Validating LLM-Generated SQL Queries through Metamorphic Prompting

LI LIN, Xiamen University, China
QINGLIN ZHU, Xiamen University, China
JINTAI HONG, Xiamen University, China
CHONG WANG, Nanyang Technological University, Singapore
YANG LIU, Nanyang Technological University, Singapore
RONGXIN WU, Xiamen University, China

Large Language Models (LLMs) can translate natural language (NL) into SQL, enabling non-experts to query databases via conversational interfaces. However, the generated SQL often contains *intent-violating hallucinations*—queries that are syntactically valid and executable, yet semantically misaligned with the user's question. These failures are especially risky in real-world settings where users cannot verify the correctness.

In this paper, we propose MRSQLGen, a framework for detecting intent-violating hallucinations, built on the metamorphic prompting paradigm. MRSQLGen rewrites the input prompt using task-specific transformation rules derived from a hallucination taxonomy, and validates the generated SQL by checking behavioral consistency across multiple executions. Each transformation is associated with a metamorphic relationship (MR) that defines the expected relation between results; discrepancies are aggregated through a majority-vote strategy to robustly flag hallucinations without ground-truth SQL. We evaluate MRSQLGen on two benchmarks (Spider and Bird) using five representative LLMs, including GPT-4o. Experimental results demonstrate that MRSQLGen consistently outperforms state-of-the-art hallucination detection techniques, achieving higher precision and recall in detecting hallucinated SQL queries.

CCS Concepts: • **Software and its engineering → Software maintenance tools**.

Additional Key Words and Phrases: Metamorphic Prompting; NL2SQL; Hallucinations

ACM Reference Format:
Li Lin, Qinglin Zhu, Jintai Hong, Chong Wang, Yang Liu, and Rongxin Wu. 2026. Validating LLM-Generated SQL Queries through Metamorphic Prompting. *Proc. ACM Softw. Eng.* 3, FSE, Article FSE019 (July 2026), 23 pages. https://doi.org/10.1145/3797146

1 Introduction

Emerging trends and challenges. With the rise of large language models (LLMs), translating natural language (NL) into SQL has become a practical solution for non-experts to query databases. NL2SQL methods simplify access to structured data by lowering the barrier to understanding SQL syntax and database concepts, enabling broader adoption in data analysis [1–4], business intelligence [5–8], and other data-centric domains. However, one of the critical challenges they face is the problem of hallucination, where the models generate plausible but factually incorrect answer.

Authors' Contact Information: Li Lin, Xiamen University, , China, linli1210@stu.xmu.edu.cn; Qinglin Zhu, Xiamen University, , China, 23020241154481@stu.xmu.edu.cn; Jintai Hong, Xiamen University, , China, hongjintai@stu.xmu.edu.cn; Chong Wang, Nanyang Technological University, , Singapore, chong.wang@ntu.edu.sg; Yang Liu, Nanyang Technological University, , Singapore, Yangliu@ntu.edu.sg; Rongxin Wu, Xiamen University, , China, wurongxin@xmu.edu.cn.

ACM 2994-970X/2026/7-ARTFSE019
https://doi.org/10.1145/3797146

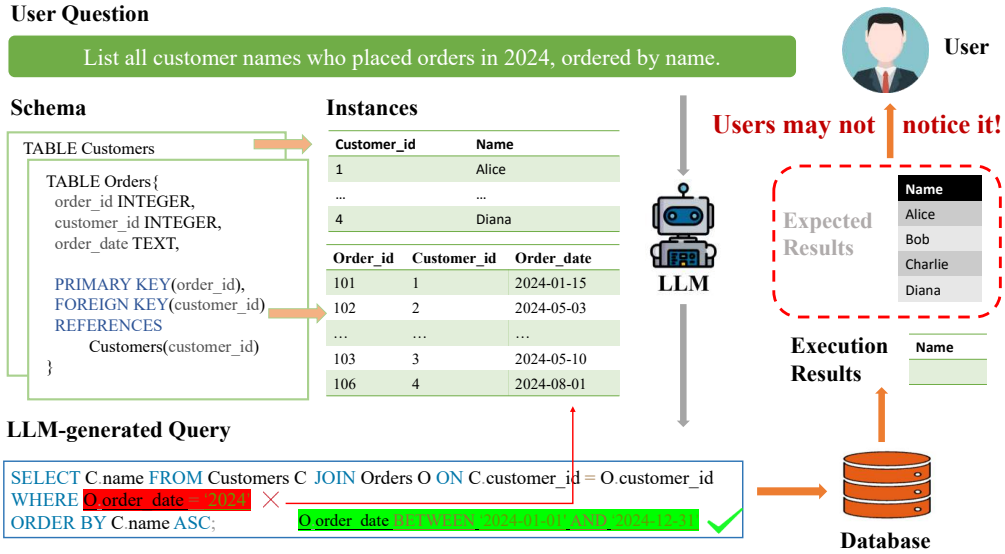

Fig. 1. **A Case of Intent-Violating Hallucination in NL2SQL**

Some user studies [9, 10] show that programmers find it hard to trust and debug the LLM-generated SQL query as the generation procedure is opaque and out of control. Intuitively, since LLMs are trained on numerous code repositories, such as GitHub, they may inadvertently learn from code that contains bugs and flaws. Therefore, without rigorous validation and testing, incorporating these unverified queries into production systems could lead to system failures, data corruption, or other critical malfunctions, ultimately undermining the reliability of the entire application.

Detecting hallucinations in NL2SQL systems faces a critical challenge. While syntax errors and schema violations are relatively easy to detect and resolve, hallucinations that violate a user's query intent are significantly more difficult to identify. We refer to these errors as **intent-violating hallucinations** [1] —SQL queries that are syntactically valid and executable, yet semantically misaligned with the user's original question. As shown in Figure 1, an LLM may generate a syntactically valid SQL query that subtly deviates from the user's intent—for example, filtering records using "`order_date = '2024'`" instead of the correct range-based condition "`BETWEEN '2024-01-01' AND '2024-12-31'`". Although the query executes successfully, it silently returns an empty result or a partial result set, misleading users into thinking that no matching records exist. This is particularly dangerous in real-world data analysis and decision-making scenarios, where users typically lack the technical expertise or schema awareness needed to verify the correctness of the generated query. The deceptive validity of such hallucinations underscores the urgent need for real-time hallucination detection mechanisms in NL2SQL systems. This motivates our work: we aim to systematically identify and mitigate intent-violating hallucinations in real-time SQL generation by ensuring consistency between query behavior and user intent.

