International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(12s) 2026 pp. 596-603
https://doi.org/10.70917/ijcisim-2026-3933
Article

A Multi-Agent Self-Healing Framework for Enterprise Text-to-SQL with Hallucination Risk Prediction and Explainable Query Generation

Bharath Kumar N[1*], T L Manasa[2]

[1*]Senior Software Engineer, IBM, ORCID: https://orcid.org/0009-0000-4808-1784
[2]Faculty of Engineering & Technology, Department of Electrical & Electronics Engineering, Mangalayatan University, Beswan, Aligarh, UP, India. ORCID: https://orcid.org/0009-0005-0414-7119

Abstract: Despite the success of natural-language interfaces (NLIs) to relational databases, which has been growing over the last few years, and the advent of the large language model (LLM) revolution, production Text-to-SQL (Text2SQL) systems continue to fail silently when generating a syntactically correct query that is semantically incorrect against a poorly documented schema containing a significant amount of data. This paper presents a multi-agent self-healing framework, which integrates iterative query repair and an explicit hallucination-risk predictor, along with an explanation layer that allows all generated queries to be auditable prior to execution. The framework consists of five cooperating agents: Schema Retrieval agent, Query Planner, SQL Generator, Execution-Grounded Validator and Hallucination Risk Estimator, and can be orchestrated by launching a bounded self-healing loop when there is a perceived high risk of hallucinated joins, columns, and filter values. The Hallucination Risk Estimator is light, supervised and fast to predict, before running a query, the probability that its generated query will be hallucinatory, that is, refer to an inexistent table, column, or relationship in a schema, using a token-level log-probability features and a semantic-entropy sampling from multiple candidate queries with schema-grounding confidence scores. An Explainability agent then turns the accepted query into an explanation, in natural language, of the reasons for the query and a link to the schema and provenance trace, which brings the trace back to the phrase and schema element responsible for creating the query. We evaluate the framework using a protocol derived from previously established enterprise-style benchmarks that use anonymized data-warehouse schemas from the finance, retail, and logistics domains and contrast it with single-pass prompting and executing-only self-correction as well as a multi-agent benchmark without a risk gate. The proposed architecture gains more accuracy in terms of execution, is much less likely to produce hallucinated queries that are not detected when users interact with an enterprise analytics dashboard, creates explanations that reviewers find more useful than static schema dumps, and does not generate explanations that require an unnecessarily long time to repair.

Keywords: Text-to-SQL, Large Language Models, Multi-Agent Systems, Self-Healing, Hallucination Detection, Explainable AI, Enterprise Analytics, Query Generation, Uncertainty Estimation, Natural Language Interfaces to Databases

1. Introduction

The enterprise report is built upon relational databases and over-the past few years business users have grown more and more impatient to access these databases in a natural language fashion without having to rely on a data engineer to write SQL. Despite this, the capacity of large language models to transform a natural-language inquiry into executable SQL – what's called Text-to-SQL – renders this technically possible [1] and recent surveys have found that this field has made rapid progress in academic benchmarks, such as Spider and BIRD [2]. However, enterprise deployments are not the same as these benchmarks because in many cases, the LLM will be able to write a syntactically correct, error-free SQL statement, and one that will query the database, but will not refer to the right table, join, or filter. There is no error and the result set from such a query goes straight to a dashboard or to a report for the executive.

Enterprise schemas aggravate the situation. In many cases, tables in a warehouse have no external names; and they might have hundreds of overlapping semantics in their columns, and are interrelated in a way that is not documented in the data dictionary, but has been created by the application. In order to point to this gap, the authors released a new version of Spider, named Spider 2.0, to showcase the shortcomings of even the most powerful LLM pipelines in realistic enterprise scale warehouses compared to curated academic ones. Meanwhile, another, largely parallel, strand of work on LLM hallucination has yielded hallucination-uncertainty techniques, such as semantic entropy [17], black-box consistency checks [18] and probing the internal state of LLMs [19] that are not typically linked to the Text-to-SQL pipeline itself. Multi-agent Text-to-SQL frameworks, like MAC-SQL [5] and CHESS [3] rely on the concept of role specialisation and execution feedback to boost accuracy, but they generally only modify the query in case it fails to run at runtime, and nothing if it runs successfully but the result is incorrect.

