The Impact of Metadata Configurations on Text-to-SQL Performance: A Comprehensive Analysis

Gareth Price*
Chief Technology Officer
CORRALDATA

January 27, 2025

Abstract

This research presents a comprehensive empirical study analyzing the impact of different metadata configurations on text-to-SQL system performance. Through extensive testing across multiple queries of varying complexity, we evaluate combinations of seven metadata types: schema information, column descriptions, table descriptions, examples, foreign keys, primary keys, and common queries. Our analysis encompasses multiple performance metrics including query execution success, answer correctness, and results equivalence. The empirical analysis demonstrates that metadata configurations incorporating schema information, column descriptions, table descriptions, foreign keys, primary keys, and common queries achieves the highest answer correctness 84.5%, outperforming simpler configurations. We also find that while incorporating additional metadata elements generally improves performance, the benefits diminish beyond three or four components. These results provide practical guidelines for implementing efficient text-to-SQL systems.

Keywords: text-to-SQL, metadata configuration, natural language processing, database systems

*Corresponding author: gareth@corraldata.com

1 Introduction

Text-to-SQL systems are emerging as a crucial tool for making databases accessible to non-technical users. While recent advances in large language models have improved the ability of these systems to generate correct output, the optimal use of metadata configuration in order to ground system outputs to accurately map to real-world databases requires further investigation. This paper presents a comprehensive analysis of how different metadata configurations affect multiple aspects of text-to-SQL performance, including success rates, answer correctness, results equivalence, and execution times.

2 Background and Related Work

Recent work in text-to-SQL systems has primarily focused on model architectures and training approaches. While some studies have explored the impact of schema information and examples, a systematic analysis of metadata configuration impact has been lacking. Our work fills this gap by providing empirical evidence of the effectiveness of different metadata combinations across multiple performance metrics.

Our analysis is informed by CorralData's extensive production deployment of text-to-SQL systems, which in 2024 processed over 100,000 natural language queries from enterprise users, generating SQL that analyzed more than 6 trillion rows of real-world business data across diverse industries and producing 30,000 comprehensive analytical reports. This production experience provides unique insights into the challenges and requirements of deploying text-to-SQL systems at scale with messy, real-world data.

2.1 The Role of Metadata in text-to-SQL

Metadata serves as a crucial intermediary layer in text-to-SQL systems, facilitating the semantic bridge between natural language queries and structured database operations. This interpretive layer encompasses multiple dimensions of contextual information: schema structures that define the architectural framework of the database, relationship mappings that establish connections between different data entities, and semantic annotations that provide meaning and context to raw database elements. The metadata layer functions as both a constraint mechanism—limiting the solution space to valid database operations—and an enrichment mechanism that provides essential context for accurate query interpretation. This dual role proves particularly important in enterprise environments where databases often contain complex relationships and domain-specific terminology that must be accurately interpreted for successful SQL generation. Furthermore, metadata serves as a critical optimization mechanism, enabling systems to generate not just syntactically correct queries, but queries that align with established performance patterns and business logic requirements.

2.2 Recent Work on Metadata in Text-to-SQL

A recent comprehensive survey by Singh et al., 2024 highlights the growing importance of LLM-based approaches in text-to-SQL systems, particularly emphasizing the role of metadata in improving query generation accuracy and domain adaptation. Their analysis of current benchmarks and applications provides valuable context for understanding the impact of different metadata configurations on system performance.

Recent advances in LLM adaptation techniques by Sun et al., 2024 have shown promising results in improving text-to-SQL performance through comprehensive frameworks that combine few-shot prompting with instruction fine-tuning. Their work demonstrates the importance of expanded training data coverage and query-specific database content in enhancing model performance.

Recent work by Zhang et al., 2024 has demonstrated the potential of integrating multiple LLM components for enhanced text-to-SQL performance, particularly through their SQLfuse system which leverages schema mining and linking capabilities. Their work highlights

the importance of comprehensive metadata utilization in improving query generation accuracy.

While Nguyen et al., 2023 highlight significant challenges in creating universal text-to-SQL solutions, particularly around dataset creation and evaluation, our experience suggests that well-scoped domain-specific applications can achieve commercially viable success rates. The key differentiator lies in carefully controlled metadata configurations and domain-specific optimizations that can substantially improve performance in targeted business contexts.

A comprehensive survey by Johnson et al., 2023 analyzed the evolution of metadata management in text-to-SQL systems, highlighting how different approaches to metadata augmentation have impacted system performance. Their analysis revealed that systems incorporating rich metadata configurations consistently outperformed those relying solely on basic schema information, with particularly strong improvements in handling complex queries involving multiple tables and nested operations.

3 Methodology

3.1 Experimental Setup

Our experiments were conducted using the Northwind sample database (White et al., 2022) accessed via SQLite 3, a commonly implemented demonstration database with clear semantic naming conventions. All text-to-SQL generation and results evaluation was performed using OpenAI's GPT-4o model.

3.1.1 Note on Schema Semantics

It's important to note that our results reflect performance on a semantically explicit schema where table and column names are descriptively named (e.g., "`Products`", "`OrderDetails`", "`UnitPrice`"). In real-world applications with less intuitive naming conventions (e.g., "`t1`", "`col_7`"), the impact of column descriptions would likely be substantially higher than observed in our study.

3.1.2 Tested Combinations

This study analyzed 128 different metadata combinations across a test set of 19 queries for a total of 2,432 combinations across a comprehensive test set spanning three complexity levels:

- Simple queries (basic SELECT statements)
- Medium complexity queries (joins and aggregations)
- Complex queries (nested queries and complex conditions)

3.1.3 Seven Types of Metadata Configuration Tested

The seven metadata types tested were:

Schema information (`schema`) SQL CREATE TABLE statements describing the database structure. For example:

```
CREATE TABLE Order Details (
    OrderID INTEGER PRIMARY KEY,
    ProductID INTEGER PRIMARY KEY,
    UnitPrice NUMERIC NOT NULL,
    Quantity INTEGER NOT NULL,
    Discount REAL NOT NULL
);
```

Column descriptions (`column_desc`) Natural language descriptions of each column's contents. For example:

```
"Order Details.OrderID": "Foreign key reference to Orders table",
"Order Details.ProductID": "Foreign key reference to Products table",
"Order Details.UnitPrice": "Price per unit at the time of order",
"Order Details.Quantity": "Number of units ordered",
"Order Details.Discount": "Discount applied to the order item (0-1)"
```

Table descriptions (`table_desc`) Natural language descriptions of each table's purpose and contents. For example:

```
"Order Details": "Contains the line items for each order, including product
    information, pricing, quantity, and any applied discounts. Links orders
    to specific products."
```

Examples (`examples`) Sample rows from the database demonstrating actual data patterns. For example:

```
{
    "OrderID": 10248,
    "ProductID": 11,
    "UnitPrice": 14,
    "Quantity": 12,
    "Discount": 0.0
}
```

Foreign keys (`foreign_keys`) Specifications of how tables are linked through their columns. For example:

```
{
    "table": "Order Details",
    "column": "ProductID",
```

```
    "references_table": "Products",
    "references_column": "ProductID"
}
```

Primary keys (`primary_keys`) Identification of unique identifier columns for each table. For example:

```
"Order Details": [
    "OrderID",
    "ProductID"
]
```

Common queries (`common_queries`) Curated set of functional SQL query examples with natural language names describing their purpose. For example:

```
"revenue_by_customer": "
SELECT
    c.CompanyName,
    c.Country,
    SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)) as total_revenue
FROM Customers c
JOIN Orders o ON c.CustomerID = o.CustomerID
JOIN "Order Details" od ON o.OrderID = od.OrderID
GROUP BY c.CompanyName, c.Country
ORDER BY total_revenue DESC"
```

3.2 Evaluation Metrics

We measured three key metrics:

1. **Answer correctness:** Degree of semantic alignment with expected output retrieved by executing a reference query against the same database that returned the correct values (0-1)
2. **Query Execution success rate:** Percentage of queries that were valid SQL that was executed successfully
3. **Execution time:** Time taken to generate and execute the SQL query

Answer correctness was chosen as the primary success metric, as query execution and performance are irrelevant if the returned data is incorrect. Execution time was taken as an average of 5 runs.

