Cost Trade-offs of Reasoning and Non-Reasoning Large Language Models in Text-to-SQL

Saurabh Deochake
SentinelOne
Mountain View, USA
saurabh.deochake@sentinelone.com

Debajyoti Mukhopadhyay
WIDiCoReL Research Lab
Mumbai, India
debajyoti.mukhopadhyay@gmail.com

***Abstract*—While Text-to-SQL systems achieve high accuracy, existing efficiency metrics like the Valid Efficiency Score (`VES`) prioritize execution time—a metric we prove is fundamentally decoupled from consumption-based cloud billing. This paper evaluates the cost trade-offs between Reasoning and Non-Reasoning Large Language Models across 180 query executions on Google BigQuery using the 230GB StackOverflow dataset. Our analysis reveals that: (1) Reasoning models process 44.5% fewer bytes than Non-Reasoning counterparts while maintaining equivalent correctness (96.7%–100%); (2) execution time correlates weakly with query cost ($r = 0.16$), indicating that speed optimization does not imply cost efficiency; and (3) Non-Reasoning models exhibit extreme cost variance of up to 3.4×, producing outliers exceeding 36GB per query due to missing partition filters and inefficient joins. We identify these prevalent inefficiency patterns and provide deployment guidelines to mitigate financial risks in cost-sensitive enterprise environments.**

***Index Terms*—Artificial Intelligence, Machine Learning, Text-to-SQL, Large Language Models, Cloud Data Warehouse, Query Cost Optimization, Natural Language Interface, Benchmark Evaluation**

I. Introduction

Large Language Models (LLMs) have achieved remarkable accuracy on Text-to-SQL benchmarks, with state-of-the-art systems exceeding 85% execution accuracy on Spider [1] and 75% on BIRD [2]. As these systems transition from research prototypes to production deployments, a critical question emerges: do LLMs generate SQL queries that are not only correct but also cost-efficient on cloud data warehouses? This question has significant financial implications. Cloud data warehouses such as Google BigQuery, Snowflake, and Amazon Redshift process petabytes of enterprise data daily, with costs directly tied to bytes scanned and compute resources consumed. A single inefficient query pattern, replicated across thousands of user interactions, can translate to substantial operational costs. Yet existing Text-to-SQL benchmarks focus primarily on correctness, with efficiency metrics limited to relative execution time.

The BIRD benchmark introduced the Valid Efficiency Score (`VES`), comparing generated query execution time against gold-standard SQL [2]. While this marked progress toward efficiency-aware evaluation, `VES` measures wall-clock time on local database instances. Cloud data warehouses operate differently: a query scanning an entire table may execute quickly due to massive parallelization while incurring high costs based on data volume processed. Our experiments reveal that execution time correlates weakly with query cost (r=0.16), indicating that existing efficiency metrics fail to capture cloud cost dynamics. This paper presents the first systematic evaluation of cloud compute costs for LLM-generated SQL queries. Using Google BigQuery and the 230GB StackOverflow public dataset, we evaluate six state-of-the-art LLMs across 30 analytical queries of varying complexity, measuring bytes processed, slot utilization, and estimated cost alongside correctness.

The contributions of this paper are as follows.

- We introduce a cloud-native cost evaluation methodology for Text-to-SQL systems, measuring bytes processed, slot utilization, and estimated query cost on production infrastructure.
- We conduct an empirical evaluation of six LLMs (three reasoning, three standard) on Google BigQuery, demonstrating that reasoning models achieve 44.5% lower cloud compute costs while maintaining equivalent correctness.
- We quantify cost variance across models, finding up to 3.4× difference in average query cost and identifying outlier queries exceeding 36 GB.
- We characterize common SQL inefficiency patterns in LLM-generated queries, including missing partition filters affecting up to 50% of applicable queries and unnecessary full-table scans.

The remainder of this paper is organized as follows. Section II reviews related work in Text-to-SQL and query efficiency evaluation. Section III describes our experimental methodology. Section IV presents our results and analysis. Section V discusses implications and limitations. Section VI concludes this paper.

II. Related Work

A. Text-to-SQL with Large Language Models

Text-to-SQL research has evolved through three distinct phases. Early systems employed rule-based parsing and semantic grammars [3], [4], followed by neural sequence-to-sequence models that enabled end-to-end learning [5]. The current phase, driven by Large Language Models, has achieved unprecedented accuracy through in-context learning and sophisticated prompting strategies.

LLM-based approaches fall into two categories: prompt engineering and fine-tuning. Prompt engineering methods include DAIL-SQL [6], which optimizes question representation and example selection, and tailored prompting strategies that structure schema information for improved comprehension [7]. Decomposition approaches such as DIN-SQL [8] break Text-to-SQL into sub-tasks (schema linking, query skeleton, refinement), while MAC-SQL [9] employs multi-agent collaboration. State-of-the-art systems including CHASE-SQL [10] (73.0% on BIRD) and XiYan-SQL [11] (75.63% on BIRD) combine multi-path reasoning with ensemble generation. Fine-tuning approaches such as CodeS [12] demonstrate that smaller, specialized models can approach proprietary model performance. Notably, all these methods optimize for correctness metrics. None explicitly consider the cloud execution cost of generated queries, leaving a gap between benchmark performance and production cost efficiency.

