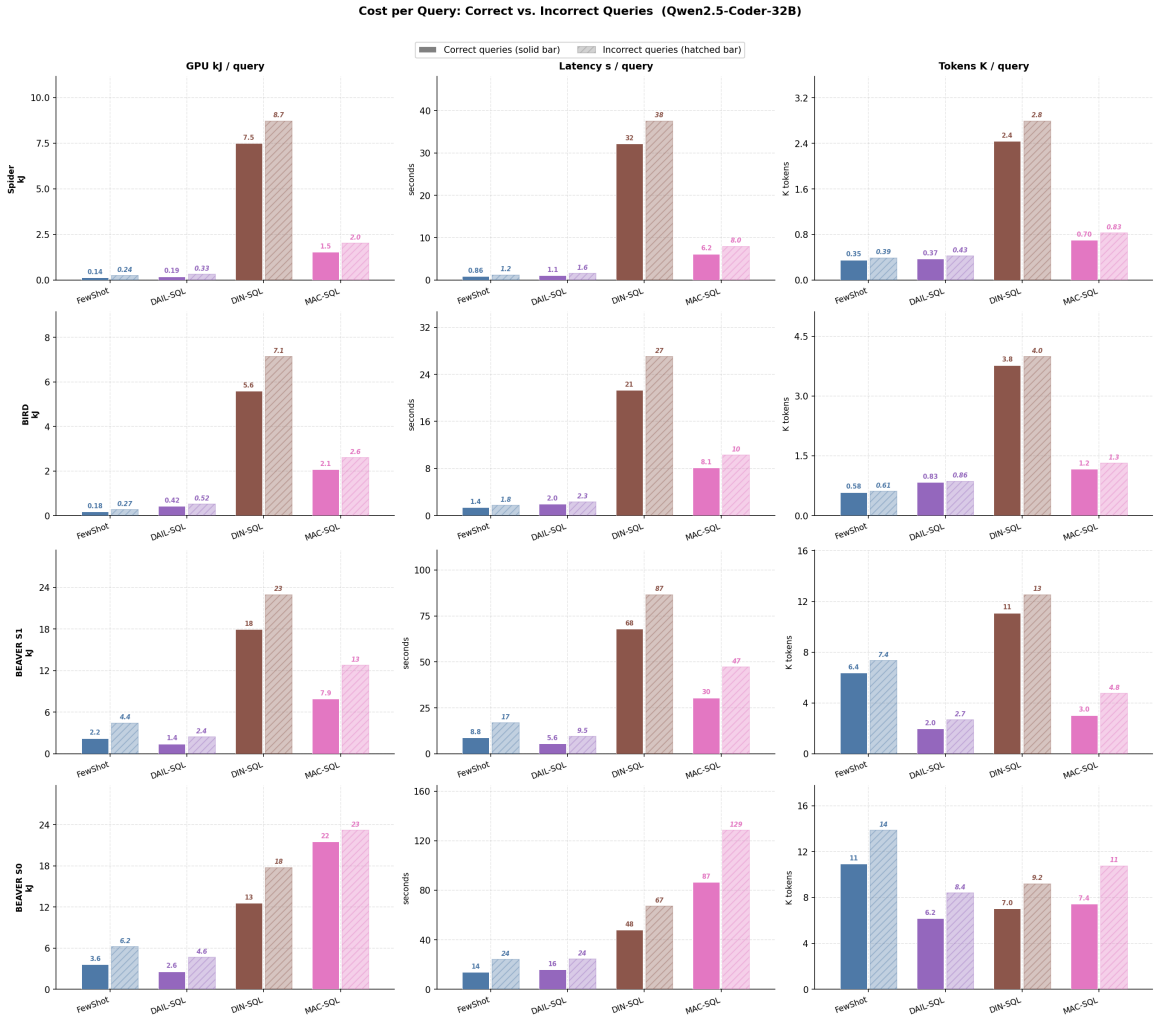

Figure 4: Average cost per query broken down by outcome. Rows = benchmarks; columns = GPU energy, latency, tokens. For each method, the left (solid) bar shows the average cost for correct queries and the right (hatched) bar shows the average cost for incorrect queries; the two bars are independent and share the same y-scale.