Evaluation was performed using an LLM-as-a-judge evaluation approach using OpenAI's GPT4-o model, with human review to calibrate results.

4 Results

4.1 Overall Performance

4.1.1 Top Performing Configurations

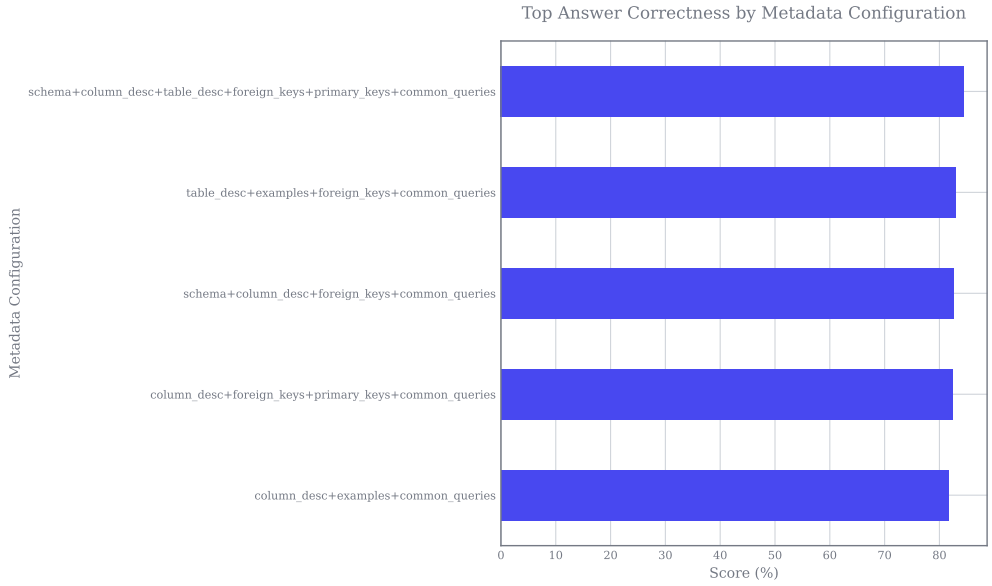

Figure 1: Performance comparison of top configurations

1. schema ∪ column_desc ∪ table_desc ∪ foreign_keys ∪ primary_keys ∪ common_queries
   - **84.5%** answer correctness
   - 100% query execution success rate
   - 2.946s average execution time
2. table_desc ∪ examples ∪ foreign_keys ∪ common_queries
   - **82.9%** answer correctness
   - 100% query execution success rate
   - 3.403s average execution time
3. schema ∪ column_desc ∪ foreign_keys ∪ common_queries
   - **82.6%** answer correctness
   - 100% query execution success rate
   - 3.018s average execution time
4. column_desc ∪ foreign_keys ∪ primary_keys ∪ common_queries
   - **82.4%** answer correctness

- 100% query execution success rate
- 2.953s average execution time

5. table_desc ∪ examples ∪ common_queries
   - **81.6%** answer correctness
   - 100% query execution success rate
   - 3.269s average execution time

4.1.2 Worst Performing Configurations

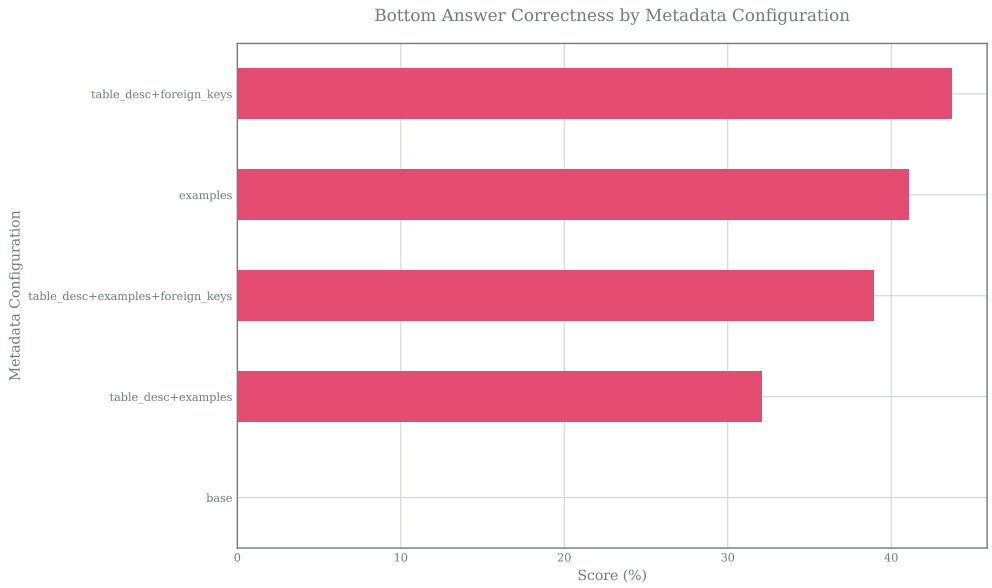

Figure 2: Performance comparison of worst configurations

1. Base (no metadata)
   - 0% answer correctness
   - 0% query execution success rate
   - 3.118s average execution time
2. table_desc ∪ examples
   - **32.1%** answer correctness
   - **47.3%** query execution success rate
   - 2.459s average execution time
3. table_desc ∪ examples ∪ foreign_keys
   - **38.9%** answer correctness

- **52.6%** query execution success rate
- 2.827s average execution time

4. examples
   - 41.1% answer correctness
   - 57.8% query execution success rate
   - 3.484s average execution time
5. table_desc ∪ foreign_keys
   - **43.7%** answer correctness
   - **52.6%** query execution success rate
   - 2.644s average execution time