Related work and their limitations. Existing techniques on model and code quality validation can hardly detect erroneous LLM-generated SQL queries. ① A common line of work adopts *external-resource-based validation methods* to detect hallucinations in LLM-generated SQL. One

[1]In this paper, we primarily focus on intent-violating hallucination detection, which we sometimes refer to simply as hallucination detection for brevity.

representative category is benchmark-based evaluation. For instance, Spider [11] and similar benchmarks are widely used to assess the SQL generation capabilities of LLMs by comparing model outputs against gold-standard queries. However, such evaluations require access to predefined ground-truth SQL, which are generally unavailable in real-world applications—otherwise, automated generation would not be needed. Another category involves static or dynamic analysis tools [12, 13]. These tools detect surface-level issues such as syntax errors, type mismatches, or invalid schema references using SQL parsers (e.g., ANTLR [12]) or runtime checks. While effective for generic bugs, they fail to assess whether the generated SQL semantically aligns with the user's intent. ② Another line of research leverages *confidence estimation* to detect hallucinations, especially in natural language generation (NLG) tasks such as question answering [14], summarization [15], and dialogue systems [16]. These methods [17, 18] assume that low-confidence outputs—measured via token probabilities or entropy—are more likely to be hallucinated. However, most LLMs are black-box systems that do not expose confidence scores. To address this limitation, SelfCheckGPT [19] proposes a model-agnostic strategy that queries the LLM multiple times to generate alternative responses and compares them to the original output. While promising, SelfCheckGPT is less effective for NL2SQL tasks, as the model tends to generate nearly identical SQL queries across samples [20], making it hard to spot hallucinated segments through output comparison. ③ A third line of work uses *LLM-as-a-judge methods* to evaluate whether generated outputs are consistent with the input prompt, a strategy primarily applied in NLG tasks. While effective for assessing semantic consistency in natural language outputs, these methods struggle in NL2SQL tasks, where correctness depends on precise schema usage and execution logic rather than shallow semantic alignment, making intent-violating hallucinations difficult to detect [21].

Key insights. To address the limitations of prior work, we propose a novel hallucination detection technique called **metamorphic prompting** (MP)—a resource-agnostic approach that requires neither predefined ground-truth SQL nor access to external validators or model confidence scores. Our method draws inspiration from metamorphic testing (MT), a software testing paradigm that validates program behavior without test oracles by applying controlled input transformations that preserve expected input-output relationships, known as metamorphic relationships (MRs); violations of these MRs often signal latent faults. We extend this principle to the LLM-based hallucination detection. Our key insight is that hallucinated SQL queries often reveal behavioral inconsistencies when subjected to carefully crafted prompt transformations. These transformations are not necessarily semantically equivalent but are designed to preserve or slightly alter the original intent in a controlled manner. A correct SQL query should still produce consistent behavior under such changes, whereas hallucinated queries—often based on brittle or shortcut reasoning—tend to fail this consistency check.

Unlike sampling-based methods (e.g. SelfCheckGPT) that passively generate multiple outputs from the same prompt, MP actively rewrites the input question using intent-preserving or intent-perturbing modifications. These rewrites guide the model along alternative reasoning paths, thereby increasing the chances of revealing hidden inconsistencies in its SQL generation behavior. By analyzing discrepancies across the outputs of these transformed prompts, we can detect intent-violating hallucinations without relying on any reference answers.

MRSQLGen. Leveraging these insights, we implement a novel framework, MRSQLGen, which applies MP to validate LLM-generated SQL queries. The workflow consists of two modules:

- **Prompt Paraphrasing.** Given an input natural language prompt, the Prompt Paraphrasing module generates a set of metamorphic variants using a suite of transformation rules. To construct these rules, we first conduct an empirical study to identify common hallucination patterns and their root causes in LLM-generated SQL. Based on this analysis, we construct a hallucination

knowledge base (HKB) and design a set of prompt-level metamorphic rules tailored to these failure modes. To ensure each prompt is paired with the most relevant transformations, we employ a similarity-based retrieval mechanism that identifies the most likely hallucination types from HKB, based on the semantics of the input prompt and the structure of the generated SQL. We then select and adapt appropriate transformation rules accordingly. These rules are used to generate metamorphic prompts, which are fed back into the LLM to produce transformed SQL queries, referred to as metamorphic queries.

- **Cross Validation.** The Cross Validation module executes both the original and metamorphic queries against the same database and compares their outputs to detect behavioral inconsistencies. Inspired by the prior work [22], we adopt a balanced comparison strategy that flags a hallucination only when the majority of metamorphic queries break expected MR, thereby enhancing robustness and reducing false positives.

Evaluation. We conduct a comprehensive evaluation of MRSQLGen on two widely used NL2SQL benchmarks: Spider [11] and Bird [23], both of which provide ground truth SQL queries, allowing us to use the execution accuracy (EX) metric to determine whether an LLM-generated SQL query contains hallucinations. Our evaluation covers five representative LLMs, including GPT-4o, spanning a range of model families and capabilities. We compare MRSQLGen against two strong baselines: SelfCheckGPT, a sampling-based hallucination detector, and an in-house implementation of *LLM-as-a-judge*, which prompts the LLM itself to assess the correctness of its own output. Experimental results show that MRSQLGen consistently outperforms both baselines in terms of precision, recall, and F1 score across all evaluated models. In particular, with CodeLlama, MRSQLGen attains an F1 score of 0.805 on Spider and 0.944 on Bird, exceeding SelfCheckGPT by 121%/181% and LLM-as-a-judge by 225%/272%, respectively. Our ablation study further confirms that both the Prompt Paraphrasing and Cross Validation modules are essential to reliable hallucination detection.

Contributions. Our major contributions include:

- We propose the first hallucination detection technique based on metamorphic prompting to identify intent-violating hallucinations in LLM-generated SQL queries, without requiring ground truth or external validation signals.
- We conduct a comprehensive empirical study of hallucinations in LLM-based NL2SQL generation, leading to a structured taxonomy of hallucination types and an analysis of their root causes.
- We develop and open-source MRSQLGen, a practical framework for hallucination detection in NL2SQL. Extensive experiments on benchmark datasets show that it significantly outperforms strong baselines across multiple LLMs.