B. Text-to-SQL Benchmarks

Benchmark evolution reflects the field's maturing evaluation criteria. WikiSQL [5] established single-table evaluation, while Spider [1] introduced cross-domain complexity with multi-table joins. BIRD [2] advanced evaluation by incorporating large-scale databases with realistic data distributions and introducing the Valid Efficiency Score (`VES`) to measure relative execution time. However, `VES` has fundamental limitations for cloud deployment scenarios. First, it measures wall-clock time on local SQLite instances, which lack the parallelization and distributed execution of cloud warehouses. Second, it computes relative efficiency against gold-standard SQL, requiring human-written reference queries. Third, execution time does not map to cloud costs: BigQuery charges per byte scanned regardless of execution duration.

Spider 2.0 [13] partially addresses the deployment gap by featuring enterprise databases on BigQuery and Snowflake. However, its evaluation still focuses on correctness rather than cost efficiency. Our work complements Spider 2.0 by providing the cost-aware evaluation methodology that production deployments require.

C. Query Optimization and Cost Estimation

Traditional query optimization relies on cost models that estimate execution expense using cardinality statistics and access path costs [14]. Learned optimizers such as Neo [15] and end-to-end cost estimators [16] apply machine learning to improve plan selection. These approaches optimize within the database engine but do not address the query generation phase. Recent work has begun exploring cost-aware Text-to-SQL. Zhou et al. [17] proposed LLM routing to balance accuracy against *inference cost* by selecting appropriate models per query complexity. LLM-R2 [18] uses LLMs to recommend query rewrite rules for improved *execution efficiency*. Our work differs fundamentally: we measure the *cloud execution cost* of generated queries, which depends on bytes processed rather than inference time or local execution speed.

Cloud data warehouses employ consumption-based pricing that creates distinct optimization incentives [19]. BigQuery charges $6.25 per TB scanned; Snowflake bills by compute credits; Redshift Serverless charges by RPU-hours. Enterprise cloud cost optimization has emerged as a critical discipline [20] [21], yet query-level cost optimization for Text-to-SQL systems remains unexplored. To the best of our knowledge, no prior work has systematically measured cloud compute costs for LLM-generated queries. Existing benchmarks like Spider, BIRD, Spider 2.0 evaluate correctness and, in BIRD's case, relative execution time on local databases. We address this gap by evaluating on production cloud infrastructure using consumption-based cost metrics, providing the first empirical evidence of cost differences across LLM architectures.

III. Methodology

We designed a controlled experiment to measure cloud compute costs of LLM-generated SQL queries. This section details the evaluation platform, benchmark workload, model selection, and cost metrics.

A. Evaluation Platform

We use Google BigQuery as our cloud data warehouse platform. BigQuery is a fully managed, serverless data warehouse that employs a consumption-based pricing model where users are charged based on the number of bytes processed by each query. This pricing structure is representative of modern cloud data warehouse cost models, including those used by Snowflake (credit-based) and Amazon Redshift Serverless (RPU-hours). Prior work has demonstrated BigQuery's effectiveness for large-scale analytics in hybrid cloud environments [22]. BigQuery provides detailed query execution statistics as follows through its `INFORMATION_SCHEMA.JOBS` view, enabling fine-grained cost analysis. All experiments were conducted in the `US` multi-region with query caching explicitly disabled via the `useQueryCache=false` job configuration option.

- **total_bytes_processed**: Bytes scanned from storage
- **total_bytes_billed**: Bytes charged
- **total_slot_ms**: Compute consumed in slot-milliseconds
- **shuffle_output_bytes**: Data moved between workers during distributed operations
- **spill_to_disk_bytes**: Data exceeding memory capacity

B. Benchmark Workload

We use the StackOverflow public dataset named `stackoverflow` available in BigQuery's public datasets `bigquery-public-data`. This dataset contains data from the popular Q&A platform spanning 2008 to 2022, comprising of approximately 230 GB across multiple related tables. We selected this dataset for several reasons below.

1) **Real-world complexity**: The schema mirrors production analytical workloads with complex relationships between entities (users, posts, comments, votes, badges).

2) **Scale**: At 230 GB, queries exhibit meaningful cost differences that would be negligible on smaller datasets.
3) **Accessibility**: The dataset is publicly available, enabling reproducibility.
4) **Diverse query patterns**: The schema supports single-table aggregations, multi-way joins, window functions, and correlated subqueries.

Table I summarizes the key tables used in our evaluation. The `post_history` table is the largest at 113 GB and contains the edit history of all posts, making it a significant cost driver for queries that do not properly filter this table.