4.2 Impact Analysis

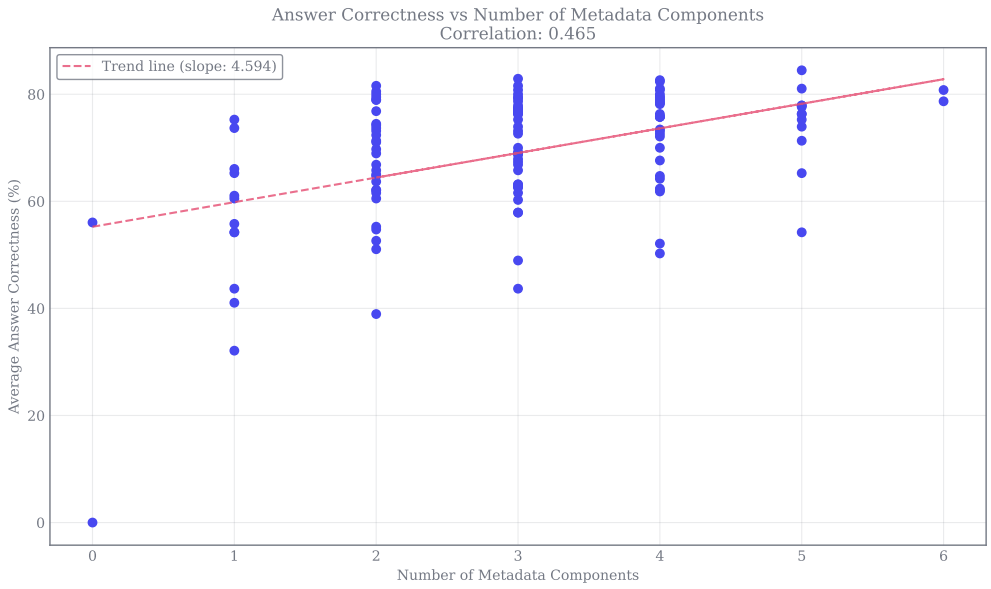

Figure 3: Correlation between number of components and performance

The analysis reveals a clear pattern of improving returns as metadata components are combined.

4.2.1 Component Combinations

- Transitioning from one to three components shows substantial improvement (average increase of ∼15 percentage points)

- Adding a fourth component produces diminished improvements (average increase of ∼5 percentage points)
- Benefits become marginal beyond four components , as execution time and computational token consumption increases
- Some combinations perform worse than simpler configurations, indicating potential interference between components or that the LLM exhibits reduced performance with too much information

4.3 Optimal Configurations

- Generally the more metadata the better, for correctness of results and query execution success rate
- Best performing combinations include 3-4 carefully selected components, but it is challenging to identify which 3-4 components to include as different queries and schemas perform better with different combinations
- Schema information combined with column descriptions and foreign keys consistently performs well
- Adding common queries to this base generally improves performance
- A preliminary implementation strategy is to include as much metadata as possible. As context windows grow and costs decrease this will likely continue to be a valid strategy that is expedient to deploy.
- Dynamically optimizing the metadata components used based on various factors could be a path to maximizing performance, efficiency and correctness.

4.3.1 Correlation Between Execution Time and Number of Components

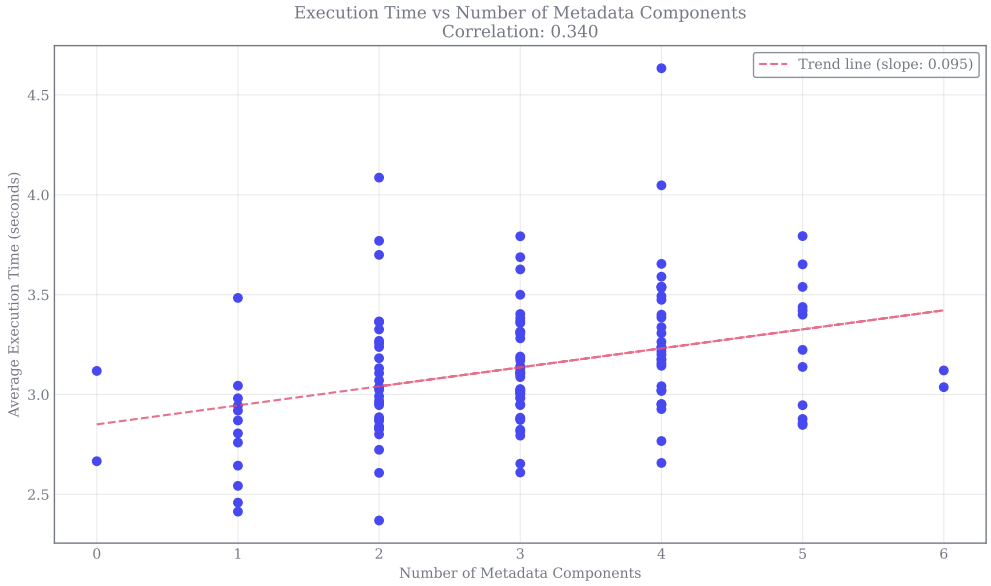

Figure 4: Correlation between number of components and execution time

With a correlation co-efficient of 0.340 there is a slight correlation between number of components and execution speed, suggesting that other factors have a larger impact on performance.

4.3.2 Performance by Query Category

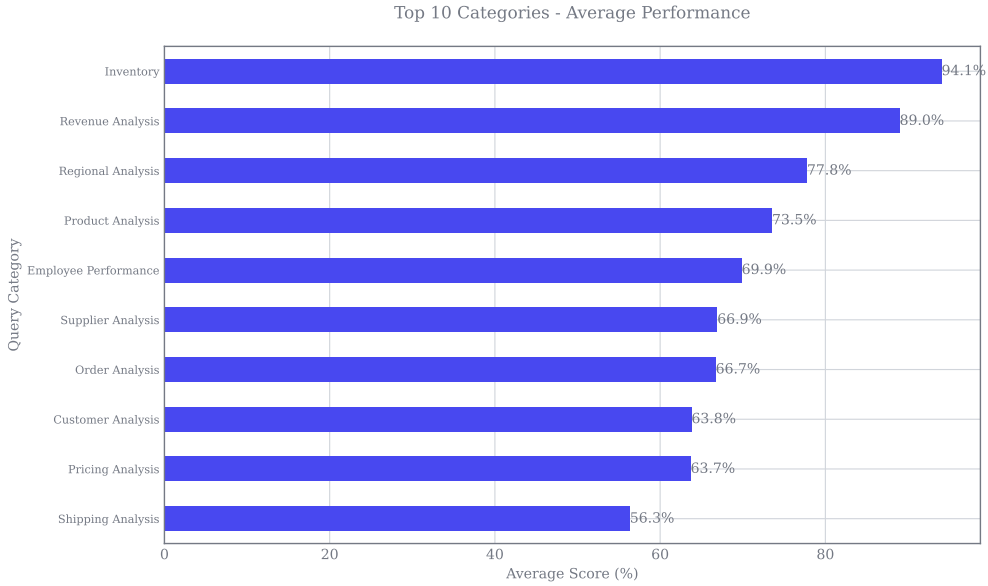

Figure 5: Performance by Query Category

Table 1: Performance by Query Category

Category	Answer Correctness	Query Executable	Execution Time
Inventory	94.1%	98.3%	1.34s
Revenue Analysis	88.98%	96.7%	2.12s
Regional Analysis	77.77%	75.0%	2.58s
Product Analysis	73.54%	89.2%	2.14s
Employee Performance	69.86%	91.7%	3.05s
Supplier Analysis	66.91%	75.0%	2.73s
Order Analysis	66.71%	77.5%	2.67s
Customer Analysis	63.78%	83.3%	3.85s
Pricing Analysis	63.68%	85.8%	2.15s
Shipping Analysis	56.32%	75.0%	3.21s

Highly variable performance by category suggests that metadata should be targeted to cover different parts of the data stored in the database.

