SPORTSQL: An Interactive System for Real-Time Sports Reasoning and Visualization

Sebastian Martinez **Naman Ahuja** **Fenil Bardoliya** **Suparno Roy Chowdhury**
Chris Bryan **Vivek Gupta**
Arizona State University
{sjmart28, nahuja11, fbardoli, srchowd3, cbryan16, vgupt140}@asu.edu

Abstract

We present a modular, interactive system SPORTSQL for natural language querying and visualization of dynamic sports data, with a focus on the English Premier League (EPL). The system translates user questions into executable SQL over a live, temporally indexed database constructed from real-time Fantasy Premier League (FPL) data. It supports both tabular and visual outputs, leveraging symbolic reasoning capabilities of Large Language Models (LLMs) for query parsing, schema linking, and visualization selection. To evaluate system performance, we introduce the **D**ynamic **S**port **Q**uestion **A**nswering benchmark (DSQABENCH), comprising 1,700+ queries annotated with SQL programs, gold answers, and database snapshots. Our demo highlights how non-expert users can seamlessly explore evolving sports statistics through a natural, conversational interface.

1 Introduction

What if a soccer fan could ask, *"How did Mohamed Salah's scoring performance trend over the last five seasons?"* or *"Which midfielders in the Premier League are the most creative this season?"* and instantly receive not only a precise answer but also a dynamic visualization, grounded in up-to-date, real-world data?

Large language models (LLMs) have shown remarkable progress in translating natural language into executable programs, such as SQL. However, most existing systems are designed for static, domain-specific datasets. In contrast, domains like sports are inherently dynamic and structurally complex: match outcomes, player statistics, team formations, and injury reports evolve daily across multiple interlinked and semi-structured tables. Querying such data effectively requires compositional, temporal, and relational reasoning, along with the ability to operate over continuously changing schemas and distributed sources.

We introduce SPORTSQL, a fully automated system for Dynamic Sports Question Answering (DSQA), enabling users to pose rich natural language queries over live sports data and receive grounded, executable, and often visual responses. SPORTSQL operates through a modular pipeline: it begins by scraping and normalizing dynamic data and transforming it into a unified, temporally indexed relational database. Given a user question, the system uses only the schema (not the data itself) to prompt an LLM to generate symbolic SQL queries, making the approach scalable and robust to changes in content (Kulkarni et al., 2025). When appropriate, SPORTSQL also generates visualization code (in matplotlib, seaborn) to produce bar charts, timelines, or other graphical responses.

For instance, a user might ask, *"Compare Arsenal's goals scored in home vs away matches"* or *"List forwards with at least ten goals and five assists."* SPORTSQL retrieves accurate answers by executing SQL over the latest data, rather than relying on potentially outdated or hallucinated information from pretrained language models (Kulkarni and Srikumar, 2025). To evaluate the effectiveness of the system, we introduce Dynamic Sports Question Answering Benchmark (DSQABENCH), a new benchmark containing over 1700 questions that span various soccer metrics, reasoning types, and output formats. Each question is paired with its corresponding SQL program, gold answer, and the database snapshot at the time of execution. We further provide a type-aware evaluation framework that supports multiple answer formats, schema-only SQL generation, and fine-grained error analysis to assess system performance under dynamic conditions. Our contributions are threefold:

- We introduce the task of Dynamic Sports Question Answering and present SPORTSQL, a modular and interpretable system that enables real-time, schema-driven symbolic rea-

Proceedings of The 14th International Joint Conference on Natural Language Processing and The 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics: System Demonstrations, pages 94–101
December 20-24, 2025

soning and dynamic visualization over evolving sports databases.

- We construct and release DSQABENCH, the first benchmark of executable sports queries paired with live data, supporting multiple answer modalities.
- We develop a type-aware evaluation framework with support for diverse answer formats (textual, numeric, tabular, visual), schema-only SQL generation, and fine-grained error analysis to assess symbolic QA systems over dynamic content.