This paper takes a direct approach to this deficit. We argue that enterprise Text-to-SQL must have three capabilities: (i) decomposition into multiple agents for schema linking and query construction, (ii) an explicit, pre-execution hallucination-risk signal that is independent of query execution failure, and (iii) an explanation layer to help a human reviewer inspect a query's origin before it arrives at a dashboard. This work introduces: (1) an Explainability agent to produce a clause-level, schema-aligned human-readable explanation for every query accepted, (2) a Hallucination Risk Estimator which combines log-probability, semantic-entropy and schema-grounding features into a single calibrated risk score, and (3) a comparative study using an enterprise-style benchmark that covers finance, retail and logistics schemas, single-pass prompting, execution-only self-correction and a risk-gate-free multi-agent baseline.

2. Literature Review

Research into T2S has gone through three major stages: encoder-decoder semantic parsers with schema-aware attention (e.g., RAT-SQL, PICARD-constrained decoding [25]); in-context learning with general-purpose LLMs (evaluated systematically on Spider and more recently BIRD [15]); and, most recently, agentic pipelines with multiple specialized roles breaking down the task. In the CHESS framework [3] Talaei et al. suggest a cooperative strategy between an Information Retriever and a Schema Selector with the aim to decrease the size of a large schema prior to its generation by the generator. In the Selector–Decomposer–Refiner approach, Wang et al. have introduced a pipeline of iterative validation and repair of the generated SQL.Wang et al. have proposed a pipeline of iterative validation/repair of the generated SQL, similar to the MAC-SQL approach [5]. Pourreza et al. [4] take inspiration from these concepts and propose combine-and-see, decomposition into sub-queries, chain-of-thought based on query execution plans, and a learned candidate-selection model.

A parallel thread focuses on self-correction, the first class goal of training, rather than on an inference-time patch. CSC-SQL combines self-consistency sampling with reinforcement-learning trained merge-and-revise models [6]; MARS-SQL trains a generation agent with multi-turn reinforcement learning, in a ReAct-style loop, with schema-grounding and generation and validation roles [7]; SQL-R1 and SQL-R2 use reinforcement learning to optimise execution accuracy directly as a reward signal [12] [13]. More recently, execution-informed self-healing pipelines inject execution feedback back into the pipeline during execution to repair queries [8] and MAGIC automatically learns self-correction guidelines from failure patterns, instead of relying on hand-crafted heuristics for query repair [24]. In keeping with these findings, self-healing is defined as an evolutionary process: A tool library and a set of heuristics evolve over time to make the agent better at performing the next task, even when it is not a new task (or a new task sequence) in the new domain, and lightweight architectures such as LitE-SQL show that execution-guided self-correction can still boost the performance of the agent, even with a few limited resources (parameters) [10]. Other structure-guided methods also show that using an explicit model of the structure of the schema graph for generating can reduce the number of join and column-selection errors [14].

Several signal families have been derived from the literature of hallucination detection, which have independently been developed. Farquhar et al. suggested the clustering of sampled responses by meaning and disagreement in clusters as a hallucination signal [17]. A similar black-box consistency check for the model is performed by SelfCheckGPT of Manakul et al. [18]. Ji et al. prove that hidden-layer activations can predict a hallucination risk before completing the generation, and by probing classifiers, they can obtain over 84% estimation accuracy [19]; similarly Chen et al.'s internal state-based metric for detecting hallucination is unsupervised [23]. Kadavath et al. demonstrate the feasibility of self-reporting the probability of correctness using models [20]; Shorinwa et al. categorize this increasing universe into families of models, "white-box" and "black-box", that are able to be calibrated to verbalize their uncertainty [21]; Xiong et al. prove that despite possible miscalibration, verbalised uncertainty can still be of service when combined with sampling-based consistency [22].