4.3.3 Correlation Between Query Executability and Number of Metadata Components

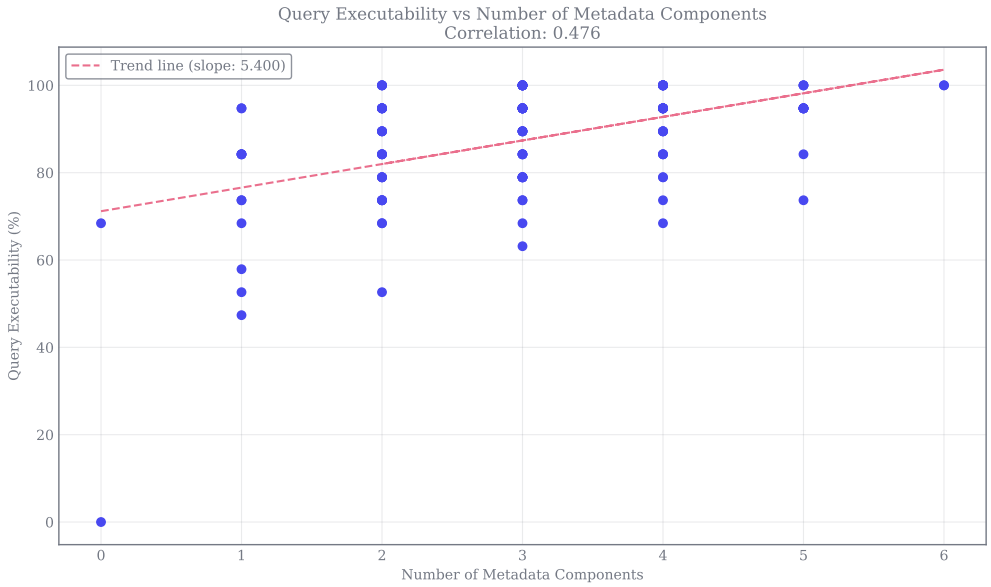

Figure 6: Correlation between number of components and query executability

While the correlation co-efficient is 0.476, there is a dramatic difference between no metadata and some metadata, even if small amounts. This indicates that there may be further research to explore in dynamically selecting which metadata components to utilize and using a small number instead of using as many as possible.

4.3.4 Performance by Query Complexity

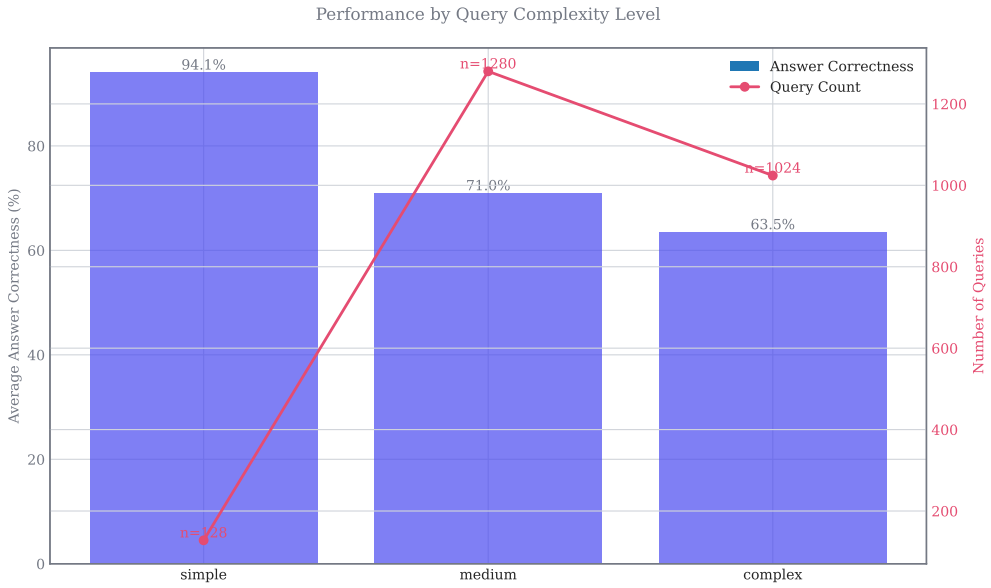

Figure 7: Performance by Query Complexity

Table 2: Performance by Query Complexity

Category	Queries	Permutations	Answer Correctness
simple	1	128	94.1%
medium	8	1280	71.0%
complex	10	1024	63.5%

4.4 Impact of Individual Components

To understand the contribution of each metadata component to overall system performance, we conducted a comprehensive ablation study. This analysis examined the system's performance at the baseline (no metadata), the performance of each component in isolation, and then the impact of the component when mixed with others.

4.5 Base Performance

Without any metadata configuration, the system was unable to correctly generate any correct queries to retrieve data, demonstrating the critical importance of metadata for text-to-SQL generation.

4.6 Individual Component Performance

Each metadata component was tested in isolation to measure its individual contribution:

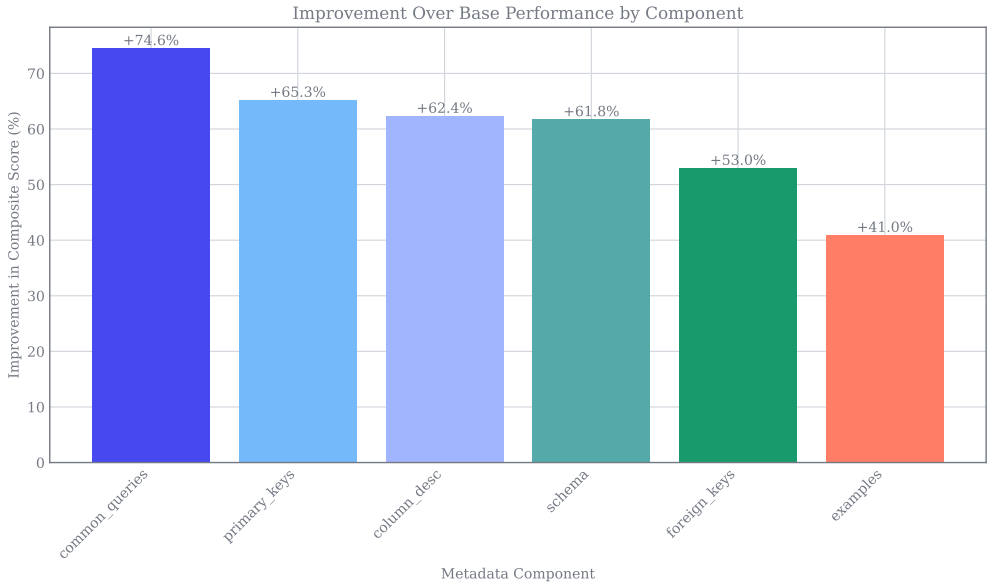

Figure 8: Impact of individual components

Common queries are a dominant performer, with the combination of metadata and business logic provided in a symbolic language giving the greatest boost to the correctness of the text-to-SQL answers. Primary Keys provide context on how to relate to other parts of the database.