We invite the readers to explore SportsSQL's functionalities at the following links:

- Code & Data: https://github.com/coral-lab-asu/SportSQL
- Main Demo Video: https://youtu.be/xqUyiA-R6aI
- Try it out: https://coral-lab-asu.github.io/SportsSQL

Although SPORTSQL is designed for sports, its architecture is general and can extend to other dynamic, structured domains such as finance, healthcare, or elections, where users seek timely, accurate insights from evolving data.

2 SPORTSQL Architecture

SPORTSQL translates free-form user queries into executable answers via a tightly integrated, multistage pipeline. The system operates over a live, dynamically updated EPL database, refreshed periodically via cronjobs and at runtime based on query requirements. Upon receiving a natural language query, the system first performs entity grounding by executing SQL lookups against curated reference tables (e.g., teams, players), mapping surface forms to canonical entities. Conditioned on this context and the database schema, it generates an executable SQL query, which is run on the live database to produce a structured result. If the query involves visual reasoning (e.g., comparisons, trends, rankings), the output is forwarded to a visualization agent, which selects an appropriate chart type and returns self-contained Python code (Matplotlib + Seaborn) to render the plot. The full workflow is outlined below.

1. Database Streaming Our system ingests data from the public *Fantasy Premier League* (FPL) API,[1] which offers structured, frequently updated endpoints covering players, teams, fixtures, and per-match statistics. After normalization and deduplication, the data is stored in a MariaDB backend.

Hybrid Storage Strategy Storing full historical data for every player would require ∼2,400 tables per season (3 per player × 800 players), resulting in a bloated schema and largely idle data. To balance granularity with efficiency, we adopt a two-tiered storage design:

- **Query-agnostic tables**: Core relations (`players`, `teams`, `fixtures`) that evolve predictably week-by-week. These are updated nightly via cronjob to maintain freshness.
- **Query-dependent tables**: Fine-grained views (e.g., "past 5 games", "next 3 fixtures") fetched on demand from the FPL API. These are materialized in memory for the duration of a query and discarded after use.

This hybrid architecture ensures (i) *freshness* via automated updates, (ii) *coverage* through just-in-time API access, and (iii) *efficiency* by limiting persistent storage. Figure 1 illustrates the relational schema and data flow.

2. Entity Recognition User queries often contain abbreviations, nicknames, or informal spellings (e.g., *"CR7"* for *Cristiano Ronaldo*, *"Donatello"* for *Kylian Mbappé*), making exact string matching unreliable. Additionally, the LLM operates only over the database schema and lacks direct access to cell-level values. To resolve entity mentions, we employ a prompt-guided procedure. The prompt instructs the LLM to: (i) use domain knowledge to infer canonical player or team names, and (ii) generate a case-insensitive wildcard SQL query over reference tables. The database returns a filtered set of candidate rows with unique IDs, which are retained as the resolved entity identifiers.

3. SQL Generation and Execution Given the resolved entity identifiers, we prompt a large language model to generate an executable SQL query. The model is provided with: (i) the user question, (ii) the set of resolved primary keys, and (iii) the database schema, along with targeted instructions to mitigate common pitfalls:

- **Table hints**: e.g., `players` is preferred for individual statistics

[1] https://fantasy.premierleague.com/

players		teams	
player_id	int (PK)	team_id	int (PK)
web_name	varchar	team_name	varchar
player_position	varchar	position	int
team_id	int	played	int
goals_scored	int	win	int
assists	int	draw	int
minutes	int	loss	int

fixtures	
game_id	int (PK)
gw	int
finished	boolean
team_h_name	varchar
team_a_name	varchar
kickoff_time	varchar

player_history	
season_name	varchar (PK)
goals_scored	int
assists	int
minutes	int
yellow_cards	int
red_cards	int
saves	int

player_past	
player_id	int
event	int (PK)
goals_scored	int
assists	int
minutes	int
yellow_cards	int
red_cards	int

player_future	
event	int (PK)
event_name	varchar
team_h_name	varchar
team_a_name	varchar
is_home	boolean
difficulty	int
kickoff_time	varchar

Figure 1: DB schema, all tables shown, not all columns. Here, PK represent primary key.