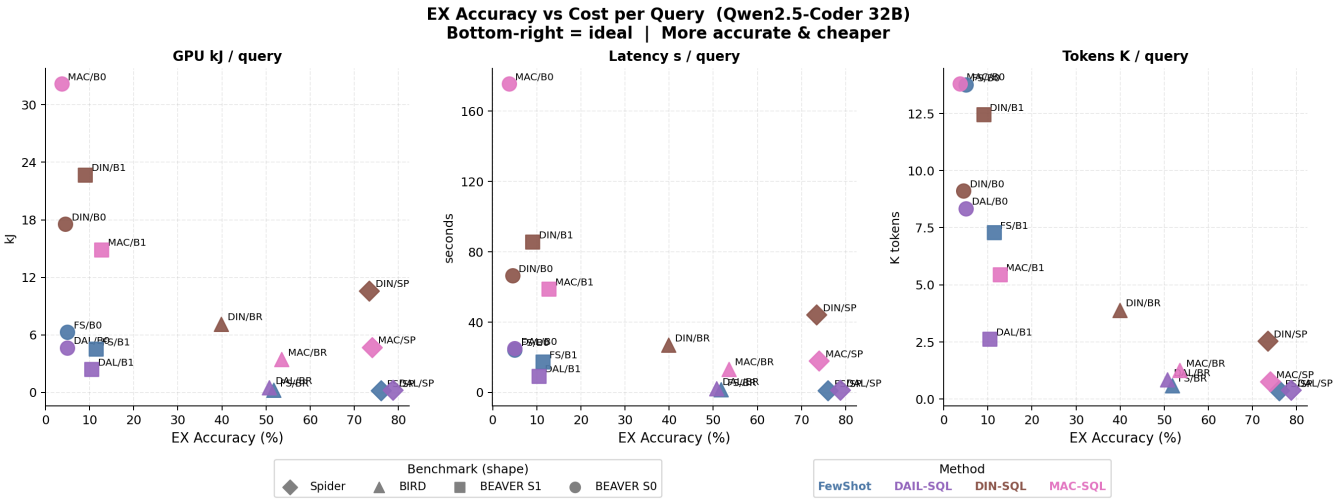

Figure 5: Execution accuracy (EX%) vs. cost per metric across benchmarks. Each panel shows one cost dimension (GPU kJ/query, latency, tokens/query). DAIL-SQL dominates the efficiency frontier on BEAVER; FewShot dominates on Spider.

DIN-SQL is strictly dominated at 23 kJ/q, 86 s/q, and 12.5K tok/q. Cheaper methods are not necessarily less accurate on BEAVER. DAIL-SQL achieves ∼10.5% EX under S_1 while consuming only 2.4 kJ/q GPU energy (vs. FewShot's 4.5 kJ) and 2.6K tok/q (vs. 7.3K), sitting on the Pareto frontier across all cost dimensions. MAC-SQL pays 59–176 s/q in latency (vs. FewShot's 17–24 s) for marginal accuracy gains, yielding poor cost-per-correct-query efficiency. The correct-vs-incorrect cost gap (Figure 4) is an artifact of query difficulty distribution, not method behavior; a distinction that matters when interpreting per-query cost averages.

Key Insight: *"Less agentic" methods (FewShot and DAIL-SQL) dominate the efficiency frontier across all benchmarks and cost dimensions, an advantage not visible when accuracy and cost are examined separately, only when considered jointly.*

4.5 Accuracy/Cost Tradeoff (Proprietary Model)

Running all four methods via the OpenAI API adds two observables absent from local runs: *dollar cost* and *Automatic Prompt Caching* (APC). APC serves the longest stable prefix of a prompt from a server-side cache at lower prices; it activates only for prompts exceeding 1,024 tokens, making it irrelevant for short academic prompts but significant on BEAVER where schema-heavy inputs routinely surpass this threshold. Methods with fixed prompt structure benefit automatically, while fully dynamic pipelines pay full price every call. Figure 6 reports these per-metric absolute costs across all three benchmarks, mirroring Figure 3 for the local model.