2 Background and Motivation

Metamorphic Testing. Metamorphic Testing (MT) is a well-established software testing paradigm designed to address the oracle problem—the challenge of verifying correctness when the expected output is unknown or difficult to specify [24]. Rather than checking correctness against a fixed oracle, MT relies on Metamorphic Relationships (MRs)—expected relationships between inputs and outputs that should hold under specific transformations. Given a source test input and its output, MT generates one or more follow-up test inputs through predefined rules named metamorphic rules and verifies whether the outputs satisfy the corresponding MRs [25, 26].

Due to its ability to handle unknown outputs, MT has been effectively applied to testing LLMs, which often produce non-deterministic results. Some studies [27–29] have explored automated testing frameworks that integrate logic programming with MT to detect fact-conflicting hallucinations in LLMs. Ma et al. [30] proposed a MT-based approach to evaluate NLP models and identify potential biases. Additionally, other studies [31–34] have explored the robustness of LLMs

by testing them with adversarial input text or modified prompt descriptions. In contrast to these works, we are the first to adopt the methodology of MT to validate LLM-generated SQL queries by elevating the MT process to the prompt level.

Metamorphic Prompting Paradigm. Given a natural language prompt x and an LLM-generated SQL query $q = \text{LLM}(x)$, our goal is to determine whether q exhibits an *intent-violating hallucination*. Since no ground truth SQL is assumed, we adopt a metamorphic prompting (MP) approach, which defines a set of semantic-preserving transformations $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ over the input prompt.

For each transformation $T_i \in \mathcal{T}$, a new prompt $x_i = T_i(x)$ is generated, and the corresponding SQL $q_i = \text{LLM}(x_i)$ is obtained. Let r and r_i denote the execution results of q and q_i on the same database D, i.e., $r = \text{Exec}(q, D)$ and $r_i = \text{Exec}(q_i, D)$. Each transformation T_i is associated with a predefined MR $\mathcal{R}_i$ that specifies the expected relation between r and r_i.

We define that q is hallucinated if the number of violated MRs exceeds a threshold ρn, where $\rho \in [0, 1]$ is a configurable proportion parameter:

$$\text{Hallucination}(q) = \left(\sum_{i=1}^{n} \mathbb{I}\left[\neg \mathcal{R}_i(r, r_i)\right] > \rho \cdot n \right),$$

where $\mathbb{I}[\cdot]$ returns 1 if the condition is true, and 0 otherwise. In other words, a hallucination is flagged when the proportion of violated MRs exceeds the threshold ρ.

Empirical Benefits of Metamorphic Prompting. We aim to empirically verify our key hypothesis that MP can alter the reasoning trajectories of LLMs, thereby improving their ability to detect hallucinations. Neuron activation coverage has been widely adopted as a proxy for how thoroughly a model explores its internal representation space [35]. Higher coverage indicates that more neurons or activation regions are engaged during inference, leading to richer and more diverse reasoning trajectories. Building on this, Dong et al. [36] argue that when a task admits multiple valid reasoning paths, these paths are unlikely to produce the same erroneous output. In other words, greater diversity in reasoning paths tends to yield more varied error patterns, making discrepancies between outputs easier to detect—discrepancies that can be leveraged to identify hallucinations.

To empirically test this hypothesis, we design a preliminary experiment to examine whether MP can indeed alter the reasoning trajectories of LLMs. We randomly select 200 prompts from the Spider [11] dataset and compare two prompting strategies: (i) *Sampling-based prompting*, following the SelfCheckGPT [19], where the model is asked with the same original prompt 10 times (using stochastic sampling) to produce 10 outputs; (ii) *Metamorphic prompting*, where we construct 10 metamorphic variants of the original prompt and feed each variant to the model once, yielding 10 outputs. Since neuron-level activations are generally inaccessible in closed-source LLMs, we conduct the experiment on the open-source Mistral-7B-Instruct-v0.2 model [37]. Following prior work [38], we measure **Neuron Coverage (NC)**, **Top-k Neuron Coverage (TKNC)**, and **Neuron Layer Coverage (NLC)**, which collectively quantify the diversity and breadth of neuron activations as a proxy for exploring alternative reasoning paths. As shown in Table 1, MP achieves improvements of 160.36%, 95.83%, and 3.19% in NC, TKNC, and NLC, respectively, compared to sampling-based prompting, indicating substantially broader and more diverse neuron activation. These findings provide empirical evidence that MP effectively diversifies the model's reasoning trajectories, thereby enhancing its ability to detect hallucinations.

Challenges in Metamorphic Prompting. Applying MP to hallucination detection in LLM-based SQL generation introduces two fundamental challenges.

- **Challenge 1: Constructing reliable metamorphic prompts.** In traditional software, MRs are often derived from formal specifications or predefined input-output patterns [39–43]. However, for natural language prompts, there is no canonical form or deterministic specification to guide

Table 1. **Neuron Activation Coverage under Sampling-based vs. Metamorphic Prompting.**

Prompting Method	NC (%)	TKNC (%)	NLC (Neurons)
Sampling-based Prompting	25.91	0.24	790,450
Metamorphic Prompting	67.46(+160.36%)	0.47(+95.83%)	815,636(+3.19%)

Note: NC (**Neuron Coverage**) measures whether each individual neuron is activated by the inputs; TKNC (**Top-k Neuron Coverage**) measures whether the top-k most activated neurons are covered; NLC (**Neuron Layer Coverage**) analyzes the distribution of neuron activations to assess whether new regions of the activation space are explored.

MR design. Since LLMs are sensitive to subtle linguistic variations, crafting prompt transformations that reliably preserve or perturb intent—without introducing ambiguity or drift—is inherently difficult. Effective MP requires transformations that are semantically controlled yet diverse enough to expose behavioral inconsistencies in the model.

- **Challenge 2: Verifying hallucinations under uncertainty.** The probabilistic nature of LLMs introduces ambiguity in result interpretation. Specifically, randomness arises from two sources: (i) the non-deterministic decoding process may yield different SQLs across identical prompts, and (ii) behavioral inconsistency between the original and metamorphic prompts makes it unclear which query is hallucinated. This dual uncertainty complicates hallucination diagnosis and requires a robust verification mechanism that tolerates variation while ensuring semantic consistency.