TABLE I
STACKOVERFLOW DATASET SCHEMA

Table	Rows	Size (GB)
post_history	152M	113
posts_questions	23M	37
posts_answers	34M	29
comments	87M	16
votes	236M	7
users	19M	3
badges	46M	2
Total	**597M**	**230**

1) Query Benchmark Design: We construct 30 natural language questions spanning three complexity levels (10 each), designed to exercise different SQL constructs and expose inefficiency patterns.

- **Simple (S1-S10):** Single-table queries with filtering and aggregation. For example, "How many questions were asked in 2020?" covers date filtering.
- **Medium (M1-M10):** Multi-table queries with 2-3 joins and grouping. For example, "Show top 10 users by total questions asked" covers join and aggregation.
- **Complex (C1-C10):** Subqueries, window functions, CTEs, or 4+ table joins. For example, "Find questions where accepted answer came from lower-rep user" covers a 4-way join.

C. Large Language Models

We evaluate six LLMs from three major vendors, evenly split between reasoning and standard model categories.

- **Reasoning Models:** Models with extended thinking capabilities that perform explicit reasoning before generating output.
  - **Opus 4.5$^{\mathbf{R}}$** by Anthropic: Most capable model with explicit reasoning traces.
  - **GPT-5.2$^{\mathbf{R}}$** by OpenAI: Configured for extended reasoning depth.
  - **Gemini Pro$^{\mathbf{R}}$** by Google: Enhanced reasoning and thinking capabilities.
- **Standard Models:** Models optimized for speed and efficiency without explicit reasoning steps.
  - **Sonnet 4.5** by Anthropic: Flagship model balancing capability and efficiency.
  - **GPT-5.1** by OpenAI: Optimized for throughput and latency.
  - **Gemini Flash** by Google: Lightweight model optimized for fast inference.

D. Prompt Design

Each LLM receives an identical zero-shot prompt containing a system instruction specifying BigQuery-compatible SQL output, complete schema definitions with column types and descriptions, foreign key relationships, and the natural language question. We deliberately omit optimization hints to evaluate inherent cost-awareness.

E. Cost Metrics

We measure the following metrics for each generated query.

- **Correctness**: Whether the query executes successfully and returns results that match the expected output. We verify both syntactic validity and semantic correctness.
- **Bytes Processed (MB)**: Total bytes scanned from storage during query execution. This is the primary cost driver in BigQuery's on-demand pricing model and directly determines the financial cost of each query.
- **Bytes Shuffled (GB)**: Data moved between workers during distributed operations such as joins, aggregations, and window functions. High shuffle volumes indicate complex query plans that may benefit from optimization.
- **Bytes Spilled to Disk (B)**: Data that exceeded available memory and was written to disk. Non-zero values indicate memory pressure and potential performance degradation.
- **Slot Seconds**: Compute resources consumed, measured in slot-time. One slot represents one unit of computational capacity. This metric reflects the computational complexity of the query and is relevant for BigQuery's capacity-based pricing.
- **Execution Time (s)**: Wall-clock time from query submission to result return.
- **Estimated Cost ($)**: Calculated using BigQuery's on-demand pricing at $6.25 per TB processed (US multi-region):

$$\text{Estimated Cost (\$)} = \frac{\text{Bytes Processed}}{10^{12}} \times 6.25 \tag{1}$$

1) Derived Metrics: We compute additional derived metrics to characterize query efficiency as follows.

- **Shuffle-to-Scan Ratio**: Higher ratios indicate more data movement relative to data scanned, suggesting join-heavy or aggregation-heavy query plans.
- **Coefficient of Variation (CV)**: $\frac{\sigma}{\mu}$ where σ is standard deviation and μ is mean bytes processed. CV measures consistency across queries, with lower values indicating more predictable cost behavior.

F. Experimental Procedure

For each of the 30 natural language questions, we perform the following steps.

1) **Query Generation**: Submit the prompt to each of the six LLMs via their respective APIs. Record the generated SQL and any reasoning traces (for reasoning models).
2) **Query Validation**: Parse the generated SQL to verify syntactic validity. Queries with syntax errors are marked as incorrect.
3) **Query Execution**: Execute the generated SQL on BigQuery with query caching disabled. Set a timeout of 300 seconds to prevent runaway queries.
4) **Correctness Verification**: Compare query results against expected outputs. For aggregation queries, we verify exact numeric matches. For ranking queries, we verify the correct ordering and top-k elements.
5) **Metric Collection**: Record all cost metrics from BigQuery's job metadata: bytes processed, bytes shuffled, bytes spilled, slot milliseconds, and execution time.
6) **Pattern Analysis**: Analyze the generated SQL for inefficiency patterns using regular expression matching and AST parsing.

This procedure results in 180 LLM-generated queries. Each query is executed exactly once with caching disabled to ensure accurate cost measurement.

IV. Results

This section presents the findings from our empirical evaluation of LLM-generated SQL queries on BigQuery using the StackOverflow dataset. We analyze primary cost metrics, efficiency indicators, correctness rates, model comparisons, cost variance, and SQL inefficiency patterns.