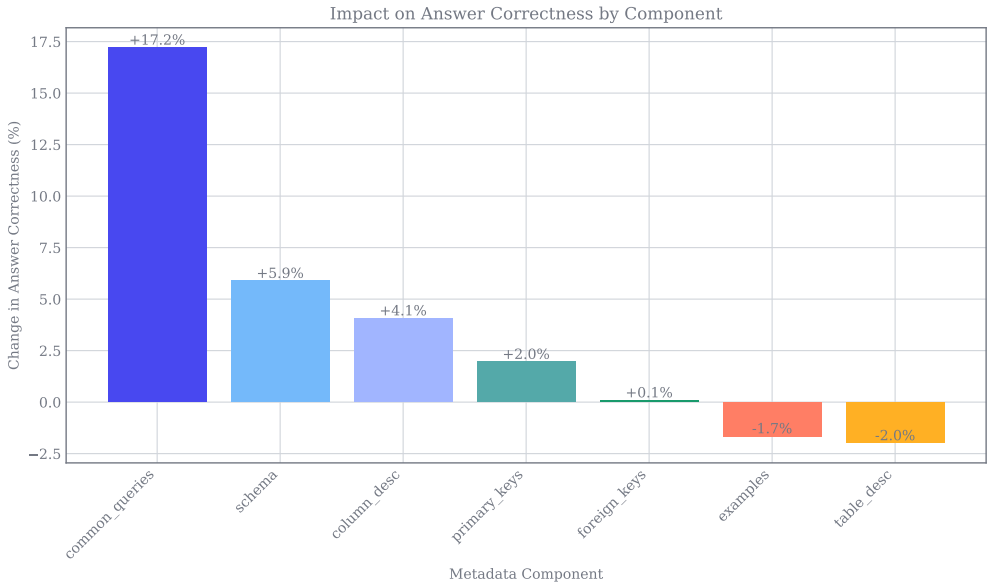

Figure 9: Impact of individual components on correctness

The decrease in correctness from examples and `table_desc` indicates that the large amount of data included for these is causing context overload in which the AI demonstrates reduced efficacy in extracting meaning. Noting as above that `table_desc` may be of more importance in schemas without clear semantic naming schemes.

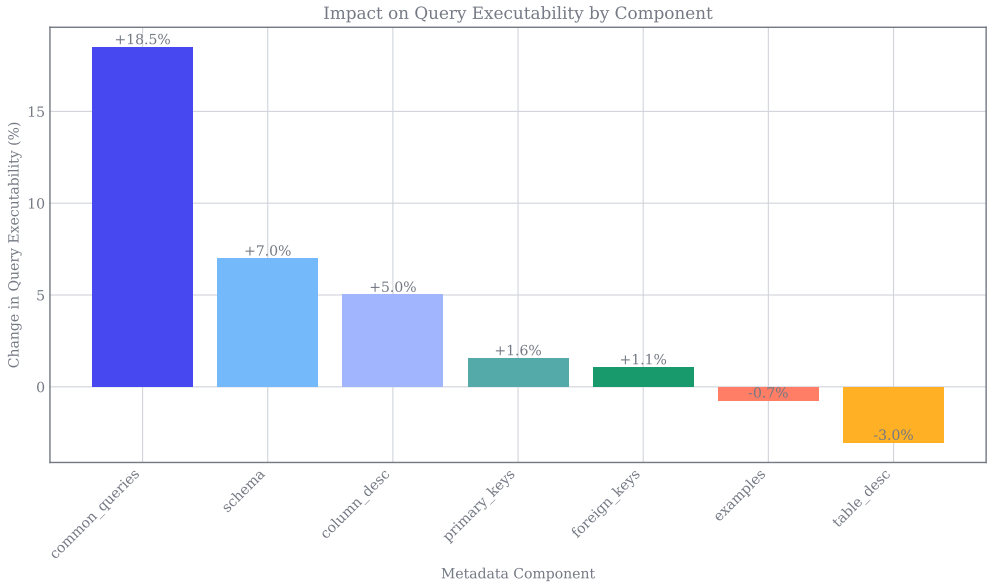

Figure 10: Impact of individual components on query executability

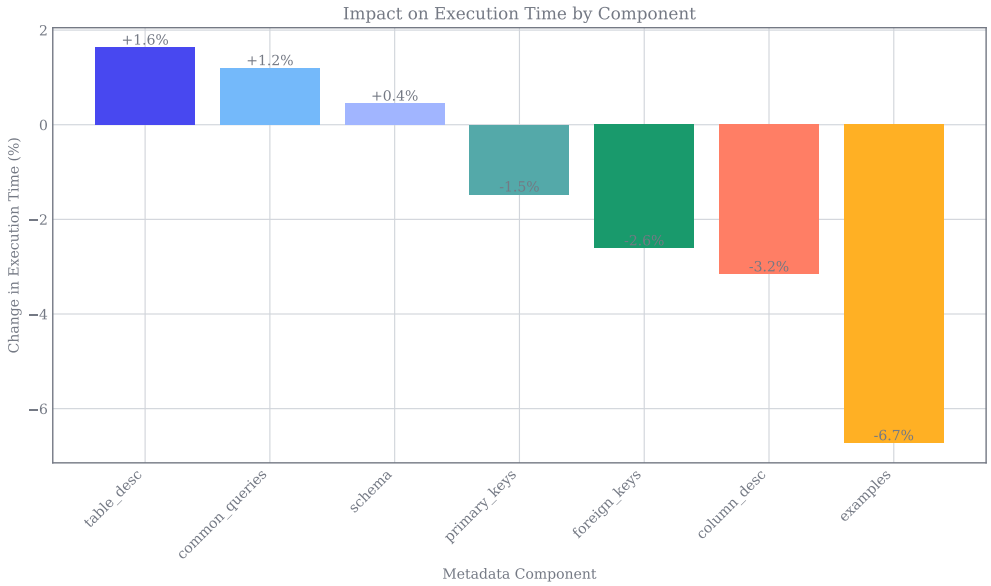

Figure 11: Impact of individual components on execution time

The LLM appears to struggle with decoding the tabular example row data and how to apply it.

Table 3: Individual Component Impact Analysis

Component	Individual Impact	Correctness	Execution Time	Executability
Common Queries	**73.68%**	**17.21%**	1.20%	**18.50%**
Primary Keys	65.26%	1.96%	-1.47%	1.56%
Column Descriptions	61.05%	4.07%	-3.16%	5.02%
Schema	60.53%	5.89%	0.45%	6.99%
Table Descriptions	56.05%	-1.99%	**1.63%**	-3.04%
Foreign Keys	54.21%	0.09%	-2.61%	1.07%
Examples	41.05%	-1.65%	-6.73%	-0.74%

4.7 Component-wise Analysis

4.7.1 Common Queries (Δ: +73.7%)

- Most impactful individual component
- Provides essential query patterns
- Critical for complex SQL generation

4.7.2 Primary Keys (Δ: +65.3%)

- Second most effective individual component
- Important for unique identification
- Crucial for aggregation queries

4.7.3 Column Descriptions (Δ: +61.1%)

- Strong impact on semantic understanding
- Helps with field selection
- Important for field selection accuracy

4.7.4 Schema Information (Δ: +60.5%)

- Foundational component
- Provides essential structural understanding
- Critical for table relationships

4.7.5 Table Descriptions (Δ: +56.1%)

- Moderate improvement
- Helps with table selection
- Provides context for joins

4.7.6 Foreign Keys (Δ: +54.2%)

- Important for relationship understanding
- Enables proper table joining
- Crucial for multi-table queries

4.7.7 Examples (Δ: +41.1%)

- Least effective individual component
- May introduce extraneous information
- More effective when utilized as supplementary information

5 Discussion

5.1 Key Insights

1. **Optimal Configuration:** The combination of schema information, column descriptions, table descriptions, foreign keys, primary keys, and common queries provides the best balance of performance across all metrics.
2. **Component Synergy:** While individual components show significant improvements, their combination achieves higher performance than any individual component.
3. **Diminishing Returns:** The performance improvements from adding metadata components follow a logarithmic curve, with significant gains for the first three or four components followed by minimal improvements.