Figure 6's Cache Hit Rate panel reveals how prompt architecture interacts with caching. FewShot's 26% miss rate on BEAVER S_1 reflects its fixed instruction block and example, with only the per-query schema tail escaping the cache. DIN-SQL's 59% miss rate comes from *intra-query* reuse: its four sequential calls share the schema from call 1, so calls 2–4 hit the cache for that block. DAIL-SQL hits 100% on Spider and BIRD, since similarity-selected examples change every query, collapsing the stable prefix; on BEAVER its fixed example lowers this to 87–90%. MAC-SQL exceeds 99% under S_1, as its Selector and Refiner make every call effectively novel, falling to 88% under S_0. All methods hit 100% on Spider, where 20 different databases force the schema to change every query.

The primary cost driver across benchmarks is **schema volume**, not method complexity. FewShot costs 32× more per query on BEAVER S_0 than on Spider ($\$5.17\times10^{-3}$ vs $\$0.16\times10^{-3}$) purely from injecting 15 retrieved tables. Switching to gold hints (S_1) cuts cost 4.5× for FewShot and 1.75× for MAC-SQL, making table retrieval the dominant expenditure in end-to-end settings.

Figure 6 and our dual-environment evaluation show that token-based pricing does not track computational effort. Locally (Section 4.3), FewShot and MAC-SQL consume the same 13.8K tokens on BEAVER S_0, yet MAC-SQL uses over five times the GPU energy (32 kJ vs. 6.3 kJ) and seven times the latency (176 s vs. 24.2 s). On the API side, this overhead is largely absorbed by the provider: DIN-SQL and MAC-SQL draw nearly identical tokens on GPT-5-mini (13,265 vs. 13,272), yet MAC-SQL's latency is double (44.1 s vs. 21.2 s) while its dollar cost is only 1.2× higher ($\$7.3\times10^{-3}$ vs. $\$6.0\times10^{-3}$), a gap at least partly explained by caching (41% hit rate for DIN-SQL vs. 12% for MAC-SQL) and input/output token-mix differences. Token-centric pricing thus fails to capture the true hardware cost of complex agentic pipelines.

Figure 7 plots EX% against cost and efficiency metrics across benchmarks. Accuracy on Spider and BIRD stays competitive with local Qwen2.5-Coder 32B: DAIL-SQL leads Spider (74.6%) at equal cost and half the latency of FewShot (73.5%), and leads BIRD (50.3%) over DIN-SQL (41.5%) despite a simpler design. On BEAVER, EX collapses to 3.5–10.3% across all methods (MAC-SQL best), confirming enterprise schema complexity as the binding constraint regardless of provider.

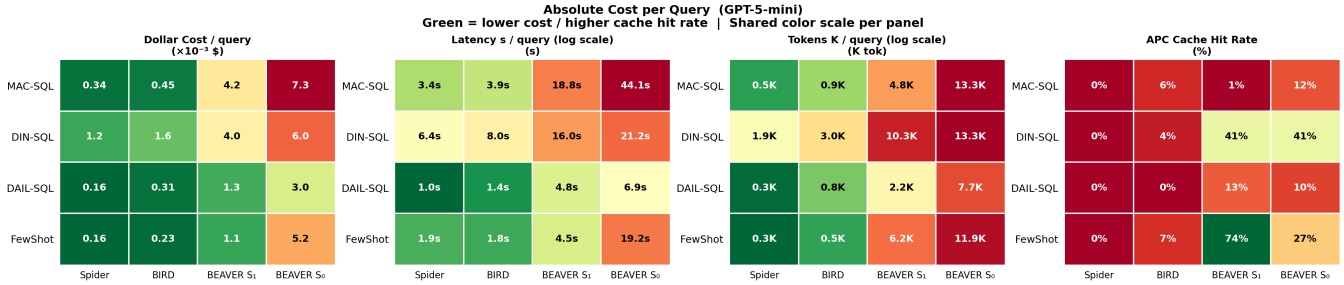

Figure 6: GPT-5-mini absolute cost heatmap. Dollar cost, latency, tokens, and APC cache hit rate per query across benchmarks. Green = lower cost / higher cache hit rate.