Solutions. To tackle the first challenge, we observe that hallucinations in NL2SQL generation often follow recurrent patterns rooted in specific reasoning failures (e.g., join logic errors). This motivates a taxonomy-driven approach: we conduct an empirical study to identify common hallucination types, store them in a structured Hallucination Knowledge Base (HKB), and then distill a set of MRs tailored to each failure mode. Given a new prompt, we retrieve the most similar historical cases from the HKB and apply targeted prompt transformation rules to generate semantically controlled variants. These transformations are designed to preserve or minimally perturb the original query intent while surfacing inconsistencies in the model's reasoning. To address the second challenge, we propose the metamorphic prompting paradigm to detect hallucinations through behavioral consistency checks. Specifically, we compare the execution results of the original query and its metamorphic variants to assess semantic stability. Instead of relying on any single discrepancy, we introduce a cross-validation mechanism [22] that flags a hallucination only when a majority of the metamorphic queries violate their expected relationships. This voting mechanism mitigates the impact of LLM randomness and allows robust, reference-free hallucination detection.

Motivating Example. Figure 2 illustrates a motivating example of MRSQLGen. In this case, the LLM generates a SQL query that incorrectly filters by an exact year string `"2024"`, resulting in a *violating value specification* error that silently returns an empty result. To detect such subtle hallucinations, MRSQLGen applies the metamorphic prompting paradigm: it rewrites the original prompt into several semantically controlled variants, each targeting different aspects of model reasoning. Specifically, MP_1 introduces a temporal zoom-in (e.g., "May 2024") to test fine-grained value understanding, MP_2 broadens the temporal scope (e.g., "2023 to 2024") to verify coverage consistency, and MP_3 applies a chain-of-thought (CoT) style to encourage step-by-step reasoning. By executing the resulting SQL queries and comparing their outputs against the original under predefined MRs, MRSQLGen identifies behavioral inconsistencies and flags the original query as hallucinated when a majority of transformations violate expectations.

3 System Overview

The overview of our approach is illustrated in Figure 3. Our goal is to systematically detect and mitigate hallucinations in real-time SQL generation by ensuring consistent alignment between query behavior and user intent. To achieve this, we introduce a dedicated Validating Component, which performs runtime validation of the generated SQL queries. In this component, the input

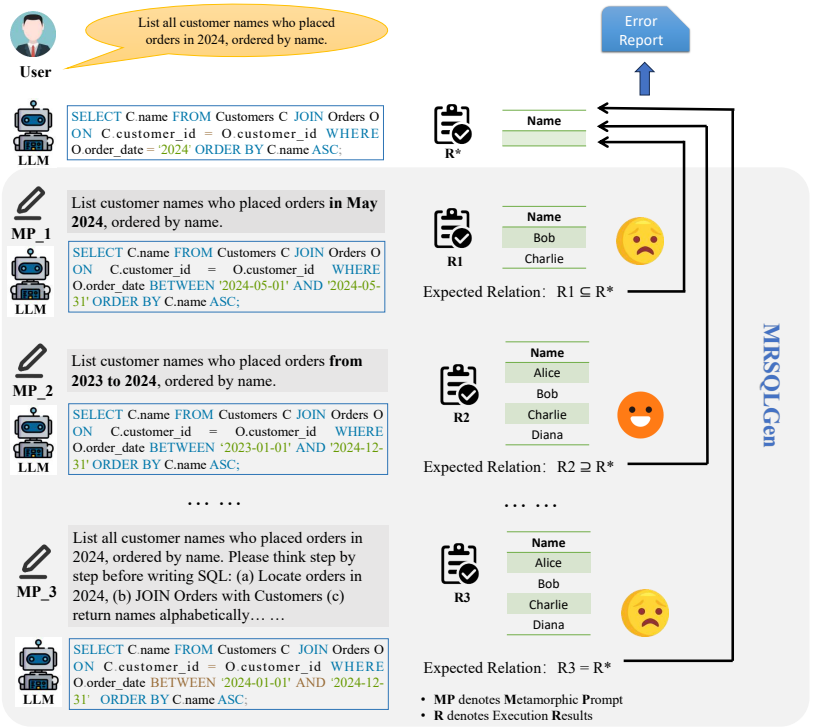

Fig. 2. **A Motivating Example**

prompt is first passed through a **Prompt Paraphrasing** module that generates a set of metamorphic prompts—reformulations that maintain a well-defined MR with the original prompt. These metamorphic prompts are then issued to the same LLM, generating multiple metamorphic queries. Each of these queries is executed against the same database to produce a set of corresponding results. Finally, all the results—including the original target result—are compared using a **Cross Validation** module to determine whether the outputs violate the behavioral expectations defined by the MRs. Our proposed framework, MRSQLGen, consists of two key modules.

Prompt Paraphrasing. This module addresses a key challenge in MP: how to effectively generate metamorphic prompts for a given NL2SQL task. To tackle this, we begin by conducting an empirical study to identify common hallucination patterns and their root causes, and organize them into a hallucination knowledge base (HKB). Given a new task, we retrieve the most likely hallucination types by considering both the prompt's semantics and the structure of the LLM-generated SQL. We then select and adapt metamorphic rules targeting these hallucination types to produce prompts that maintain a well-defined relationship with the original, enabling downstream validation.

Cross Validation. This module compares the outputs of the original and metamorphic queries to identify behavioral inconsistencies that indicate hallucinations. We adopt a balanced comparison strategy inspired by prior work [22], where a hallucination is flagged only if the majority of metamorphic queries violate their expected MRs with the original.

We describe each module in the following sections. Section 4 presents our empirical study motivating the HKB construction. Section 5 details the Prompt Paraphrasing module for generating task-specific metamorphic prompts. Section 6 introduces the Cross Validation module for detecting hallucinations via behavioral consistency checks.