5.2 Practical Implications for System Design

Our findings suggest that when implementing text-to-SQL systems the optimal metadata strategy is:

5.2.1 Essential Components

- Implement common queries as the foundational layer
- Add primary keys as the second priority
- Include column descriptions for semantic understanding

5.2.2 Optional Components

- Add schema information and foreign keys for complex queries
- Consider table descriptions for specific use cases
- Use examples sparingly and only when needed

5.2.3 Implementation Order

- Start with common queries + primary keys (highest ROI)
- Add column descriptions for semantic enhancement
- Evaluate need for additional components based on specific use cases

6 Conclusion

This study provides empirical evidence for optimal metadata configuration in text-to-SQL systems. The success of configurations including common queries, primary keys, and column descriptions (84.5% correctness score) suggests that focusing on these core components is more effective than comprehensive metadata inclusion. Our findings provide clear guidance for both academic research directions and commercial implementation strategies.

Our results also highlight the importance of carefully selecting metadata components based on their individual and combined impact on performance. The clear hierarchy of component effectiveness (`common_queries` $\cup$ `primary_keys` $\cup$ `column_desc` $\cup$ `schema` $\cup$ `table_desc` $\cup$ `foreign_keys` $\cup$ `examples`) provides a practical roadmap for implementing text-to-SQL systems with optimal performance characteristics.

Through our comprehensive analysis of 128 different metadata combinations across 19 queries, we have demonstrated that while metadata richness generally correlates with improved performance, there exists a practical limit to these improvements. The diminishing returns observed beyond three to four components suggest that system designers should focus on implementing a carefully selected subset of metadata types rather than attempting to include all possible metadata configurations.

7 Future Research Directions

While our study provides valuable insights into metadata configuration impacts on text-to-SQL system performance, several promising areas warrant further investigation. We identify six key directions for future research:

7.1 Cost-Performance Analysis of Metadata Components

Future studies should investigate the relationship between metadata components and computational costs, particularly focusing on computational token consumption and generation expenses. Understanding this relationship will enable organizations to optimize their text-to-SQL implementations for both performance and cost-effectiveness. Research should examine whether certain metadata combinations provide better cost-benefit ratios than others.

7.2 Impact on Non-Standard Schema Structures

Our research primarily focused on well-defined semantic schemas. Further investigation is needed to understand how metadata configuration effectiveness varies with schemas that lack clear semantic definitions or exhibit irregular structures. This research could provide valuable insights for organizations dealing with legacy or non-standardized database systems.

7.3 Vector-Enhanced Metadata Approaches

The inclusion of vectorized actual data, such as distinct column value enumerations, presents a compelling direction for investigation. Research should examine how incorporating this additional context affects SQL generation correctness and execution performance. Particular attention should be paid to the trade-offs between increased context richness and computational overhead.

7.4 Cross-Model Performance Analysis

While our study focused on specific LLM implementations, comprehensive comparative analysis across different language models could reveal important variations in metadata utilization patterns. This research could help identify whether certain models are better suited for specific types of queries or metadata configurations.

7.5 Domain-Specific Model Optimization

Investigation into the feasibility of small language models specifically optimized for SQL generation could provide valuable insights into potential efficiency improvements. Research should examine whether such specialized models could offer comparable performance to general-purpose LLMs while requiring fewer computational resources.

7.6 Context Capacity Evolution Impact

As context capacity windows continue to expand and the cost-performance ratio of LLM operations improves, research should explore the implications for metadata strategy. A particularly interesting question is whether future technological advances might enable direct submission of complete datasets to LLMs, potentially eliminating the need for complex metadata configurations. This research should consider both technical feasibility and practical implications.

These research directions could significantly advance our understanding of text-to-SQL system optimization and provide valuable insights for future implementations. As the field continues to evolve rapidly, addressing these questions will become increasingly important for both academic understanding and practical applications.

A Detailed Results

A.1 Overall Performance by Configuration

Table 4: Performance Metrics Across Different Metadata Configurations

Configuration	Answer Correctness	Execution Time (s)	Query Executable
base	0.000	3.118	0.000
table_desc ∪ examples	0.321	2.459	0.474
table_desc ∪ examples ∪ foreign_keys	0.389	2.827	0.526
examples	0.411	3.484	0.579
table_desc ∪ foreign_keys	0.437	2.644	0.526
table_desc ∪ examples ∪ foreign_keys ∪ primary_keys	0.437	3.170	0.684
column_desc ∪ examples ∪ foreign_keys	0.489	3.358	0.632
schema ∪ table_desc ∪ examples ∪ foreign_keys ∪ primary_keys	0.503	3.307	0.737
column_desc ∪ table_desc ∪ examples	0.511	3.023	0.684
column_desc ∪ table_desc ∪ examples ∪ foreign_keys ∪ primary_keys	0.521	3.492	0.684
examples ∪ foreign_keys	0.526	4.086	0.737
table_desc ∪ primary_keys	0.542	2.870	0.737
foreign_keys	0.542	3.044	0.737
schema ∪ column_desc ∪ table_desc ∪ examples ∪ foreign_keys ∪ primary_keys	0.542	3.400	0.737
table_desc ∪ examples ∪ primary_keys	0.547	3.107	0.684
foreign_keys ∪ primary_keys	0.553	3.366	0.737
column_desc ∪ table_desc	0.558	2.805	0.684
table_desc	0.561	2.666	0.684
examples ∪ foreign_keys ∪ primary_keys	0.579	2.872	0.737
schema ∪ examples ∪ primary_keys	0.579	3.626	0.737
schema ∪ table_desc ∪ examples ∪ foreign_keys	0.603	3.314	0.789
schema	0.605	2.981	0.842
examples ∪ primary_keys	0.605	3.769	0.737

(Continued on next page)

(Continued from previous page)

Configuration	Answer Correctness	Execution Time (s)	Query Executable
column_desc	0.611	2.945	0.842
schema ∪ column_desc ∪ table_desc ∪ examples	0.616	3.018	0.789
column_desc ∪ table_desc ∪ foreign_keys	0.616	3.182	0.789
schema ∪ column_desc ∪ examples ∪ primary_keys	0.618	3.534	0.842
column_desc ∪ table_desc ∪ primary_keys	0.621	2.886	0.737
schema ∪ column_desc ∪ table_desc ∪ examples ∪ foreign_keys	0.621	3.144	0.789
column_desc ∪ examples	0.621	3.363	0.789
schema ∪ column_desc ∪ table_desc ∪ examples ∪ primary_keys	0.624	4.047	0.842
column_desc ∪ table_desc ∪ examples ∪ primary_keys	0.626	3.131	0.789
column_desc ∪ foreign_keys ∪ primary_keys	0.626	3.190	0.789
column_desc ∪ table_desc ∪ foreign_keys ∪ primary_keys	0.632	3.027	0.789
column_desc ∪ table_desc ∪ examples ∪ foreign_keys	0.632	3.792	0.842
schema ∪ table_desc ∪ foreign_keys	0.637	2.840	0.789
column_desc ∪ examples ∪ foreign_keys ∪ primary_keys	0.642	3.238	0.842
schema ∪ examples ∪ foreign_keys ∪ primary_keys	0.647	2.948	0.789
schema ∪ table_desc ∪ primary_keys	0.647	3.036	0.789
table_desc ∪ foreign_keys ∪ primary_keys	0.650	3.070	0.842
primary_keys	0.653	2.543	0.842
schema ∪ column_desc ∪ examples ∪ foreign_keys ∪ primary_keys	0.653	3.793	0.842
schema ∪ column_desc ∪ primary_keys	0.658	2.816	0.842
schema ∪ table_desc ∪ examples	0.658	2.957	0.842
schema ∪ table_desc	0.661	2.759	0.842
schema ∪ column_desc ∪ table_desc	0.668	3.132	0.842
schema ∪ examples ∪ foreign_keys	0.668	3.367	0.895
schema ∪ foreign_keys ∪ primary_keys	0.674	3.109	0.947
schema ∪ column_desc ∪ foreign_keys ∪ primary_keys	0.676	3.216	0.895
schema ∪ table_desc ∪ foreign_keys ∪ primary_keys	0.679	3.106	0.842
schema ∪ table_desc ∪ examples ∪ primary_keys	0.687	3.281	0.789