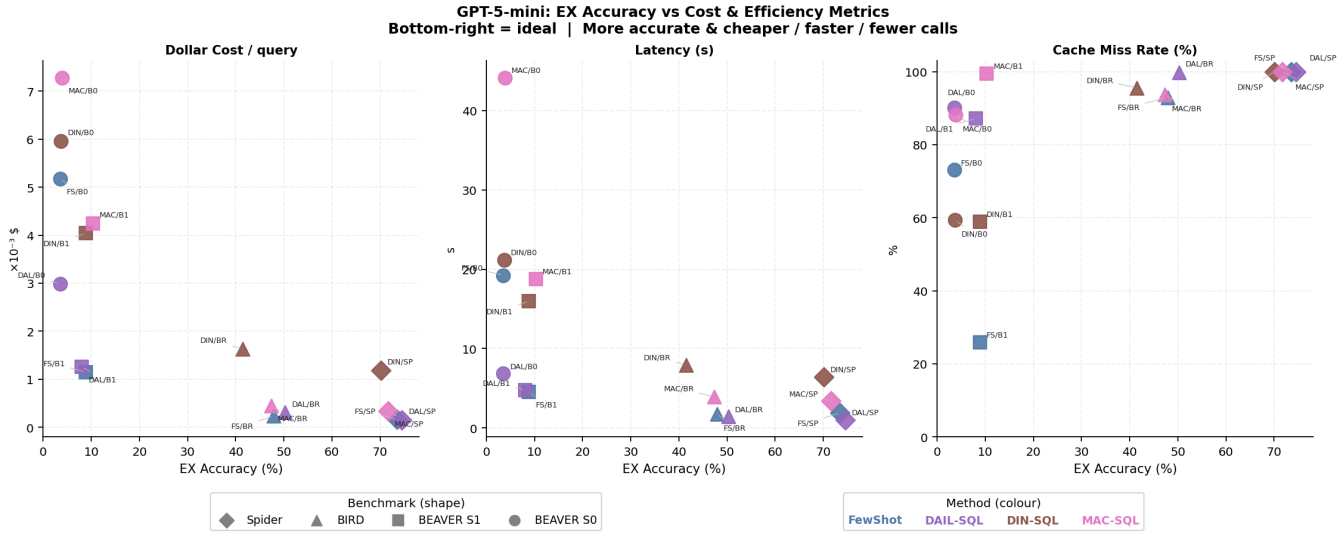

Figure 7: GPT-5-mini: execution accuracy (EX%) vs. cost per metric across benchmarks. Each panel shows one dimension (dollar cost, latency, cache miss rate).

Key Insight: *Our findings generalize across model providers: DAIL-SQL's efficiency-edge and BEAVER's accuracy-collapse both replicate on GPT-5-mini, confirming the bottleneck is schema complexity, not model choice. The one proprietary-only lever is prompt caching, which discounts cost for fixed-prefix methods like FewShot, but this alone is not sufficient to recover the loss in accuracy. Beyond caching, Figure 6 shows that billed dollar cost badly under-represents the compute gap between methods: MAC-SQL and DIN-SQL draw near-identical tokens on BEAVER S_0 yet differ 2× in latency while differing only 1.2× in price, so token-based pricing is not a reliable proxy for the underlying execution cost.*

5 CONCLUSION

This study introduces a methodology for analyzing accuracy-cost tradeoffs within the rapidly evolving text-to-SQL landscape, specifically tracking the shift toward agentic architectures and enterprise-scale benchmarks characterized by complex schemas and queries. Our empirical evaluation spans three benchmark datasets (ranging from academic to enterprise-scale) and four distinct methodologies (ranging from single LLM calls to multi-call cooperative agentic architectures). The resulting data quantifies the operational overhead inherent to each approach and isolates the underlying cost drivers, offering actionable insights for designing future enterprise-grade text-to-SQL solutions. A promising avenue for future work is the integration of our evaluation methodology into standard text-to-SQL benchmarking systems, establishing a more multi-dimensional evaluation standard for the research community.