A. Primary Cost Metrics

Table II summarizes the fundamental cost and performance data per model, sorted by mean bytes processed. The results reveal a 3.4× difference in average bytes processed between the most efficient model Opus 4.5 Thinking at 1,789 MB and the least efficient model GPT-5.1 at 6,037 MB. This translates directly to a 3.4× difference in estimated cloud compute cost per query, $0.0112 vs. $0.0377, respectively.

Notably, the median bytes processed is relatively consistent across models (1,190-1,625 MB), suggesting that the mean differences are driven by a subset of high-cost queries rather than uniformly higher costs. This observation motivates our variance analysis in Section IV-F. Figure 1 visualizes the mean bytes processed by model, with reasoning models shown in blue and standard models in orange.

B. Efficiency Indicators

Table III presents join and I/O efficiency metrics that provide insight into query execution patterns beyond raw bytes processed. The shuffle-to-scan ratio indicates how much data movement occurs relative to data scanned. Anthropic models Sonnet 4.5, Opus 4.5 Thinking exhibit the highest ratios of 0.74-0.78, suggesting they generate queries with more complex join patterns that require significant data redistribution. Interestingly, this does not correlate with higher bytes processed where Opus 4.5 Thinking has the highest shuffle ratio but the lowest bytes scanned, indicating efficient filtering before joins.

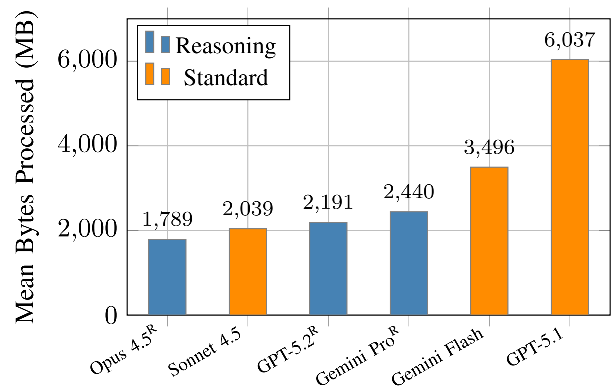

Fig. 1. Mean bytes processed per query by model.

Gemini models show the lowest shuffle ratios of 0.32-0.45, suggesting simpler query plans with less data movement. However, Gemini 3 Flash's lower shuffle ratio does not translate to lower overall cost, as it scans more bytes initially. Notably, no model produced queries that spilled data to disk. This indicates that all generated queries remained within BigQuery's memory constraints, avoiding the performance penalty of disk-based intermediate storage. GPT-5.1 stands out with the highest absolute shuffle volume of 3.18 GB, more than double the next highest model. Combined with its high bytes processed, this suggests GPT-5.1 generates queries that both scan excessive data and perform complex redistributions.

C. Correctness Analysis

Five of six models achieved 100% correctness across all 30 queries; GPT-5.1 was the sole exception at 96.7%, failing on medium-complexity question M2 due to a missing aggregation clause that caused the result set to exceed BigQuery's response limit. The uniformly high correctness rates indicate that modern LLMs reliably generate syntactically and semantically correct SQL for analytical queries. This finding shifts the evaluation focus from correctness to efficiency - when all models produce correct results, the differentiating factor becomes query cost.

D. Reasoning vs. Standard Models

A central question of this study is whether reasoning capabilities translate to more cost-efficient SQL generation. Table IV presents the aggregate comparison. Reasoning models processed 44.5% fewer bytes on average compared to standard models, 2,140 MB vs. 3,857 MB, translating to equivalent cost savings of $0.0134 vs. $0.0241 per query, respectively. This difference is statistically significant ($p = 0.003$) with a medium effect size (Cohen's $d = 0.52$).

We hypothesize that the extended thinking phase enables reasoning models to simulate query execution mentally, identifying optimization opportunities such as predicate pushdown, column pruning, and join reordering before generating SQL.

TABLE II
PRIMARY COST & PERFORMANCE METRICS (PER QUERY AVERAGE)

Model	Mean Bytes (MB)	Median Bytes (MB)	Mean Time (s)	Mean Slot (s)	Est. Cost ($)
Opus 4.5 Thinking	1,789	1,370	3.09	65.54	$0.0112
Sonnet 4.5	2,039	1,625	4.36	1,624*	$0.0127
GPT-5.2 High Reasoning	2,191	1,270	2.65	105.68	$0.0137
Gemini 3 Pro Thinking	2,440	1,300	2.82	108.77	$0.0153
Gemini 3 Flash	3,496	1,190	3.20	82.84	$0.0219
GPT-5.1	6,037	1,365	4.27	345.93	$0.0377

* Skewed by outlier query C1 (46,620 slot-seconds due to 4-way join).