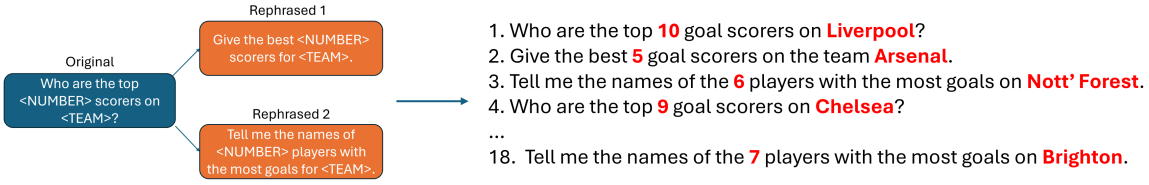


Figure 2: Sample Question Creation Expansion

- **Synonym mappings**: e.g., "team position" ↔ `league_rank`
- **Column cautions**: e.g., penalty saves are almost always non-zero for goalkeepers only
- **Derived-field formulas**: e.g., `form` is the 30-day average of match points
- **Scale explanations**: e.g., `strength` ranges from 1 (weakest) to 5 (strongest)

These prompt elements help ensure syntactic correctness and reduce semantic errors arising from natural language variability.

SQL Execution. The generated SQL is parsed and executed against the dynamic MariaDB store. If the query references a non-materialized *query-dependent* table (e.g., a player's upcoming fixtures), the system issues a just-in-time API call to fetch the necessary data, loads it into an in-memory buffer, and re-executes the query. The temporary table is discarded post-aggregation, ensuring the persistent database remains lightweight.

4. Visual Output Generation Some information needs are better served through visualizations than text. To support this, the system automatically generates plots when either: (i) the user explicitly requests a "plot," "graph," or "trend," or (ii) the output dataframe exhibits structures—such as multi-season time series or long categorical rankings—that benefit from visual interpretation. For example, the query *"Plot a line graph of Kylian Mbappé's goal totals over the past five seasons"* produces a line chart with seasons on the x-axis and goals on the y-axis, revealing temporal trends. Similarly, the query *"Which five teams recorded the highest average possession in the 2024–25 campaign?"*—though not explicitly visual—triggers a horizontal bar chart ranking clubs by possession.

When comparative or temporal reasoning is detected, the result and original query are passed to a secondary code-generating LLM, which returns self-contained `Matplotlib` code (e.g., line plots for trends, bar charts for rankings). A validation layer ensures the dataframe referenced in the code matches the SQL output byte-for-byte; any mismatch triggers automatic re-querying.

This architecture enables near real-time visual responses, maintains the persistent database under 5GB, and supports fine-grained, player-level analytics without compromising freshness or correctness. Figure 3 presents an overview of the full system pipeline.

3 DSQABENCH Benchmark

To evaluate the SPORTSQL system, we introduce the **D**ynamic **S**port **Q**uestion **A**nswering benchmark (DSQABENCH), designed to assess natural language interfaces over dynamic, multi-relational sports data.

Query Creation. We construct a diverse set of natural language questions targeting various schema elements and reasoning skills. The process begins with manually written question templates, each rephrased to capture linguistic variation. Templates contain placeholders (e.g., team names, numerical thresholds), which are instantiated using real-world entities and context-appropriate values. This approach balances lexical diversity with semantic control. Additionally, we include a set of manually crafted, challenging questions to probe complex and multi-hop reasoning. An illustration of this process is shown in Figure 2.

Answer Annotation. Each question is paired with a manually authored SQL query, serving as the gold standard. These queries are executed against the underlying MariaDB-based *"Fut-*