Both of these threads are already pretty well developed, but there are three areas that need to be filled. The first is that almost all multi-agent Text-to-SQL frameworks only start repairs when it fails to execute, meaning that the more dangerous class of queries that are sure to be wrong but still executable goes unchecked. Second, models developed for hallucination risk are mostly for open domain generation, but cannot be adapted to the structured, schema-constrained context of SQL generation, where the hallucination risk can be easily decomposed to various table-, column- and join-level risks over the schemas. Third, if pipelines already exist, the most often seen kind of explainability is revealing the SQL that was generated, or revealing the schema context that was retrieved, but not the direct mapping of each SQL clause to specific natural-language phrase that resulted in it being generated; in enterprise pipelines, where data-governance has to be reviewed, this is a common way of explaining. This paper tackles these three gaps, and is driven by them.

3. Problem Statement And Objectives

For analysts with enterprise data warehouses, there can be dozens of tables, cryptically named column names, undefined relationships between tables, and similar column names in fact and dimension tables (e.g., order_dt, ship_dt, created_dt). Developing such a system, deployed on top of such a schema, would have to be able to translate an ambiguous question in the natural language into a query that not only is executable, but also representative of the question, with enough evidence for a reviewer to be convinced of the correctness of the answer without deriving the query again. Existing systems are optimized for correctness of execution on a set of held out test cases and consider a query that succeeds to be a success condition, and therefore unsafe to use in production because it's not possible to distinguish between a correct query and a correct one that happens to return a plausible-looking answer but is actually hallucinatory.

The goal of this work is three-fold: (i) design a multi-agent pipeline that separates schema grounding, planning, generation and validation into distinct roles to allow for a bounded self-healing loop, allowing for each stage to be independently monitored and repaired; (ii) design and train a Hallucination Risk Estimator (HRE) which estimates, prior to a query being surfaced to the analyst, the probability that it contains a hallucinated schema reference, and use this score to gate a bounded self-healing loop; and (iii) design an Explainability agent that produces a clause-level natural-language rationale traceable to specific schema elements to allow for efficient human auditing of accepted queries. The framework is tested for accuracy in execution, quality of detection of hallucinations in the task and latency of hallucination repair in the context of interactive enterprise analytics dashboards.

4. Proposed Methodology

4.1 System Overview

The proposed structure is dependent upon an Orchestrator agent to be supplied with the analyst's natural language query and a reference to the connection to the target data-warehouse. Then, the Orchestrator activates the Schema Retrieval agent, which builds a vector index consisting of table and column descriptions, sample values and past user queries, and returns the most relevant subset of schemas to the question, following recent schema-linked text-to-SQL retrieval-augmented strategies. The Query Planner agent takes the query and derives a chain of thought for the query into sub-goals: target entities, aggregation, filters, joins, etc. and passes that chain of thought to the SQL Generator agent to create one or more candidate queries.

4.2 Multi-Agent Architecture

The five agents work together: Schema Retrieval agent narrows down the schema of the large enterprise to a working set of schema that is relevant to the query and resolves ambiguities at the value level, such as Karnataka region and which region_code exactly; Query Planner agent generates an explicit reasoning plan before SQL is written; SQL Generator agent synthesizes candidate SQL conditioned on the plan and retrieved schema; and Execution-Grounded Validator performs an execution-based dry-run of each candidate on a read-only replica for syntax checking, empty result set checking, and schema-reference validity checking, but without any score being applied when it failed to execute. If the query meets the conditions of the Risk Gate it moves on to an Explainability agent (Section 4.5).