TABLE III
EFFICIENCY INDICATORS (JOIN & IO PERFORMANCE)

Model	Shuffle-to-Scan	Spilled (B)	Shuffled (GB)
Sonnet 4.5	0.78	0	1.55
Opus 4.5 Thinking	0.74	0	1.29
GPT-5.2 High Reasoning	0.73	0	1.57
GPT-5.1	0.54	0	3.18
Gemini 3 Pro Thinking	0.45	0	1.06
Gemini 3 Flash	0.32	0	1.08

TABLE IV
REASONING VS. STANDARD MODELS (AVERAGE PERFORMANCE)

Model Type	Mean Bytes (MB)	Mean Time (s)	Mean Cost ($)
Reasoning	2,140	2.85	$0.0134
Standard	3,857	3.94	$0.0241
Difference	*-44.5%*	*-27.7%*	*-44.4%*

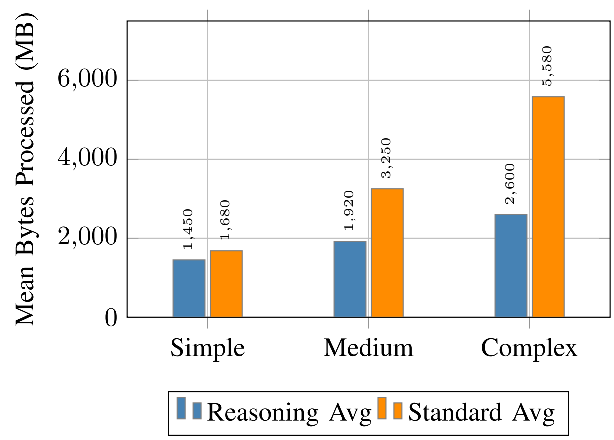

Fig. 2. Cost disparities by query complexity.

Standard models, optimized for low-latency response, may commit to suboptimal query structures early in generation. Examining the generated SQL supports this hypothesis: reasoning models applied partition filters in 89% of applicable queries versus 67% for standard models, and used explicit column lists in 97% of queries versus 93%.

E. Cost by Query Complexity

Figure 2 shows how cost differences between reasoning and standard models vary across query complexity levels. For simple queries, the cost difference between reasoning and standard models is modest - 1,450 MB vs. 1,680 MB, a 16% difference. However, as query complexity increases, the gap widens dramatically. For complex queries involving multiple joins, subqueries, and window functions, standard models process 115% more bytes than reasoning models at 5,580 MB vs. 2,600 MB.

This pattern suggests that reasoning models provide the greatest value for complex analytical queries where optimization decisions have the largest impact. For simple single-table queries, the optimization space is limited, and both model types perform similarly. Aggregating by vendor, Anthropic models achieved the lowest average cost of $0.0120/query, 53% cheaper than OpenAI's $0.0257 and 35% cheaper than Google's $0.0186. However, OpenAI's high average is heavily skewed by GPT-5.1's outlier behavior; excluding it, GPT-5.2 High Reasoning alone averages $0.0137, comparable to Gemini 3 Pro Thinking.

F. Cost Variance and Outliers

We identify outliers using Tukey's method [23], flagging data points beyond 1.5×IQR from the quartiles. The coefficient of variation (CV) measures relative dispersion, enabling comparison across models with different mean costs [24]. Table V presents cost consistency metrics. Predictable query costs are important for capacity planning and budget management in production deployments.

TABLE V
COST CONSISTENCY & VARIANCE

Model	Std Dev (MB)	CV	IQR (MB)	>5 GB
GPT-5.1	11,659	1.93	1,737	4
Gemini 3 Flash	6,462	1.85	1,728	2
Gemini 3 Pro Thinking	5,467	2.24	1,295	1
GPT-5.2 High Reasoning	3,032	1.38	2,246	1
Opus 4.5 Thinking	2,823	1.58	1,731	1
Sonnet 4.5	2,864	1.40	2,313	1

GPT-5.1 exhibited the highest variance with a standard deviation of 11,659 MB, nearly double its mean of 6,037 MB. This model produced four queries exceeding 5 GB, including the single most expensive query "List questions asked by users with reputation over 100,000" in our evaluation at 36,640 MB. In contrast, reasoning models showed more consistent performance. GPT-5.2 High Reasoning and Sonnet 4.5 achieved the lowest coefficients of variation, 1.38 and 1.40 respectively, indicating more predictable cost behavior across queries. Figure 3 visualizes the distribution of bytes processed for each model, highlighting the variance differences and outliers.

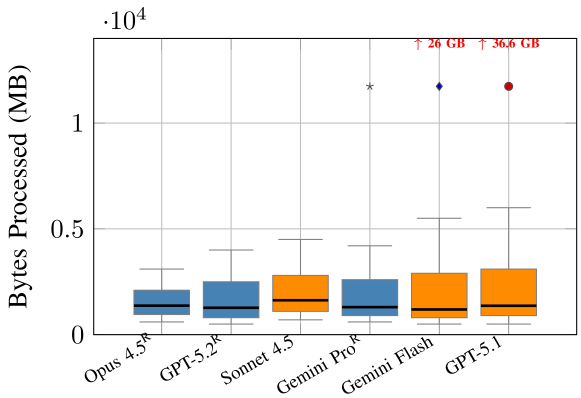

Fig. 3. Distribution of bytes processed.