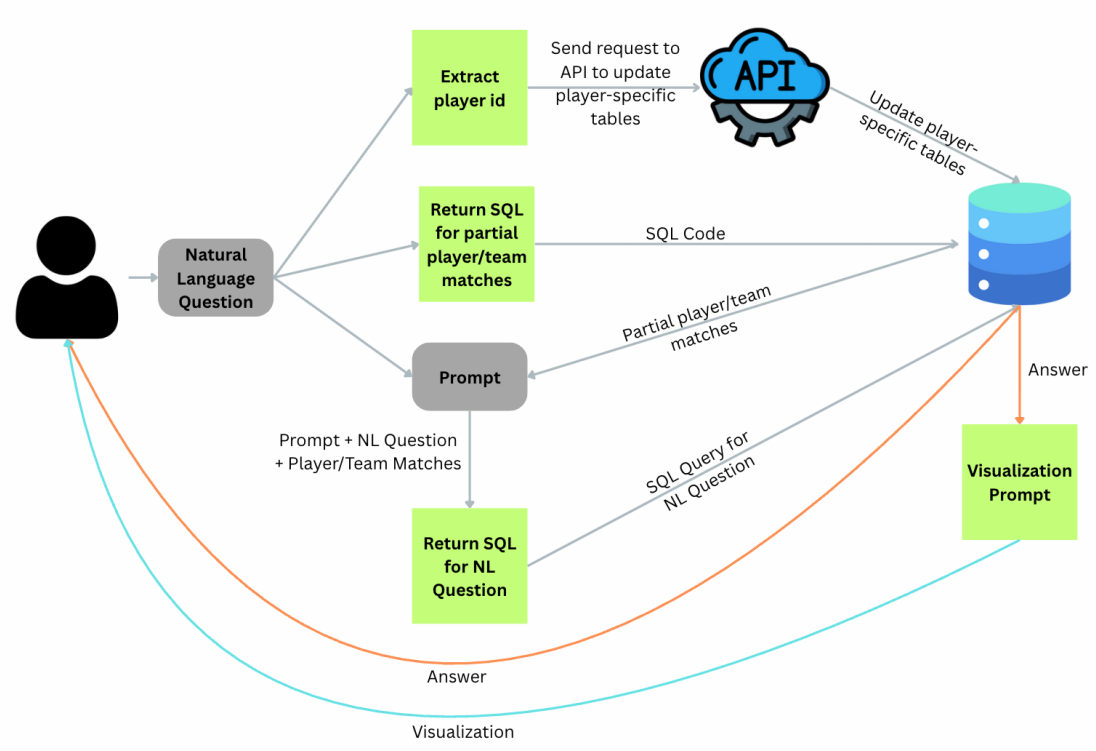


Figure 3: SportSQL Architecture and Workflow

SQL_FPL" database to verify correctness. This ensures high-quality supervision for evaluating both SQL generation and execution accuracy.

Dataset Statistics. DSQABENCH contains 1,793 questions derived from 180 base templates, each rephrased in three distinct ways and instantiated with real-world values. Among these:

- 1,395 questions yield scalar answers (e.g., strings, numbers); 398 require tabular outputs.
- 396 questions involve dynamic queries to player-specific tables via just-in-time API access:
  - `player_past`: 270 queries
  - `player_history`: 72 queries
  - `player_future`: 54 queries
- All questions are paired with manually validated SQL programs executable on the database.

DSQABENCH provides a rich and realistic benchmark for studying compositional generalization, schema coverage, and executable reasoning in sports QA systems.

4 Experiments and Analysis

Models. We evaluate two state-of-the-art LLMs: GPT-4O and GEMINI-2.0 FLASH. GEMINI-2.0-FLASH is selected for its balance of performance, latency, and cost, making it suitable for scalable deployment. GPT-4O is used to assess generalization. Both models use a temperature of 0.1 (for deterministic outputs) and a maximum token limit of 2048 (for reduced latency).

Evaluation Metrics. As the system produces both string and tabular outputs, we employ a type-aware evaluation. **String Answers**: Evaluated using exact match. We also employ LLMs as judges and prompt frontier models with the NL query, ground truth SQL, and system-generated SQL, and their SQL outputs, which models classify as equivalent/not equivalent. We use 3 frontier models: GPT-4o, GEMINI-2.5, and QWEN3-235b-A22B-INSTRUCT-2507, and take majority voting. **Table Answers**: Assessed using TABEVAL (Ramu et al., 2024), which converts tables into atomic natural language statements and computes pairwise entailment via ROBERTA-MNLI, yielding precision (Correctness), recall (Completeness), and F1 (Overall) scores.