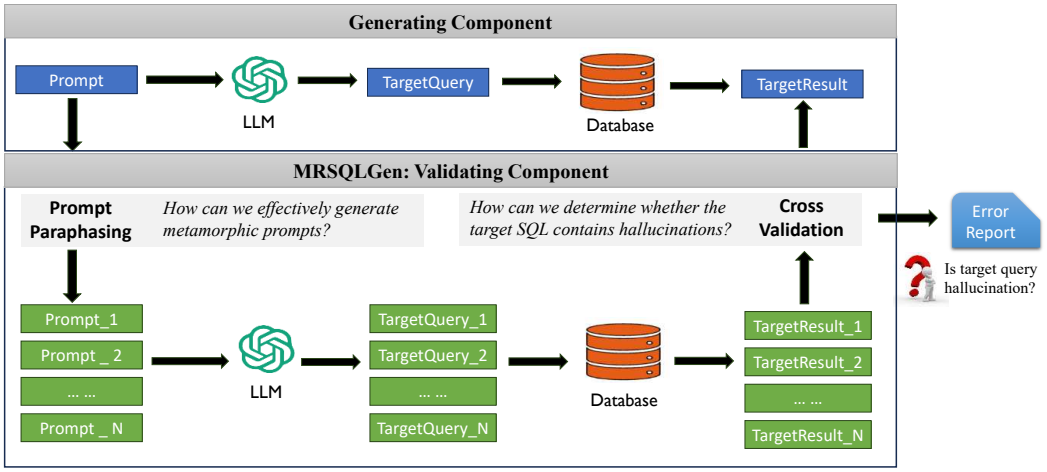

Fig. 3. **System Overview of MRSQLGen**

4 Hallucination Analysis in NL2SQL System

In this section, we empirically analyze the manifestations and root causes of LLM hallucinations in NL2SQL tasks. Our study aims to characterize the subtle manifestations and root causes of hallucinations in SQL generation, particularly intent-violating hallucinations, which often evade conventional detection. This analysis contributes to a deeper and more systematic understanding of the generative behaviors and failure modes of LLMs in NL2SQL applications.

4.1 Empirical Setup

4.1.1 Dataset. To better reflect real-world query demands and practical development settings, we adopt the widely used NL2SQL benchmark Spider [11] as the foundation for our hallucination analysis. Spider is a cross-domain semantic parsing dataset that contains 10,181 natural language questions and 5,693 unique complex SQL queries spanning 200 relational databases across 138 domains. The databases are sourced from real-world applications, ensuring that the query tasks exhibit realistic complexity and diversity. We sample 50% of the tasks from Spider. The sampling uses a stratified strategy, where we randomly select 50% of the tasks within each difficulty level to ensure balanced coverage.

4.1.2 Model and Prompt Selection. We evaluate two models: the closed-source GPT-4o-mini [44] and the open-source Deepseek-Coder-6.7B-Instruct [45]. We employ the in-context learning prompt strategy DAIL-SQL [46], which, as of January 2025, has achieved the best performance on the Spider benchmarks. All configurations follow the DAIL-SQL setup, including the nuclear sampling strategy and a temperature setting of 0.0. As a single NL question may correspond to multiple equivalent SQL queries, we adopt execution match (EM) as the correctness metric throughout all experiments. To quantify the risk of coincidental execution matches, we manually inspected 100 randomly sampled EX-matched cases whose generated SQL differs from the ground truth; 98% of them were found to be semantically correct, indicating that such cases are rare in practice.

4.1.3 Taxonomy Annotation. In order to analyze the hallucination types in the LLM-generated SQL queries, we manually perform open coding [47] on the generated SQL queries to obtain the hallucination taxonomy. Our annotation process comprises three main stages:

(1) **Initial Open Coding.** In the first stage, we randomly select 10% of the NL2SQL tasks for a preliminary analysis. For each selected task, multiple SQL snippets are generated by the evaluated

models and executed in the corresponding SQLite environment to verify their correctness. On this basis, the three authors compare the LLM-generated SQL queries against the ground-truth and discuss and record possible hallucination phenomena.

(2) **Preliminary Taxonomy Construction.** Based on the initial analysis, we document the different types of SQL hallucinations and the locations where hallucinated content occurs within queries. Since multiple types of hallucinations may appear within a single SQL snippet, three authors independently label the errors in the snippets and engage in discussions to reach a consensus. The annotation consistency is measured using Cohen's Kappa score, with a value of 0.95 indicating substantial agreement. Through this process, we group similar hallucinations to construct a preliminary taxonomy that categorizes various hallucination types and defines their implications in LLM-generated SQL queries.

(3) **Full Taxonomy Construction.** Building upon the preliminary taxonomy, the remaining SQL snippets are independently annotated by three experts with extensive experience in the database field (two with 6 years of experience and one with 4 years of experience). Any discrepancies in the annotations are resolved through discussion, and new error types are incorporated as necessary, with further expansion and refinement of the taxonomy through collaboration among the three authors. This manual analysis process, carried out by six domain experts, required considerable domain-specific knowledge of databases and SQL, and consumed approximately 600 hours in total.

4.2 Hallucination Taxonomy

After manually studying 3,968 incorrect SQL queries generated by studied models, we build a two-level error taxonomy with 25 types, as illustrated in Figure 4. This taxonomy classifies hallucinations into four major categories: (1) **Syntax Errors (3.08%)** refer to violations of the SQL grammar, such as misuse of keywords, missing quotes or parentheses, and other parser-level violations. These errors are relatively easy to identify since they are directly caught by SQL parsers and raise explicit exceptions during execution. (2) **Semantic Errors (15.73%)** are grammatically valid but fail during execution due to issues such as table-column mismatches, use of non-existent tables, or data type conflicts. These are also typically easy to detect in production settings, as database engines will return runtime exceptions when such issues occur. (3) **Intent-violating Hallucinations (77.62%)** are logically subtle: the generated queries may execute without errors, but they fail to match the user's original intent. These hallucinations are particularly insidious because they are syntactically and semantically valid but logically incorrect—thus much harder to detect automatically in practice. Importantly, this category accounts for a substantial proportion of all observed hallucinations, underscoring its critical impact on NL2SQL reliability. (4) Finally, the **Others (3.58%)** category covers output formatting issues, gold label hallucinations, and miscellaneous low-frequency errors. Due to space limitations, we primarily focus on the Intent-violating Hallucination category in our detailed analysis, as it presents the greatest challenge for reliable detection in real-world deployment scenarios. The complete taxonomy and annotation guidelines are available in our public repository [48].

Following, we break down the Intent-violating Hallucinations into ten subtypes, with percentages indicating their relative distribution, noting that each instance may belong to multiple subtypes.

C1: Operator Misuse (4.9%). This type of hallucination refers to the incorrect use of arithmetic, comparison, or logical operators in the generated SQL query, leading to query logic that deviates from the user's original intent. The hallucination typically stems from the model's misunderstanding of comparative or quantitative language expressions such as "more than," "not in," or "between." For instance, the phrase "more than 3 courses" may be incorrectly translated to `>= 3` instead of `> 3`.