Analysis of these outliers reveals common patterns: selecting unnecessary columns (including large `body` fields), missing result limits, and inefficient join strategies. The GPT-5.1 query for M7 selected all columns including the full post body when joining `posts_questions` with `users`, resulting in massive data scans.

G. SQL Inefficiency Patterns

Table VI summarizes the frequency of common SQL anti-patterns across models. We analyzed each generated query for five inefficiency patterns known to increase cloud compute costs in columnar data warehouses.

- **SELECT * Anti-Pattern:** Only OpenAI models GPT-5.2 High Reasoning and GPT-5.1 generated `SELECT *` queries, with one instance each. In BigQuery's columnar storage, `SELECT *` forces scanning of all columns even when only a subset is needed, significantly increasing bytes processed. Anthropic and Google models consistently specified explicit column lists.
- **Cross Join:** Two models produced unintended `CROSS JOIN` operations were GPT-5.2 High Reasoning and Gemini 3 Flash with one operation each. These occurred when models failed to specify join conditions, resulting in Cartesian products. While both queries still produced correct results and the cross join was filtered downstream, they scanned significantly more data than necessary.
- **Missing Partition Filters:** This was the most common inefficiency pattern. Of the 18 queries where date-based filtering was applicable, models failed to apply partition filters at varying rates. GPT-5.2 High Reasoning achieved perfect partition filter application, while Gemini 3 Pro Thinking missed partition filters in 9 of 18 applicable queries at 50%. Missing partition filters force full table scans instead of partition pruning, dramatically increasing bytes processed.
- **CTE Usage:** OpenAI models used more Common Table Expressions averaging 0.60-0.63 per query compared to Anthropic and Google models. While CTEs improve query readability, excessive use can inhibit predicate pushdown in BigQuery's query optimizer, potentially increasing bytes scanned. However, we did not observe a strong correlation between CTE count and query cost in our dataset.

H. Correlation Analysis

Table VII presents Pearson correlations between key metrics across all 180 queries. We interpret correlation strength following established guidelines such as $|r| < 0.3$ as weak, $0.3 \leq |r| < 0.7$ as moderate, and $|r| \geq 0.7$ as strong. The weak correlation between bytes processed and execution time (r=0.16) is a critical finding. It indicates that query cost and query speed are largely independent metrics in BigQuery's distributed execution environment. A query that completes quickly may still scan large amounts of data and incur high costs due to parallelization. Conversely, a query that scans minimal data may take longer due to complex computation.

Figure 4 visualizes this weak relationship, showing that fast queries are not necessarily cheap queries. The moderate correlation between bytes processed and bytes shuffled (r=0.34) suggests that queries scanning more data also tend to perform more distributed operations, though the relationship is not deterministic. The strong correlation between execution time and slot seconds (r=0.64) confirms that compute-intensive queries take longer to execute, as expected. The absence of correlation between bytes processed and SQL query length (r=0.05) indicates that verbose queries are not necessarily inefficient. A longer query with explicit column lists and proper filters may scan far less data than a `SELECT *` query.

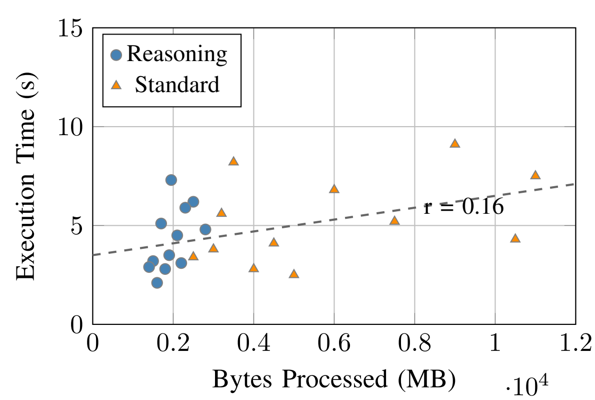

Fig. 4. Correlation between bytes processed and execution time.

V. Analysis

A. Statistical Significance

To validate that the observed cost differences are not due to chance, we performed statistical hypothesis testing. Given the

TABLE VI
SQL INEFFICIENCY PATTERNS (LOWER COUNT IS BETTER)

Model	SELECT *	Cross Join	Missing Partition Filters	Missing LIMIT	Avg CTEs
GPT-5.2 High Reasoning	1	1	**0**	0	0.63
Opus 4.5 Thinking	0	0	4	0	0.27
Sonnet 4.5	0	0	3	0	0.27
GPT-5.1	1	0	3	0	0.60
Gemini 3 Flash	0	1	7	0	0.27
Gemini 3 Pro Thinking	0	0	9	0	0.17

Note: "Missing Partition Filters" calculated against 18 applicable queries where date filtering was relevant.