4.1 Results and Analysis

Table 1 reports performance on both string and table-structured questions. The system achieves up to 80% exact-match accuracy and 0.75 macro-F1, indicating strong performance on structured

QA. GPT-4o consistently outperforms GEMINI-2.0 FLASH, with gains of 4.2 points in exact match and 0.05 in macro-F1. Completeness scores exceed correctness for both models, suggesting that relevant columns are more reliably identified than specific rows, a reflection of the higher complexity of row selection driven by SQL predicates. Moreover, we observe that the LLM-as-judge (majority voting) shows significantly higher accuracy than EM. This follows findings from previous works (Chandak et al., 2025) where deterministic evaluations over-penalize semantically coherent outputs. We observe multiple such cases, especially for more complex queries, where selecting extra/different columns in the output can lead to mis-evaluation. Hence, we use LLM as a Judge as our primary metric for further analysis.

Table 1: Model performance comparison on string and tables answered. Here, EM represents an Exact Match. Corr stands for Correctness, Comp stands for Completeness.

Model	String		Table (TabEval)		
	EM	LLM as Judge	Corr	Comp	Overall
Gemini-2.0	76.23	92.39	0.64	0.76	0.69
GPT-4o	80.48	93.82	0.70	0.81	0.75

4.2 Primitive-Based Analysis

To systematically assess performance across SQL query types, we annotate each ground-truth SQL template with a set of six reasoning primitives:

- **Calculate**: Arithmetic operations (`SUM`, `COUNT`, `AVG`, etc.)
- **Compare**: Value comparisons
- **Filter**: Conditional constraints (`WHERE`)
- **Order**: Sorting (`ORDER BY ASC/DESC`)
- **Manipulate**: Data transformations (`JOIN`, `UNION`, `MERGE`)
- **Retrieve**: Direct lookups of values (e.g., entity or attribute selection)

Clause Combinations and Their Impact. The system performs perfectly on single-primitive queries such as *Retrieve* (*"Show all EPL goalkeepers"*, 100%) and *Order* (*"Rank Premier League clubs by points"*, 97.6%). It also handles *Calculate + Compare* well (*"Did Haaland score more goals than Salah last season?"*, 96.3%). However, performance drops sharply with added complexity: *Retrieve + Filter + Calculate* (*"What's the average pass accuracy for midfielders under 23?"*) yields 69.3%, and *Calculate + Order* reaches 50.0%. Other challenging cases involve *Manipulate* operations (e.g., table joins) scoring 75%, and four-way compositions (e.g., *Compare + Manipulate + Order + Calculate*) showing similar results.

Impact of Query Complexity. We examine how performance varies with the number of reasoning primitives in a query. Figure 4 plots accuracy against the number of primitives (k), with 95% Wilson confidence intervals.

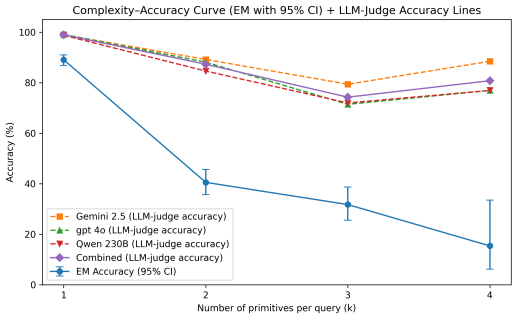


Figure 4: Accuracy Trend over Number of Clauses with LLM as Judge scores

LLM-as-a-judge accuracy is high for retrieval-based queries (99%) but drops to 91.2% with two primitives. This further drops for 3 primitives and slightly improves for queries having 4 or more clauses, possibly due to the scarcity of such complex queries generated from user questions. This trend highlights the importance of robust evaluation metrics as we see divergent trends with EM and LLM as a judge, especially for complex queries showcasing capabilities of frontier models like gemini/gpt to interpret NL questions into executable programs. It also suggests that future benchmarks should include more 3 and 4-primitive questions to better probe system limitations.