ACKNOWLEDGMENTS

We thank Peter Baile Chen and Michael Stonebraker for their feedback and support. This research was supported by the EXA4MIND project, funded by the European Union's Horizon Europe Research and Innovation Programme, under Grant Agreement No 101092944. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them. This work is also partially funded by METU under the grant no. ADEP-312-2024-11484. LLMs were utilized to improve the text and the code.

REFERENCES

[1] Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. 2025. BEAVER: An Enterprise Benchmark for Text-to-SQL. arXiv:2409.02038

[2] Saurabh Deochake and Debajyoti Mukhopadhyay. 2026. Cost Trade-offs of Reasoning and Non-Reasoning Large Language Models in Text-to-SQL. arXiv:2512.22364

[3] Germán T. Eizaguirre, Lars Tissen, and Marc Sánchez-Artigas. 2026. Both Ends Count! Just How Good are LLM Agents at Text-to-Big SQL?. In *EuroMLSys Workshop*. 333–345.

[4] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *PVLDB* 17, 5 (2024), 1132–1145.

[5] Zijin Hong et al. 2025. Next-Generation Database Interfaces: A Survey of LLM-Based Text-to-SQL. *IEEE TKDE* 37, 12 (2025), 7328–7345.

[6] Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Ozcan, Yannis Papakonstantinou, and Reynold Cheng. 2025. BIRD-INTERACT: Re-imagining Text-to-SQL Evaluation for Large Language Models via Lens of Dynamic Interactions. arXiv:2510.05318

[7] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? *PVLDB* 17, 11 (2024), 3318–3331.

[8] Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. OmniSQL: Synthesizing High-Quality Text-to-SQL Data at Scale. *PVLDB* 18, 11 (2025), 4695–4709.

[9] Haoyang Li, Jing Zhang, Cuiping Han, et al. 2023. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. In *AAAI Conference*. 13067–13075.

[10] Jinyang Li, Binyuan Hui, Ge Qu, Biao Li, Jiaxi Yang, Bowen Li, Bowen Wang, Jice Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as a Database Interface? A Big Bench for Large-Scale Database Grounded Text-to-SQLs. In *NeurIPS Conference*. 35741–35760.

[11] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2025. A Survey of Text-to-SQL in the Era of LLMs: Where Are We, and Where Are We Going? *IEEE TKDE* 37, 10 (2025), 5735–5754.

[12] Yuyu Luo, Guoliang Li, Ju Fan, Chengliang Chai, and Nan Tang. 2025. Natural Language to SQL: State of the Art and Open Problems. *PVLDB* 18, 12 (2025), 5466–5471.

[13] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *NeurIPS Conference*. 35761–35785.

[14] Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models. In *EMNLP 2024*. 8212–8220.

[15] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. In *COLING Conference*. 540–557.

[16] Zhongyuan Wang, Richong Zhang, Zhijie Nie, and Jaein Kim. 2025. ToolSQL: A Tool-Assisted Agent for SQL Verification and Refinement. In *KDD Conference*. 3102–3110.

[17] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *EMNLP Conference*. 3911–3921.

[18] Chao Zhang, Yuren Mao, Yijiang Fan, Yu Mi, Yunjun Gao, Lu Chen, Dongfang Lou, and Jinshu Lin. 2024. FinSQL: Model-Agnostic LLMs-Based Text-to-SQL Framework for Financial Analysis. In *SIGMOD*. 93–105.

[19] Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic Evaluation for Text-to-SQL with Distilled Test Suites. In *EMNLP Conference*. 396–411.