C2: Limit Error (6.5%). This type of hallucination occurs when the model mishandles result cardinality control in the generated SQL query. It mainly manifests as missing `LIMIT` clauses,

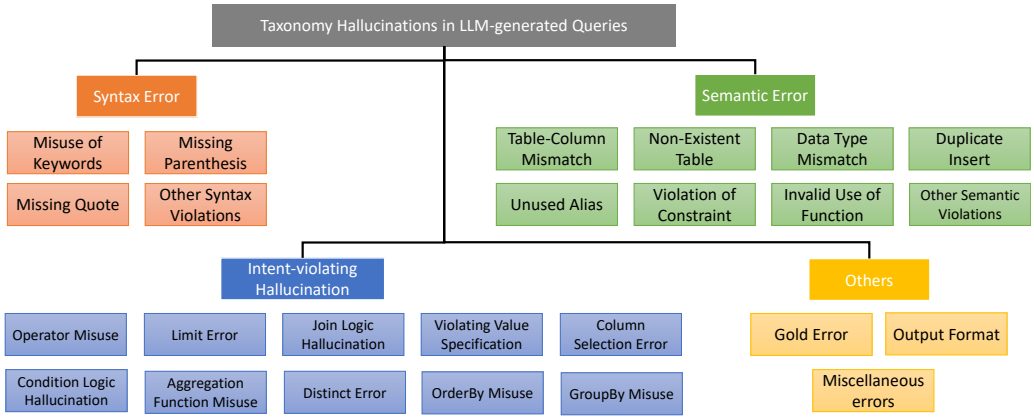

Fig. 4. **Taxonomy of Hallucinations in LLM-generated Queries**

incorrect values for "`LIMIT`" or "`OFFSET`", or the absence of supporting structures such as "`ORDER BY`", ultimately causing the result set to deviate from the actual intent expressed in natural language. The hallucination often arises from the model's misinterpretation or neglect of quantity-related semantics in natural language queries, such as "top 5", "first few rows", "rows 10 to 20", or "at most one result". For example, in response to the prompt "List the 2nd page of results, 10 per page", the model generates "`SELECT * FROM results ORDER BY score DESC LIMIT 10 OFFSET 5;`", where the correct offset should be 10 to skip the first 10 rows. The hallucinated query instead starts from the 6^{th} row, returning results inconsistent with the intended pagination logic.

C3: Join Logic Hallucination (38.7%). This hallucination refers to the incorrect construction of join logic in SQL queries, where the generated statement deviates from the intended multi-table semantics or introduces irrelevant, spurious, or logically inconsistent relationships between tables. The hallucination often arises from the model's misunderstanding of entity relationships, attribute ownership, or foreign key dependencies implied in natural language. Typical manifestations include omitted join relations (failing to connect necessary tables), spurious joins (introducing irrelevant tables), join type misuse (e.g., "`LEFT JOIN`" used instead of "`INNER JOIN`"), and faulty join predicates. For example, as shown in Figure 5, the model incorrectly uses "`c.id = o.id`" instead of the correct foreign key predicate "`c.id = o.customer_id`", leading to logically invalid join results.

C4: Violating Value Specification (12.9%). This type of hallucination occurs when the model generates incorrect values for SQL query conditions, due to misunderstanding the valid value range, format, or semantic constraints of the target fields in the database. Such errors commonly appear in `WHERE` or `HAVING` clauses. For example, as shown in Figure 1, the LLM incorrectly filters records using "`order_date = '2023'`" instead of the correct range-based condition "`BETWEEN '2023-01-01' AND '2023-12-31'`".

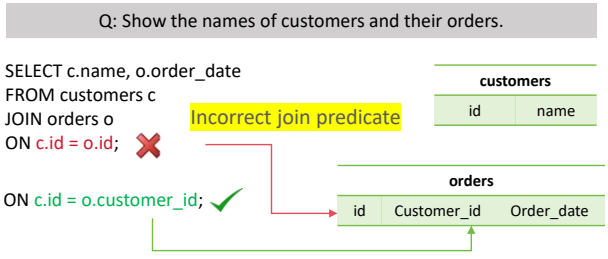

Fig. 5. **C3: Join Logic Hallucination: Faulty Predicate in LLM-Generated SQL**

C5: Column Selection Error (15.7%). This type of hallucination occurs when the model incorrectly selects columns in the `SELECT` clause. The issue may involve omitting required columns,

selecting extraneous ones, or confusing semantically similar fields. Such hallucinations often arise when the model fails to align entities mentioned in natural language with their corresponding schema fields. They are particularly common when field names are ambiguous across tables, or when the model lacks a precise understanding of column ownership and context.

C6: Condition Logic Hallucination (18.6%). This type refers to errors in constructing logical conditions, where the SQL query misrepresents the intended filtering logic due to incorrect operators, incomplete expressions, or ill-formed condition structures in clauses like `WHERE` or `JOIN`. Common manifestations include spurious conditions that were not mentioned in the prompt, underspecified conditions that fail to capture the complete filtering intent, misconstrained predicates with incorrect values or comparison operators, and faulty logical connections such as misused `AND`/`OR` or misplaced `NOT`. In more subtle cases, malformed condition structures—such as missing parentheses or incorrect nesting—lead to logical parsing errors while still producing executable queries. For example, the query in Figure 6 attempts to retrieve users with income over 5000 who live in either NY or LA, but due to missing parentheses, the generated SQL interprets `OR` with higher precedence than `AND`, incorrectly including all LA users regardless of income.

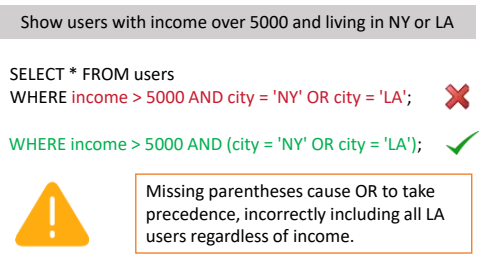

Fig. 6. **C6: Condition Logic Hallucination: Malformed Boolean Structure**

C7: Aggregation Function Misuse (4.5%). This type of hallucination occurs when the model applies incorrect aggregation operations in SQL queries. Such errors typically arise from misinterpreting the intended aggregation target, misunderstanding the summary method, or applying aggregation to inappropriate attributes. Common manifestations include the use of an aggregation function that does not match the user intent. For example, given a prompt to calculate the total quantity of ordered items, the model mistakenly uses "`COUNT(*)`" instead of the correct "`SUM(order_quantity)`", leading to a count of order rows rather than the intended item total. Other forms include applying aggregation to semantically invalid fields, or failing to properly handle aggregation alongside non-aggregated attributes.