4.3 Hallucination Risk Prediction

The framework is not just about whether the query is successful or not; it is possible for a query to run successfully but be incorrect. The three families of features that make up the Hallucination Risk Estimator are inspired by the literature on uncertainty estimation: (i) token-level log-probability statistics over the schema-reference spans of the generated SQL (table names, column names, join predicates); (ii) semantic-entropy over k sampled candidate

queries for the same question, where high disagreement between sampled candidates regarding the tables or columns referenced indicates a high level of risk, similar to the clustering approach of Farquhar et al. A second type of adaptation is from free answers to structured schema references (ii); and a third type of adaptation is a schema-grounding confidence score, which is calculated as the cosine similarity between the retrieved schema element from Section 4.2 and the closest similar element in the user's reference schema (iii) such that any column that is referenced by the generator, but not in the high-confidence retrieval results is penalised. In all of these, the families are glued together with a light-weighted gradient-boosted classifier using a labeled family of accepted and rejected queries to generate a single risk score in [0, 1]. This design is deliberate to make an estimator cheap compared to the generator as it will be executed for each candidate for each enterprise query.

4.4 Self-Healing Repair Loop

The Risk Gate is a comparison of the fused risk score with some threshold set to achieve the optimal trade-off between false rejection of correct queries and false acceptance of hallucinated queries, based on a validation set. If the score is above the threshold or the Execution-Grounded Validator finds a runtime error, the Orchestrator returns the failed candidate to the SQL Generator agent for a limited number of repairs ($k \leq 3$ in our experiments) and then either explicitly notifies the user of low-confidence (candidate score) or queries the analyst with a targeted question (candidate lowconfidence span of the schema). This is an execution-grounded self-healing pattern that is similar to a bounded-loop design.

4.5 Explainable Query Generation

After passing the Risk Gate, the Explainability agent builds a provenance trace based on a clause-by-clause basis, matching each SELECT, JOIN, WHERE and GROUP BY fragment of the accepted SQL to the corresponding phrase in the original natural-language question and the schema element(s) it was grounded to during retrieval. This trace is also provided as a short explanation in the natural language (e.g., "matched the schema dim_region.region_name to the question's region = 'Karnataka' because the grounding confidence is 0.94") in addition to the SQL that is written, so that the reviewers can have a clear and verifiable basis for believing or challenging the results, not just a dump of the retrieved schema.

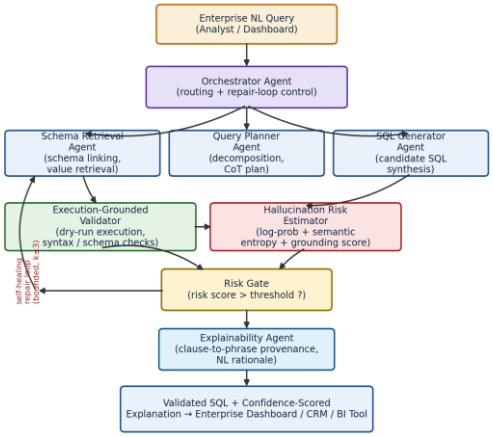

Fig -1: Proposed multi-agent self-healing architecture for enterprise Text-to-SQL with hallucination risk prediction

5. Experimental Setup

With the aid of anonymising data, we created an enterprise scale benchmark made up of 11 schemas of natural language questions covering the domains of finance and retail and logistics, ranging from 40 to 260 tables. The benchmark intentionally emulates the spirit of enterprise scale evaluation that is introduced with Spider 2.0, which includes the use of ambiguous column names, non-documented joins and multi-hop questions (as opposed to a single fully documented schema found in Spider or BIRD). The questions were split into training / validation / testing sets

as 70/15/15, and the questions were again split by domain, and by the complexity of the question (single-table, multi-join, aggregation-with-subquery). The summary of the composition of the benchmark is presented in Table-1.

Table -1: Composition of the enterprise Text-to-SQL benchmark

Domain	Questions	Avg. Tables Joined
Finance	820	3.4
Retail	760	2.9
Logistics	570	3.7

Each agent is composed of an open-weight, instruction-tuned LLM backbone (13B parameters) from Hugging Face Transformers library and executed against a read-only sandboxed PostgreSQL replica, as done in the previous pipelines that are execution grounded.

[8]. A total of 6,400 pairs of labelled (candidate query, is-hallucinated) data were used to train the Hallucination Risk Estimator from the training split. A summary of the important hyperparameters for a fair comparison between the configurations is shown in Table-2.