(Continued on next page)

(Continued from previous page)

Configuration	Answer Correctness	Execution Time (s)	Query Executable
table_desc ∪ foreign_keys ∪ common_queries	0.689	2.868	0.842
schema ∪ column_desc ∪ examples	0.689	3.314	0.947
schema ∪ column_desc ∪ foreign_keys	0.692	2.794	0.842
schema ∪ column_desc	0.697	2.947	0.895
schema ∪ column_desc ∪ table_desc ∪ foreign_keys ∪ primary_keys	0.700	2.927	0.895
schema ∪ column_desc ∪ table_desc ∪ foreign_keys	0.700	3.014	0.895
schema ∪ primary_keys	0.711	3.326	0.895
schema ∪ table_desc ∪ examples ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.713	3.539	0.947
column_desc ∪ foreign_keys	0.713	3.699	0.895
schema ∪ column_desc ∪ examples ∪ foreign_keys	0.721	3.590	0.895
column_desc ∪ primary_keys	0.724	3.256	0.947
table_desc ∪ examples ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.726	2.657	0.947
column_desc ∪ examples ∪ primary_keys	0.726	3.499	0.895
schema ∪ foreign_keys	0.732	2.608	0.947
column_desc ∪ table_desc ∪ primary_keys ∪ common_queries	0.732	2.950	0.895
schema ∪ column_desc ∪ table_desc ∪ examples ∪ common_queries	0.732	3.539	0.947
schema ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.734	2.767	0.947
common_queries	0.737	2.414	0.947
table_desc ∪ primary_keys ∪ common_queries	0.737	2.723	0.895
column_desc ∪ table_desc ∪ examples ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.739	2.878	0.947
schema ∪ column_desc ∪ table_desc ∪ primary_keys	0.739	3.005	0.895
examples ∪ common_queries	0.742	2.991	0.947
schema ∪ examples	0.745	3.238	0.947
table_desc ∪ common_queries	0.753	2.919	0.947
column_desc ∪ table_desc ∪ foreign_keys ∪ common_queries	0.753	3.116	1.000
schema ∪ column_desc ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.753	3.420	0.947
column_desc ∪ table_desc ∪ examples ∪ foreign_keys ∪ common_queries	0.758	3.179	1.000
schema ∪ table_desc ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.758	3.264	0.947

(Continued on next page)

(Continued from previous page)

Configuration	Answer Correctness	Execution Time (s)	Query Executable
column_desc ∪ table_desc ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.758	3.654	0.895
column_desc ∪ examples ∪ foreign_keys ∪ common_queries	0.758	4.633	1.000
schema ∪ column_desc ∪ examples ∪ foreign_keys ∪ common_queries	0.763	2.848	0.947
schema ∪ column_desc ∪ table_desc ∪ examples ∪ primary_keys ∪ common_queries	0.763	2.856	0.947
schema ∪ examples ∪ common_queries	0.763	2.984	0.947
column_desc ∪ table_desc ∪ examples ∪ common_queries	0.763	3.183	0.947
examples ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.763	3.400	0.947
foreign_keys ∪ primary_keys ∪ common_queries	0.766	3.002	0.947
schema ∪ primary_keys ∪ common_queries	0.768	2.654	0.947
foreign_keys ∪ common_queries	0.768	2.830	0.947
schema ∪ column_desc ∪ table_desc ∪ common_queries	0.768	3.023	1.000
schema ∪ table_desc ∪ examples ∪ common_queries	0.768	3.304	0.947
table_desc ∪ examples ∪ primary_keys ∪ common_queries	0.768	3.382	1.000
schema ∪ column_desc ∪ common_queries	0.774	2.610	1.000
schema ∪ foreign_keys ∪ common_queries	0.774	2.884	1.000
table_desc ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.774	2.947	0.947
schema ∪ column_desc ∪ table_desc ∪ examples ∪ foreign_keys ∪ common_queries	0.776	3.138	1.000
column_desc ∪ primary_keys ∪ common_queries	0.779	2.880	0.947
schema ∪ examples ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.779	3.438	0.947
column_desc ∪ examples ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.779	3.652	0.947
schema ∪ table_desc ∪ examples ∪ primary_keys ∪ common_queries	0.782	3.200	0.947
schema ∪ column_desc ∪ table_desc ∪ foreign_keys ∪ common_queries	0.784	3.474	1.000
column_desc ∪ examples ∪ primary_keys ∪ common_queries	0.784	3.540	1.000
schema ∪ column_desc ∪ examples ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.787	3.037	1.000
column_desc ∪ foreign_keys ∪ common_queries	0.787	3.087	1.000
primary_keys ∪ common_queries	0.789	2.800	0.947
column_desc ∪ common_queries	0.789	3.037	1.000
schema ∪ examples ∪ primary_keys ∪ common_queries	0.789	3.337	0.947

(Continued on next page)

(Continued from previous page)

Configuration	Answer Correctness	Execution Time (s)	Query Executable
schema ∪ column_desc ∪ table_desc ∪ primary_keys ∪ common_queries	0.789	3.385	0.947
schema ∪ table_desc ∪ primary_keys ∪ common_queries	0.792	2.823	0.947
column_desc ∪ table_desc ∪ common_queries	0.795	2.875	0.947
schema ∪ table_desc ∪ foreign_keys ∪ common_queries	0.795	2.981	0.947
schema ∪ table_desc ∪ examples ∪ foreign_keys ∪ common_queries	0.795	3.042	0.947
column_desc ∪ table_desc ∪ examples ∪ primary_keys ∪ common_queries	0.795	3.234	1.000
schema ∪ common_queries	0.800	2.369	1.000
examples ∪ primary_keys ∪ common_queries	0.800	3.099	1.000
schema ∪ column_desc ∪ primary_keys ∪ common_queries	0.800	3.156	0.947
schema ∪ table_desc ∪ common_queries	0.805	2.969	1.000
schema ∪ column_desc ∪ table_desc ∪ examples ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.808	3.120	1.000
schema ∪ examples ∪ foreign_keys ∪ common_queries	0.808	3.172	1.000
examples ∪ foreign_keys ∪ common_queries	0.808	3.687	1.000
schema ∪ column_desc ∪ examples ∪ common_queries	0.811	3.021	1.000
schema ∪ column_desc ∪ examples ∪ primary_keys ∪ common_queries	0.811	3.223	1.000
column_desc ∪ examples ∪ common_queries	0.816	3.142	1.000
table_desc ∪ examples ∪ common_queries	0.816	3.269	1.000
column_desc ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.824	2.953	1.000
schema ∪ column_desc ∪ foreign_keys ∪ common_queries	0.826	3.018	1.000
table_desc ∪ examples ∪ foreign_keys ∪ common_queries	0.829	3.403	1.000
schema ∪ column_desc ∪ table_desc ∪ foreign_keys ∪ primary_keys ∪ common_queries	0.845	2.946	1.000

B Error Analysis

B.1 Common Failure Patterns

Query Execution Failures (as a percentage of all executions)

- Syntax Errors: 3.2%
- Runtime Errors: 5.8%
- Timeout Errors (retried): 1.3%

Answer Correctness Failure Causes

- Wrong Join Conditions: 12.4%
- Incorrect Aggregations: 8.7%
- Missing Filters: 6.9%

B.2 Performance Bottlenecks

Execution Time

- Complex Joins: +1.23s
- Multiple Aggregations: +0.87s
- Subqueries: +1.45s

Memory Usage

- Large Result Sets: +25%
- Multiple CTEs: +15%
- Window Functions: +20%

A Evaluation Questions

The following questions were used in our evaluation, organized by category and complexity level.