C8: Distinct Error (14.7%). This type of hallucination refers to the incorrect use of the `DISTINCT` modifier in SQL queries, leading to unintended duplication or unintended loss of result records. Such errors occur when the model fails to correctly interpret whether uniqueness is required in the users' question.

C9: OrderBy Misuse (9.9%). This type of hallucination refers to errors in the construction of the `ORDER BY` clause, where the sorting logic expressed in natural language is incorrectly translated into SQL, resulting in output that does not match the expected order. Typical forms include omitting the `ORDER BY` clause when sorting is required, using incorrect sort attributes or directions, or misordering fields in multi-level sorting.

C10: GroupBy Misuse (15.9%). This type of hallucination refers to the incorrect use of the `GROUP BY` clause in SQL queries. Such errors typically stem from misinterpretation of the aggregation target, grouping unit, or statistical logic expressed in natural language. Common manifestations include applying aggregation functions without a required `GROUP BY` clause, grouping by incorrect fields that mismatch the expected result granularity, introducing unnecessary grouping in non-aggregation queries, or failing to aggregate non-grouped fields—potentially causing runtime errors or ambiguous semantics.

4.3 Root Cause Analysis and Observations

To better understand the underlying factors contributing to intent-violating hallucinations in LLM-generated SQL queries, we conduct a detailed analysis of both root causes and recurring structural patterns observed in our annotated corpus. This section outlines the primary cognitive and contextual limitations of LLMs, followed by empirical patterns that inspire the design of a hallucination-aware prediction module for downstream validation.

Root Cause #1: Intention Understanding Capacity. LLMs often lack deep comprehension of user intent and instead rely on surface-level pattern matching from training data [49]. Rather than accurately interpreting the specific semantic requirements of a given task, LLMs tend to reproduce frequently observed SQL patterns, which may only approximate the desired logic. This weakness explains hallucination types where the model misinterprets the semantics of comparative or quantitative language, such as *C1/C2/C6*. This limitation further manifests in aggregation-related hallucinations (*C7–C10*), which primarily stem from an insufficient understanding of the intended aggregation target and the appropriate semantic granularity for grouping.

Root Cause #2: Context Awareness Deficiency. LLMs often lack fine-grained awareness of database-specific context, such as schema constraints, data types, and value distributions. This limitation directly contributes to hallucinations involving schema grounding and value alignment, including *C3/C4/C5*. Without robust schema grounding, the model may hallucinate literals in invalid formats (e.g., filtering a `DATE` field with "`= '2024'`"), misalign natural language entities with the wrong columns, or construct faulty join predicates between unrelated tables. These issues illustrate that intent understanding alone is insufficient—accurate query generation also requires deep contextual reasoning over schema and data.

Observation #1: Error Co-occurrence and Overlap Patterns. We observe that hallucination types frequently co-occur within a single SQL query. For example, *C3* often induce *C5*, and *C1* commonly co-occurs with *C4*. These patterns suggest that hallucinations are not isolated phenomena but emerge from interdependent semantic failures, forming latent *error clusters* that are valuable for predictive modeling.

Observation #2: Prompt-level Inducing Factors. Certain prompt structures are more likely to induce hallucinations. Ambiguous quantifiers increase the likelihood of *C1/C2*, vague temporal expressions often lead to *C4*, and multi-objective queries (e.g., "list customers and count their orders") frequently induce *C7/C10*. These findings suggest that linguistic cues in prompts can serve as effective features for estimating hallucination risk and type.

Observation #3: SQL Structural Triggers. Hallucinations are more prevalent in SQL structures involving joins, aggregations, nested queries, or cardinality constraints. These structures provide greater degrees of freedom for generation, often leading to logically flawed but syntactically and semantically acceptable queries. This makes hallucinations in such structures more difficult to detect using traditional execution-based validation methods.

These empirical insights motivate a hallucination-type retrieval module that predicts the most likely hallucination types (*Observation #1*) based on a prompt and its corresponding SQL structure (*Observation #2 and Observation #3*). By leveraging co-occurrence patterns, prompt semantics, and SQL complexity, our framework proactively identifies vulnerable queries and selects targeted metamorphic rules for validation.

5 Prompt Paraphrasing

5.1 Hallucination Knowledge Base Builder

To enable targeted prompt transformation, we construct a structured Hallucination Knowledge Base (HKB) that captures empirical hallucination patterns in LLM-generated SQL queries. The

HKB serves as the foundation for selecting appropriate metamorphic rules, linking common failure modes with prompt-level transformation strategies.

The construction of the HKB is grounded in our annotated hallucination corpus. Each entry $(x, q, \mathcal{H})$ consists of three elements: (1) the natural language prompt x, (2) the corresponding LLM-generated SQL query q, and (3) a set of hallucination types $\mathcal{H}$ identified in q. To ensure consistent comparison and retrieval, both the prompt x and query q are preprocessed through a standardization pipeline that normalizes surface-level variations while preserving semantic structure. We detail this standardization process in Section 5.2. The hallucination types are based on a taxonomy derived from our empirical analysis. Since hallucinations often co-occur, $\mathcal{H}$ is represented as a set rather than a single label. These features allow us to embed the HKB into a searchable latent space, supporting similarity-based retrieval of past failure cases during inference. In later stages, this knowledge base allows us to retrieve the most relevant hallucination types for a new prompt and apply corresponding metamorphic transformation rules tailored to those hallucination risks.

5.2 Potential Hallucination Type Retrieval

To enable targeted metamorphic transformations, we retrieve potential hallucination types by querying the HKB. Figure 7 shows the overall retrieval workflow. Given a user question and its corresponding LLM-generated SQL query, we first normalize and embed both inputs to form a hybrid representation. These embeddings are fused into a hybrid representation via a multi-head cross-attention module to capture semantic alignment. We then use this hybrid embedding to query the HKB and retrieve the top-k most similar historical cases. Each case is annotated with a set of hallucination types in HKB. Since cases may contain multiple types, we aggregate all retrieved labels and apply a voting strategy to rank them by frequency. The top-r types are selected as the most likely hallucination risks for the input.