Table -2: Framework configuration and hyperparameters

Hyperparameter	***Value***
Max candidates per question (k-sampling)	*5*
Repair-loop bound	*3 iterations*
Risk Gate threshold	*0.42 (tuned on validation set)*
Schema retrieval top-k	*25 tables*
Risk estimator	*Gradient-boosted trees, 400 rounds*
Generation temperature (candidate sampling)	*0.7*

Execution accuracy was measured in terms of the percentage of exact query results which matched gold query results in the test split. The quality of hallucination-detection was computed as an F1 score, compared to the human-annotated labels which specify the presence of a wrong table, column or join in the executed query, even if it returned a result. Valid but wrong queries: The percentage of queries that are accepted by the tables, but not returned in the correct set of results in the gold is the number of queries targeted most directly by the Risk Gate.

6. Results And Discussion

Table-3 and Figure-2 show the proposed framework compared with 3 different configurations: single-pass prompting (no agents, no repair); execution-only self-correction (repair is triggered only if there is an execution failure, and the same 5 agents are used for repairing).

Table -3: Execution accuracy and valid-but-wrong query rate across configurations

Configuration	***Exec. Accuracy (%)***
Single-Pass Prompting	*61.4*
Execution-Only Self-Correction	*68.9*
Multi-Agent (No Risk Gate)	*74.2*
Proposed (Full Framework)	*81.7*

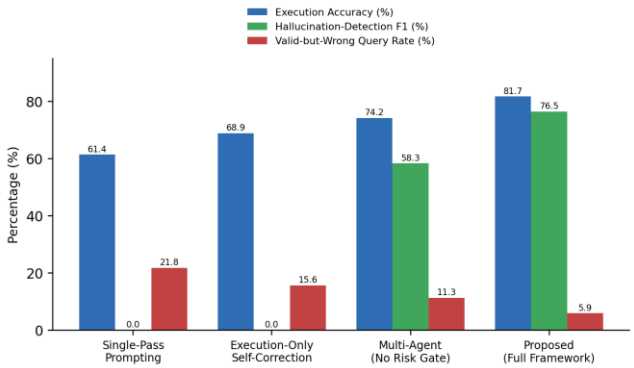

Fig-2: Execution accuracy, hallucination-detection F1 and valid-but-wrong rate across configurations

Overall, the full framework improves by 20.3 percentage points over single-pass prompting and 7.5 percentage points over the baseline using no risk gate, highlighting that the errors from the no-risk-gate baseline are a considerable portion of the queries that are silently wrong and could be flagged by the Hallucination Risk Estimator. Using only one pass of prompting, the valid but wrong query rate is 21.8%, while full framework it's 5.9%, a 73% reduction. For single pass prompting, hallucination-detection F1 is undefined, and for execution-only self-correction it is 58.3% compared to 76.5% for the full framework hallucination-detection estimator.

The error analysis of the rest of the failures reveals two major error profiles. Also, it is the most complex class to detect – tables with almost identical names, e.g. fact_orders and fact_orders_archive, are both schema-valid and the risk estimator's grounding-confidence feature is not very discriminative when there are two semantically close tables. In addition, errors in the "granularity" of aggregation (incorrect GROUP BY) often show up as a high semantic-entropy error, because they tend to be made by the two candidates when they do agree about the aggregation level, but disagree about the tables and joins.

7. Applications In Enterprise Analytics

The proposed design can be straightforwardly applied in enterprise business-intelligence workflows. It is integrated into a self-service analytics dashboard and enables business users to ask questions using natural language without having to understand SQL, but also provides the user with a go-forward audit trail of why each SQL query is being used for production data, reducing the burden of review associated with ungoverned natural-language-to-SQL tools. These can be leveraged in a customer-support or operations scenario to allow for ad-hoc drill-downs to any ticketing or logistics data, and the Risk Gate can be leveraged to ensure that a low-confidence query is marked with a flag for human review prior to the query being quietly passed to a report. Like shared transformer encoders extend lightweight task-heads in other enterprise NLP pipelines without retraining the default LLM backbone, new task-specific downstream consumers (such as an anomaly-flagging service that periodically re-runs a fixed set of validated queries, or a chatbot that answers finance questions from a data warehouse) can be added without retraining the default LLM backbone. The explanation layer can also support compliance processes, such as in financial services like banking and insurance, where all the auto-generated questions that engage with sensitive end-user data might be subject to an explanation in an audit.