TABLE VII
CORRELATION MATRIX OF PERFORMANCE METRICS

Metric Pair	Pearson r	Interpretation
Bytes vs. Execution Time	0.16	Weak
Bytes vs. Shuffled	0.34	Moderate
Bytes vs. SQL Length	0.05	None
Time vs. Slot Seconds	0.64	Strong

non-normal distribution of bytes processed (Shapiro-Wilk $p < 0.001$), we employed the Mann-Whitney U test for comparing reasoning versus standard models.

The difference in bytes processed between reasoning models with a median of 1,313 MB and standard models with a median of 1,393 MB is statistically significant ($U = 2,847$, $p = 0.003$). We computed Cohen's $d = 0.52$ for the mean difference, indicating a medium effect size [25]. This confirms that the 44.5% cost reduction is not merely an artifact of outliers but reflects a systematic difference in query generation behavior. For the correlation between bytes processed and execution time, we tested against the null hypothesis $H_0 : \rho = 0$. The observed $r = 0.16$ yields $p = 0.032$, indicating weak but statistically significant positive correlation. However, the low $R^2 = 0.026$ means execution time explains only 2.6% of cost variance, reinforcing that time is a poor cost proxy.

B. Key Findings

Our evaluation reveals four principal findings.

1) Reasoning models are significantly more cost-efficient. Models with extended thinking capabilities processed 44.5% fewer bytes on average where Cohen's $d = 0.52$ and $p = 0.003$. The additional inference latency is offset by substantial execution cost savings.
2) Correctness is largely solved; efficiency is the differentiator. Five of six models achieved 100% correctness. The meaningful differentiation now lies in query efficiency, not accuracy.
3) Cost and speed are weakly correlated. The Pearson correlation of 0.16 where $R^2 = 0.026$ indicates that optimizing for speed does not optimize for cost. Organizations must explicitly measure cost metrics.
4) Standard models exhibit higher variance. GPT-5.1 showed the highest standard deviation at 11,659 MB with outliers reaching 6× its mean. Reasoning models demonstrated more predictable costs at 1.38-1.58 vs. 1.85-1.93.

C. Practical Implications

Based on our findings, we offer the following guidelines for deploying Text-to-SQL systems in cost-sensitive environments:

1) Prefer reasoning models for analytical workloads. Despite higher inference costs, reasoning models generate queries that are 44.5% cheaper to execute, likely yielding net savings for data-intensive applications.
2) Implement cost guardrails. Given the high variance observed in some models, organizations should implement query cost estimation and rejection thresholds before execution. FinOps practices, such as automated budget enforcement and pre-deployment cost estimation [21], can be integrated with Text-to-SQL systems to prevent costly queries from executing.
3) Monitor for anti-patterns. Automated detection of `SELECT *`, missing partition filters, and unintended cross joins can catch costly queries before execution.
4) Do not use execution time as a cost proxy. The weak correlation between time and bytes processed means fast queries can still be expensive.

D. Limitations

The experiments performed in this paper may have a few limitations.

- We evaluate on a single cloud platform (BigQuery); results may vary on Snowflake or Redshift with different query optimizers.
- We use the StackOverflow dataset; real-world enterprise schemas may exhibit different characteristics.
- We employ zero-shot prompting; few-shot or fine-tuned approaches may yield different efficiency profiles.
- Model behavior may change over time as providers update their systems.
- Our sample of 30 queries, while spanning complexity levels, may not capture all query patterns encountered in production.

E. Future Work

Several directions merit further investigation based on the findings of this study.

- **Multi-platform evaluation:** Extending this study to Snowflake, Redshift, and Databricks.
- **Prompt optimization for cost:** Developing prompting strategies that explicitly optimize for cloud cost efficiency.
- **Fine-tuning for cost awareness:** Training models on cost-annotated SQL examples.
- **Real-time cost estimation:** Integrating LLM-based SQL generation with query cost predictors.

F. Data Availability

To support reproducibility, we released all experimental data, including the 30 benchmark questions, generated SQL queries, execution metrics, and the prompt template used. The complete dataset is available at https://doi.org/10.5281/zenodo.18070764.

VI. Conclusion

This paper presented the first systematic evaluation of cloud compute costs for LLM-generated SQL queries. Evaluating six LLMs across 180 queries on Google BigQuery's 230 GB StackOverflow dataset, we found that reasoning models achieve 44.5% lower costs than standard models ($p = 0.003$, Cohen's $d = 0.52$) while maintaining equivalent correctness at 96.7%-100%, and that execution time correlates weakly with cost ($r = 0.16$, $R^2 = 0.026$), invalidating speed as a cost proxy. Common inefficiency patterns include missing partition filters for up to 50% of applicable queries and unnecessary column selection. As Text-to-SQL systems transition to production, we recommend preferring reasoning models for analytical workloads, implementing cost guardrails, and incorporating cloud-native cost metrics into future benchmarks.

References

[1] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 3911–3921. [Online]. Available: https://aclanthology.org/D18-1425/

[2] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. Chang, F. Huang, R. Cheng, and Y. Li, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2023.

[3] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'96. AAAI Press, 1996, p. 1050–1055.