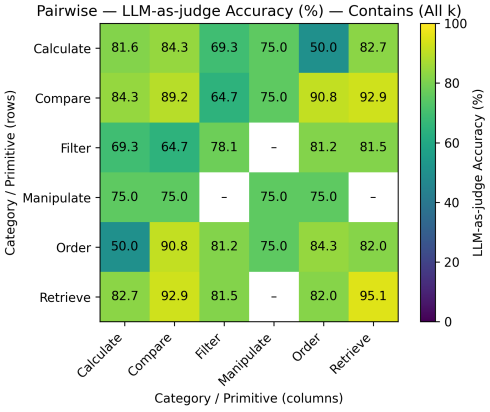


Figure 5: Pairwise Primitive Accuracy with LLM as Judge

Bottleneck Clause Pairs. Figure 5 shows accuracy for all pairs of reasoning primitives. *Retrieve +*

Compare performs well (≥92%), indicating strong compatibility between basic operations. In contrast, any pair involving *Manipulate* or *Calculate* drops sharply, even when combined with otherwise reliable primitives like *Filter*. These patterns align with the decline in Figure 4, where queries involving aggregation or table restructuring introduce significant error.

5 Related Works

Text-to-SQL. Text-to-SQL research has primarily framed the task as cross-domain semantic parsing over static relational schemas. Benchmarks like Spider (Yu et al., 2018b) and its extensions (Li et al., 2023; Zhang et al., 2024; Pourreza and Rafiei, 2023) focus on generalization to unseen databases, yet operate over fixed snapshots with limited domain dynamics. SyntaxSQLNet (Yu et al., 2018a) introduced syntax-tree decoders for nested queries, while recent advances (Zhang et al., 2023; Xie et al., 2024) improve compositional reasoning and execution accuracy.

However, these methods assume immutable schemas, overlook temporal drift in cell values, and sidestep challenges like domain-specific entity resolution (e.g., player aliases) that arise in continuously evolving datasets.

Sports QA. Prior work in sports question answering has largely centered on unstructured text or multiple-choice formats. LiveQA (Liu et al., 2020) explores NBA commentary, using timeline-based MCQs grounded in broadcast text. AskSport (Stoisser et al., 2025) retrieves top-k passages via BM25+RoBERTa, but lacks symbolic execution and numerical guarantees. These systems do not support natural language aggregation (e.g., "average points in last 5 matches") or multi-table joins—capabilities native to SQL.

Our work bridges Text-to-SQL and SportsQA by introducing SPORTSQL, a pipeline tailored to dynamic sports data, and DSQABENCH, the first benchmark pairing natural language queries with executable SQL over temporally indexed, continuously refreshed soccer statistics.

6 Conclusion and Future Work

SPORTSQL demonstrates how natural language interfaces can make complex, evolving sports data accessible to everyday users without technical expertise. The release of DSQABENCH provides a valuable resource for benchmarking and advancing research in dynamic, temporally grounded question answering.

In future work, we plan to (1) support more advanced query types, including comparative and multi-turn analyses across players, teams, and seasons (see Appendix (8)) and (2) generalize the framework to additional structured domains such as finance, healthcare, and other sports like basketball or American soccer. We also plan to perform comprehensive user studies (3) that showcase the effectiveness and applicability of this system for real-time analytics. This work lays the groundwork for scalable, domain-agnostic natural language access to complex, real-world databases.

7 Limitations

While our system performs well on natural language to SQL translation over dynamic sports data, several limitations remain. First, ranked queries using `LIMIT` (e.g., "top 5 goal scorers") may omit tied results due to default lexicographic ordering, yielding incomplete answers. Second, the system supports only English input, limiting accessibility for multilingual users. Third, context length constraints restrict the ability to encode real-time metadata such as recent transfers or lineup changes.

Moreover, the current system is tailored to the English Premier League and does not readily generalize to other sports or leagues without domain-specific adaptation. Expanding to new domains would require schema remapping and possible model fine-tuning. Future work may incorporate multilingual LLMs, retrieval-augmented generation, and adaptive components to improve robustness across languages, domains, and evolving contexts.