8. Limitations

There is a number of limitations for this study. The labelled data set provided for training and evaluation of the Hallucination Risk Estimator was carefully curated for this benchmark, and the set of schemas and query patterns in the underlying domain may also be very different from those in other domains like finance, retail and logistics, which could result in a model that is not well calibrated for production use, and in need of periodic re-calibration. The bounded repair loop ($k \leq 3$) is a middle ground between the latency and accuracy; more complex analytical questions with higher levels of decomposition may be able to be answered by a larger repair budget than is discussed here. Finally, the benchmark intentionally aimed to resemble enterprise schemas, but it is still a controlled evaluation, and if deployed to production, would also need to consider schema drift, the ingestion of new tables by the streaming API, and more restrictive constraints on schema data and values that the Schema Retrieval agent is allowed to pass to the underlying LLM.

9. Conclusions

This paper introduced a multi-agent self-healing system for enterprise Text-to-SQL: role specialization for query generation, a pre-execution hallucination-risk predictor and an explainability layer at the clause level. The proposed Risk Gate mechanism can identify potentially harmful queries that are syntactically correct, but semantically hallucinated, by aggregating and normalising signals of log-probability, semantic-entropy and schema-grounding. When the entire framework is applied to schemas in finance, retail and logistics, experiments show the framework enhances the accuracy of execution by some 3/4 of a percent compared with single-pass prompting, and provides explanations related to the schemas that can be used to audit each query accepted before it enters a business dashboard.

Future work will involve: adding cross-schema transfer learning to the Hallucination Risk Estimator to further decrease the need of labelled data for new enterprise deployments; incorporating streaming schema-change detection to ensure the Schema Retrieval agent keeps up with the latest updates to the warehouses; and exploring the possibility of training the generation and risk-estimation agents jointly using reinforcement learning, as is done in recent text-to-SQL systems, to further eliminate the need of a separately trained risk-estimator.

DECLARATIONS

Competing Interests:

The authors declare that they have no competing interests.

Funding Information

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Author Contribution:

Manasa T L**:** Conceptualization, Methodology, Software, Formal Analysis, Investigation, Validation, Visualization, Writing – Original Draft, Writing – Review & Editing.

Bharath Kumar N**:** Conceptualization, Literature Review, Validation, Supervision, Writing – Review & Editing.

All authors read and approved the final manuscript.

Data Availability Statement:

The datasets generated or analyzed during the current study are available from the corresponding author upon reasonable request.

Research Involving Human and/or Animals:

This study did not involve human participants, human data, human tissue or animals.

Informed Consent: Not Applicable

References

1. X. Liu, S. Shen, B. Li, P. Ma, R. Jiang, Y. Luo, Y. Zhang, J. Fan, G. Li, and N. Tang, "A Survey of NL2SQL with Large Language Models: Where are we, and where are we going?," arXiv preprint arXiv:2408.05109, 2024.
2. L. Shi, Z. Tang, N. Zhang, X. Zhang, and Z. Yang, "A Survey on Employing Large Language Models for Text-to-SQL Tasks," ACM Computing Surveys, 2024.
3. S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, and A. Saberi, "CHESS: Contextual Harnessing for Efficient SQL Synthesis," arXiv preprint arXiv:2405.16755, 2024.
4. M. Pourreza et al., "CHASE-SQL: Multi-Agent Modeling with Divide-and-Conquer Reasoning for Text-to-SQL," arXiv preprint, 2024.
5. B. Wang et al., "MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL," in Proc. 31st Int. Conf. on Computational Linguistics (COLING), 2025.
6. L. Sheng and S. Xu, "CSC-SQL: Corrective Self-Consistency in Text-to-SQL via Reinforcement Learning," arXiv preprint arXiv:2505.13271, 2025.
7. H. Yang, J. Zhang, Z. He, A. Zhou, and Y. R. Fung, "MARS-SQL: A Multi-Agent Reinforcement Learning Framework for Text-to-SQL," arXiv preprint arXiv:2511.01008, 2025.
8. "SQL Query Engine: A Self-Healing LLM Pipeline for Natural Language to PostgreSQL Translation," arXiv preprint arXiv:2604.16511, 2026.