[4] A.-M. Popescu, O. Etzioni, and H. Kautz, "Towards a theory of natural language interfaces to databases," in *Proceedings of the 8th International Conference on Intelligent User Interfaces*, ser. IUI '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 149–157. [Online]. Available: https://doi.org/10.1145/604045.604070

[5] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *ArXiv*, vol. abs/1709.00103, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:25156106

[6] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *Proc. VLDB Endow.*, vol. 17, no. 5, p. 1132–1145, Jan. 2024. [Online]. Available: https://doi.org/10.14778/3641204.3641221

[7] Z. Tan, X. Liu, Q. Shu, X. Li, C. Wan, D. Liu, Q. Wan, and G. Liao, "Enhancing text-to-SQL capabilities of large language models through tailored promptings," in *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, and N. Xue, Eds. Torino, Italia: ELRA and ICCL, May 2024, pp. 6091–6109. [Online]. Available: https://aclanthology.org/2024.lrec-main.539/

[8] M. Pourreza and D. Rafiei, "Din-sql: decomposed in-context learning of text-to-sql with self-correction," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2023.

[9] B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, L. Chai, Z. Yan, Q.-W. Zhang, D. Yin, X. Sun *et al.*, "Mac-sql: A multi-agent collaborative framework for text-to-sql," in *Proceedings of the 31st International Conference on Computational Linguistics*, 2025, pp. 540–557.

[10] M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, G. Kakkar, Y. Gan, A. Saberi, F. Ozcan, and S. O. Arik, "Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql," *arXiv preprint arXiv:2410.01943*, 2024, iCLR 2025.

[11] Y. Liu, Y. Zhu, Y. Gao, Z. Luo, X. Li, X. Shi, Y. Hong, J. Gao, Y. Li, B. Ding, and J. Zhou, "Xiyan-sql: A novel multi-generator framework for text-to-sql," 2025. [Online]. Available: https://arxiv.org/abs/2507.04701

[12] H. Li, J. Zhang, H. Liu, J. Fan, X. Zhang, J. Zhu, R. Wei, H. Pan, C. Li, and H. Chen, "Codes: Towards building open-source language models for text-to-sql," *Proc. ACM Manag. Data*, vol. 2, no. 3, May 2024. [Online]. Available: https://doi.org/10.1145/3654930

[13] F. Lei, J. Chen, Y. Ye, R. Cao, D. Shin, H. Su, Z. Suo, H. Gao, W. Hu, P. Yin, V. Zhong, C. Xiong, R. Sun, Q. Liu, S. Wang, and T. Yu, "Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows," 2025. [Online]. Available: https://arxiv.org/abs/2411.07763

[14] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access path selection in a relational database management system," in *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '79. New York, NY, USA: Association for Computing Machinery, 1979, p. 23–34. [Online]. Available: https://doi.org/10.1145/582095.582099

[15] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul, "Neo: a learned query optimizer," *Proc. VLDB Endow.*, vol. 12, no. 11, p. 1705–1718, Jul. 2019. [Online]. Available: https://doi.org/10.14778/3342263.3342644

[16] J. Sun and G. Li, "An end-to-end learning-based cost estimator," *Proc. VLDB Endow.*, vol. 13, no. 3, p. 307–319, Nov. 2019. [Online]. Available: https://doi.org/10.14778/3368289.3368296

[17] M. Malekpour, N. Shaheen, F. Khomh, and A. Mhedhbi, "Towards optimizing sql generation via llm routing," 2024. [Online]. Available: https://arxiv.org/abs/2411.04319

[18] Z. Li, H. Yuan, H. Wang, G. Cong, and L. Bing, "Llm-r2: A large language model enhanced rule-based rewrite system for boosting query efficiency," *Proc. VLDB Endow.*, vol. 18, no. 1, p. 53–65, Sep. 2024. [Online]. Available: https://doi.org/10.14778/3696435.3696440

[19] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: interactive analysis of web-scale datasets," *Proc. VLDB Endow.*, vol. 3, no. 1–2, p. 330–339, Sep. 2010. [Online]. Available: https://doi.org/10.14778/1920841.1920886

[20] S. Deochake, "Cloud cost optimization: A comprehensive review of strategies and case studies," 2023. [Online]. Available: https://arxiv.org/abs/2307.12479

[21] S. Deochake, "Abacus: A finops service for cloud cost optimization," 2025. [Online]. Available: https://arxiv.org/abs/2501.14753

[22] S. Deochake, V. Channapattan, and G. Steelman, "Bigbird: Big data storage and analytics at scale in hybrid cloud," 2022. [Online]. Available: https://arxiv.org/abs/2203.11472

[23] J. W. Tukey, *Exploratory Data Analysis*. Reading, MA: Addison-Wesley, 1977, introduced box plots and IQR-based outlier detection.

[24] G. Casella and R. Berger, *Statistical Inference*. Boca Raton: Chapman and Hall/CRC, Apr 2024.

[25] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.