References

Nikhil Chandak, Shashwat Goel, Ameya Prabhu, Moritz Hardt, and Jonas Geiping. 2025. Answer matching outperforms multiple choice for language model evaluation. *arXiv preprint arXiv:2507.02856*.

Atharv Kulkarni, Kushagra Dixit, Vivek Srikumar, Dan Roth, and Vivek Gupta. 2025. LLM-symbolic integration for robust temporal tabular reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19914–19940, Vienna, Austria. Association for Computational Linguistics.

Atharv Kulkarni and Vivek Srikumar. 2025. Reinforcing code generation: Improving text-to-sql with execution-based learning. *arXiv preprint arXiv:2506.06093*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357.

Qianying Liu, Sicong Jiang, Yizhong Wang, and Sujian Li. 2020. LiveQA: A question answering dataset over sports live. In *Proceedings of the 19th Chinese National Conference on Computational Linguistics*, pages 1057–1067, Haikou, China. Chinese Information Processing Society of China.

Mohammadreza Pourreza and Davood Rafiei. 2023. Evaluating cross-domain text-to-SQL models and benchmarks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1601–1611, Singapore. Association for Computational Linguistics.

Pritika Ramu, Aparna Garimella, and Sambaran Bandyopadhyay. 2024. Is this a bad table? a closer look at the evaluation of table generation from text. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22206–22216, Miami, Florida, USA. Association for Computational Linguistics.

Josefa Lia Stoisser, Marc Boubnovski Martell, and Julien Fauqueur. 2025. Sparks of tabular reasoning via text2sql reinforcement learning. *arXiv preprint arXiv:2505.00016*.

Yuanzhen Xie, Xinzhou Jin, Tao Xie, Matrixmxlin Matrixmxlin, Liang Chen, Chenyun Yu, Cheng Lei, Chengxiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for enhancing attention: Improving LLM-based text-to-SQL through workflow paradigm. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10796–10816, Bangkok, Thailand. Association for Computational Linguistics.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018a. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663, Brussels, Belgium. Association for Computational Linguistics.

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. 2018b. 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*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024. Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. *arXiv preprint arXiv:2403.02951*.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. ACT-SQL: In-context learning for text-to-SQL with automatically-generated chain-of-thought. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3501–3532, Singapore. Association for Computational Linguistics.

8 Appendix

8.1 Qualitative Examples

Query 1: *Show me the top 10 goal scorers and their goal count.*

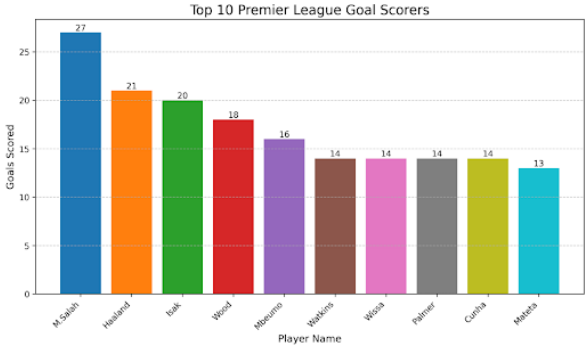
Generated SQL 1:

```
SELECT web_name, goals_scored
    FROM players
    ORDER BY goals_scored DESC
    LIMIT 10;
```

Generated Table 1:

web_name	goals_scored
M.Salah	27
Haaland	21
Isak	20
Wood	18
Mbeumo	16
Watkins	14
Wissa	14
Palmer	14
Cunha	14
Mateta	13

Generated Visual 1:



Query 2: *Give me the player history table for James Milner.*

Generated SQL 2:

```
SELECT * FROM player_history;
```

Generated Table 2:

season name	total points	minutes	goals scored	assists	clean sheet
2006/07	114	2675	3	7	0
2007/08	84	2227	2	2	0
2008/09	128	3060	3	9	0
2009/10	184	3172	7	12	0
2010/11	97	2134	1	7	11
2011/12	86	1586	3	5	6
2012/13	96	1724	4	4	11
2013/14	67	1373	1	6	5
2014/15	107	1749	5	8	7
2015/16	123	2409	5	11	8
2016/17	139	3154	7	4	12
2017/18	77	1759	0	3	6
2018/19	101	1778	5	5	9
2019/20	49	924	2	2	4
2020/21	44	1056	0	2	4
2021/22	38	844	0	1	4
2022/23	42	889	0	1	3
2023/24	28	770	0	2	4