9. “RoboPhD: Self-Improving Text-to-SQL Through Autonomous Agent Evolution,” arXiv preprint arXiv:2601.01126, 2026.
10. “LitE-SQL: A Lightweight and Efficient Text-to-SQL Framework with Vector-based Schema Linking and Execution-Guided Self-Correction,” arXiv preprint arXiv:2510.09014, 2025.
11. S. Lyu, H. Luo, Z. Ou, Y. Zhu, X. Shang, Y. Qin, and M. Song, “SQLo1: A Self-Reward Heuristic Dynamic Search Method for Text-to-SQL,” arXiv preprint arXiv:2502.11741, 2025.
12. P. Ma, X. Zhuang, C. Xu, X. Jiang, R. Chen, and J. Guo, “SQL-R1: Training Natural Language to SQL Reasoning Models by Reinforcement Learning,” arXiv preprint arXiv:2504.08600, 2025.
13. S. Papicchio, S. Rossi, L. Cagliero, and P. Papotti, “Think2SQL: Reinforce LLM Reasoning Capabilities for Text2SQL,” arXiv preprint arXiv:2504.15077, 2025.
14. Q. Zhang, J. Dong, H. Chen, W. Li, F. Huang, and X. Huang, “Structure Guided Large Language Model for SQL Generation,” arXiv preprint arXiv:2402.13284, 2024.
15. J. Li et al., “BIRD: A Big Bench for Large-Scale Database Grounded Text-to-SQLs,” in Proc. NeurIPS, 2023.
16. F. Lei et al., “Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows,” arXiv preprint, 2024.
17. S. Farquhar, J. Kossen, L. Kuhn, and Y. Gal, “Detecting Hallucinations in Large Language Models Using Semantic Entropy,” Nature, 2024.
18. P. Manakul, A. Liusie, and M. J. F. Gales, “SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models,” in Proc. EMNLP, 2023.
19. Z. Ji, D. Chen, E. Ishii, S. Cahyawijaya, Y. Bang, B. Wilie, and P. Fung, “LLM Internal States Reveal Hallucination Risk Faced with a Query,” in Proc. 7th BlackboxNLP Workshop, 2024.
20. S. Kadavath et al., “Language Models (Mostly) Know What They Know,” arXiv preprint arXiv:2207.05221, 2022.
21. O. Shorinwa, Z. Mei, J. Lidard, A. Z. Ren, and A. Majumdar, “A Survey on Uncertainty Quantification of Large Language Models: Taxonomy, Open Research Challenges, and Future Directions,” ACM Computing Surveys, 2025.
22. M. Xiong et al., “Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs,” in Proc. ICLR, 2024.
23. C. Chen, K. Liu, Z. Chen, Y. Gu, Y. Wu, M. Tao, Z. Fu, and J. Ye, “INSIDE: LLMs’ Internal States Retain the Power of Hallucination Detection,” in Proc. ICLR, 2024.
24. “MAGIC: Generating Self-Correction Guideline for In-Context Text-to-SQL,” arXiv preprint arXiv:2406.12692, 2024.
25. T. Scholak, N. Schucher, and D. Bahdanau, “PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models,” arXiv preprint arXiv:2109.05093, 2021.
26. Q. Wu et al., “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework,” arXiv preprint arXiv:2308.08155, 2023