Revenue Analysis (Medium Complexity)

1. What is the total revenue by category?
2. List the top 5 products by revenue

Inventory (Simple Complexity)

1. Which products have less than 10 units in stock?

Order Analysis (Complex Complexity)

1. What is the average order value and number of orders per month in 2024?
2. What is the seasonal trend in order volume and revenue by month?

Customer Analysis (Medium/Complex Complexity)

1. Who are our top 5 customers by revenue, and what is their total order count?
2. What is the customer retention rate by quarter?

Product Analysis (Medium/Complex Complexity)

1. Which products have never been ordered?
2. Find products that are frequently ordered together in the same order
3. Which products have the highest profit margin percentage?

Employee Performance (Medium/Complex Complexity)

1. Find the monthly sales totals for each employee in the last 3 months
2. What is the employee performance ranking based on order processing time and revenue?

Pricing Analysis (Medium/Complex Complexity)

1. What is the average discount percentage offered for each category, and how does it correlate with the total revenue?
2. What is the impact of discount levels on order quantity?
3. How does the average order quantity change with different discount levels by category?

Regional Analysis (Complex Complexity)

1. Which territories have the highest revenue per customer?

Shipping Analysis (Medium/Complex Complexity)

1. Which shipping companies have the best on-time delivery performance?
2. Find orders with unusually high shipping costs compared to their category average

B LLM Prompts and Evaluation Protocol

B.1 Text-to-SQL Generation Prompt Template

```
Given the following database, write a SQL query in the SQLite3 dialect to answer
this question.

Schema:
CREATE TABLE Categories (
    CategoryID INTEGER PRIMARY KEY,
    CategoryName TEXT,
    Description TEXT
);
// ... rest of schema ...

Table Descriptions:
- Categories: Contains product category information including names and descriptions
- Products: List of products sold by the company
// ... other table descriptions ...

Column Descriptions:
- Categories.CategoryID: Unique identifier for each product category
- Categories.CategoryName: Name of the product category
// ... other column descriptions ...

Example Data:

Categories sample rows:
{"CategoryID": 1, "CategoryName": "Beverages", "Description": "Soft drinks, coffees,
    teas, beers"}
{"CategoryID": 2, "CategoryName": "Condiments", "Description": "Sweet and savory
    sauces, relishes, spreads"}
// ... other example rows ...

Foreign Keys:
- Products.CategoryID → Categories.CategoryID
- OrderDetails.ProductID → Products.ProductID
// ... other foreign keys ...

Primary Keys:
- Categories: CategoryID
- Order Details: (OrderID, ProductID) composite key
// ... other primary keys ...

Common Queries:
```

```
Top selling products:
SELECT p.ProductName, SUM(od.Quantity) as TotalQuantity
FROM Products p
JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY p.ProductID, p.ProductName
ORDER BY TotalQuantity DESC
LIMIT 5
// ... other example queries ...

Question: What is the total revenue by category?
```

B.2 Evaluation Prompt

```
You are an SQL expert. Analyze these two SQL queries and their results to determine:

1. Whether the generated query correctly answers the original question (0.0-1.0)
2. Whether the results are logically equivalent for answering the question (0.0-1.0)

Original Question:
What is the total revenue by category?

Schema:
CREATE TABLE Categories (
    CategoryID INTEGER PRIMARY KEY,
    CategoryName TEXT,
    Description TEXT
);
// ... rest of schema ...

Generated SQL:
SELECT c.CategoryName, SUM(od.UnitPrice * od.Quantity) as Revenue
FROM Categories c
JOIN Products p ON c.CategoryID = p.CategoryID
JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY c.CategoryName
ORDER BY Revenue DESC

Query Executable: Success
Time: 0.023s
Query Execution Results: [('Beverages', 286527.0), ('Seafood', 264172.0),
('Dairy Products', 234516.0)] ...

Expected SQL:
SELECT c.CategoryName, ROUND(SUM(od.UnitPrice * od.Quantity * (1 - od.Discount)), 2)
```

```
as TotalRevenue
FROM Categories c
JOIN Products p ON c.CategoryID = p.CategoryID
JOIN [Order Details] od ON p.ProductID = od.ProductID
GROUP BY c.CategoryName
ORDER BY TotalRevenue DESC

Query Executable: Success
Time: 0.025s
Query Execution Results: [('Beverages', 267868.21), ('Seafood', 246802.83),
('Dairy Products', 221573.82)] ...
```

Metadata Configuration Format

```
{
  "schema": boolean,        // Include table schemas
  "table_desc": boolean,    // Include table descriptions
  "column_desc": boolean,   // Include column descriptions
  "examples": boolean,      // Include common example queries
  "foreign_keys": boolean,  // Include foreign key relationships
  "primary_keys": boolean,  // Include primary key information
  "golden_queries": boolean // Include frequently used queries
}
```

The evaluation protocol was automated using a testing framework that systematically applied each metadata configuration to all test questions, collecting metrics for analysis. Results were aggregated and analyzed using statistical methods to ensure reliability and significance.

References

Johnson, S., Lee, M., & Chen, W. (2023). A survey on metadata augmentation and management in text-to-sql systems. *arXiv preprint arXiv:2307.05074*.

Nguyen, I., Gurung, B., & Bartels, M. (2023). Using generative ai to query large bi tables: Our findings. *Medium*. https://medium.com/deepset-ai/using-generative-ai-to-query-large-bi-tables-our-findings-e215070bfa5a

Singh, A., Shetty, A., Ehtesham, A., Kumar, S., & Khoei, T. T. (2024). A survey of large language model-based generative ai for text-to-sql: Benchmarks, applications, use cases, and challenges. *arXiv preprint arXiv:2412.05208*.

Sun, R., Arik, S. Ö., Muzio, A., Miculicich, L., Gundabathula, S., Yin, P., Dai, H., Nakhost, H., Sinha, R., Wang, Z., & Pfister, T. (2024). Sql-palm: Improved large language model adaptation for text-to-sql. *arXiv preprint arXiv:2306.00739*.

White, J., et al. (2022). Northwind-sqlite3: Sqlite3 version of microsoft's northwind database. https://github.com/jpwhite3/northwind-SQLite3

Zhang, T., Chen, C., Liao, C., Wang, J., Zhao, X., Yu, H., Wang, J., Li, J., & Shi, W. (2024). Sqlfuse: Enhancing text-to-sql performance through comprehensive llm synergy. *arXiv preprint arXiv:2407.14568*.