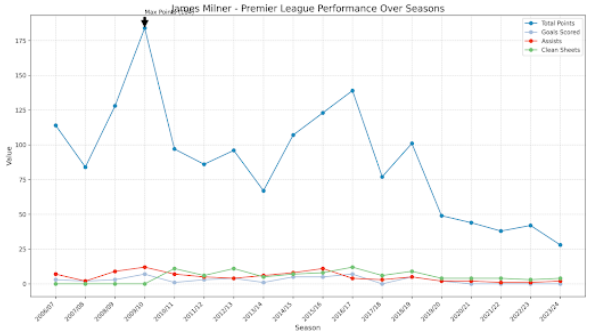


Figure 6: Visualization Output Query 1.

Query 3: *Show me the team names, positions, points, and strength in a color scatterplot.*

Generated SQL 3:

```
SELECT team_name, position, points, strength
FROM teams;
```

Generated Table 3:

team_name	position	points	strength
Liverpool	1	76	5
Arsenal	2	63	4
Nott'm Forest	3	57	4
Newcastle	4	56	4
Man City	5	55	4
Chelsea	6	54	4
Aston Villa	7	54	3
Bournemouth	8	48	4
Fulham	9	48	3
Brighton	10	48	3
Brentford	11	43	3
Crystal Palace	12	43	3
Everton	13	38	3
Man Utd	14	38	3
Spurs	15	37	3
Wolves	16	35	3
West Ham	17	35	3
Ipswich	18	21	3
Leicester	19	18	3
Southampton	20	10	2

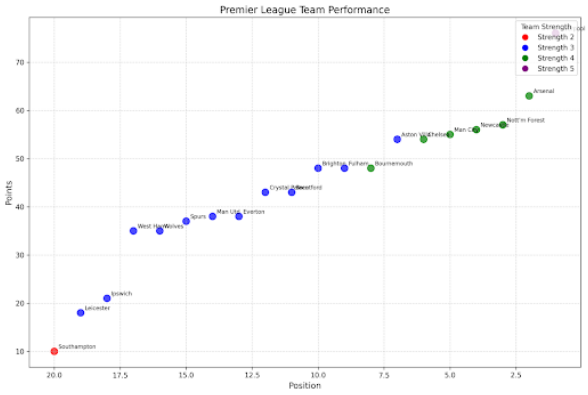


Figure 7: Visualization Output Query 3.

8.2 Deep-Analysis Mode: Multi-Step Reasoning Architecture

To address the limitations of single-shot natural language to SQL translation for complex analytical queries, we introduce a deep-analysis mode that implements a hierarchical decomposition strategy. This mode is specifically designed to handle subjective user questions requiring comprehensive insights that span multiple data dimensions, temporal ranges, and comparative analyses.

- **Stage 1: Entity Extraction and Resolution** The system first extracts entities using the same workflow as described in section 2.1.
- **Stage 2: Hierarchical Query Decomposition** After entity extraction, the system uses a planning module to break the query into focused sub-questions. The system analyzes the query's intent (player insight, team comparison, multi-season analysis) and generates 3-10 prioritized sub-questions that address the user's needs. Each sub-question targets a specific data retrieval requirement, with table hints and priority rankings to guide execution.
- **Stage 3: Parallel SQL Compilation and Execution** Each sub-question is compiled into SQL using a prompt-based translation mechanism, with added context for overall intent. Queries are executed in parallel where possible, and the results are aggregated into a comprehensive response, preserving the semantic relationships across dimensions.

We further plan to extend this feature by adding an insight generation module, extending the utility of this work to support real-time sports analytics at scale. To use this feature, toggle to deep analytics on the platform.