Prompt and SQL Normalization. To enable structure-aware matching and reduce surface-level variability, we perform normalization on both the user question and the LLM-generated SQL query. The goal is to abstract away irrelevant lexical differences while preserving the core semantics, ensuring robust retrieval and matching in the HKB. For SQL normalization, we parse the query into an abstract syntax tree (AST) using SQLGLOT [50], and apply a depth-first traversal to replace node values with canonical placeholders. Specifically, we normalize schema-bound table names, column aliases, literals and expression aliases into generic tokens. For prompt normalization, we use SPACY [51] to identify named entities and numbers in the question text. Nouns and proper nouns are replaced with `[NOUN]`, while numeric tokens are replaced with `[NUMBER]`. This aligns with the SQL normalization strategy, enabling unified hybrid embedding for prompt-query pairs during hallucination type retrieval.

Retrieval Process. Given the normalized prompt $\tilde{x}$ and SQL query $\tilde{q}$, we first obtain their contextualized token-level embeddings using a shared bidirectional Transformer encoder:

$$\mathbf{h}_{\tilde{x}} = \texttt{Encode}(\tilde{x}), \quad \mathbf{h}_{\tilde{q}} = \texttt{Encode}(\tilde{q}), \quad \mathbf{h}_{\tilde{x}}, \mathbf{h}_{\tilde{q}} \in \mathbb{R}^{L \times d}$$

To capture cross-modal semantic alignment, we apply a bidirectional multi-head cross-attention mechanism. Specifically, each modality is used as the query to attend to the other:

$$\mathbf{h}^{x \to q} = \texttt{MHA}(\mathbf{h}_{\tilde{x}}, \mathbf{h}_{\tilde{q}}, \mathbf{h}_{\tilde{q}}), \quad \mathbf{h}^{q \to x} = \texttt{MHA}(\mathbf{h}_{\tilde{q}}, \mathbf{h}_{\tilde{x}}, \mathbf{h}_{\tilde{x}})$$

The final fused representation is obtained by concatenating the attention outputs along the sequence dimension, followed by mean pooling:

$$\mathbf{z} = \texttt{MeanPool}\left(\texttt{Concat}(\mathbf{h}^{x \to q}, \mathbf{h}^{q \to x})\right) \in \mathbb{R}^{d}$$

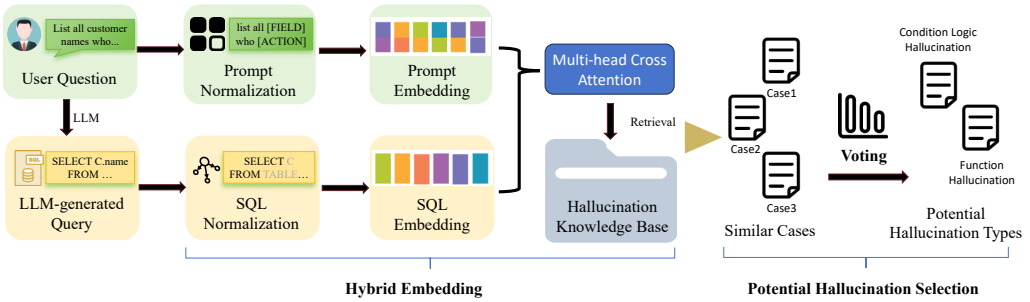

Fig. 7. **Workflow of Potential Hallucination Type Retrieval**

We compare this global embedding $\mathbf{z}$ with all entries in the HKB, each of which stores a precomputed embedding $\mathbf{z}_i$, and compute their cosine similarity:

$$\text{sim}_i = \cos(\mathbf{z}, \mathbf{z}_i), \quad i = 1, \ldots, N$$

The top-k most similar entries are selected as candidate matches for hallucination type inference.

Voting Strategy. Each retrieved HKB unit is associated with one or more hallucination types. Let $\mathcal{H}_i$ denote the hallucination type set of the i-th retrieved match. To compute the aggregated confidence for a specific hallucination type τ, we perform similarity-weighted scoring across the top-k matches:

$$\text{score}(\tau) = \frac{1}{k} \sum_{i=1}^{k} \mathbb{I}[\tau \in \mathcal{H}_i] \cdot \text{sim}_i$$

where $\mathbb{I}[\cdot]$ is the indicator function. Hallucination types with scores exceeding a threshold θ_{sim} are selected as final predictions:

$$\mathcal{H}_{\text{pred}} = \{\tau \mid \text{score}(\tau) \geq \theta_{\text{sim}}\}$$

This soft aggregation strategy enables robust hallucination type identification by integrating partial signals from semantically similar examples.

Hyperparameter Settings. In our experiments, we set the number of retrieved cases $k = 10$ and the similarity threshold $\theta_{\text{sim}} = 0.75$. The number of predicted hallucination types r varies dynamically depending on how many type scores exceed the threshold. These settings strike a balance between precision and recall in hallucination type retrieval and are fixed across all downstream experiments.

5.3 MR Rules Construction

Based on our empirical analysis, we design 26 targeted metamorphic rules for each hallucination type and organize them into four categories of prompt transformation strategies; due to space limitations, the full rule set is available in our GitHub repository [48]. Table 2 summarizes these MR categories, along with their transformation strategies, targeted hallucination types, and examples.

- **QMR-1: Intent Perturbation & Reversal.** This category introduces slight modifications to the original intent—such as changing comparison operators, reversing sorting directions, or altering temporal constraints—to assess whether the generated SQL query adapts appropriately. Violations of expected behavioral changes indicate semantic hallucinations.
- **QMR-2: Logical Decomposition.** Complex or compound prompts are rewritten into step-by-step instructions that explicitly decompose multi-part logic, such as filtering, grouping, or joining. This strategy guides the model toward more robust reasoning paths and mitigates shortcut-based generation.

Table 2. **Metamorphic Rule Categories for Prompt Transformation in NL2SQL.**

MR Category	Targeted Types	Example Transformation
QMR-1: Intent Perturbation & Reversal	C1, C4, C6	"List products above 100" → "List products below 100"
QMR-2: Logical Decomposition	C6, C7, C10	"Find users in NY or LA with income > 5000" → "Step 1: Filter income > 5000. Step 2: Filter location is NY or LA"
QMR-3: Constraint Explicitization	C2, C6, C9	"Return the second page of results" → "Assume 10 results per page. Return rows 11–20 sorted by time."
QMR-4: Error-Type Reflection	All	"Show orders in 2024" → "Show orders in 2024. Then verify whether the SQL query introduces a value specification hallucination."

- **QMR-3: Constraint Explicitization.** This category clarifies implicit constraints within the original prompt, such as pagination logic, result limits, or sorting criteria, by reformulating vague instructions into precise control parameters. It helps prevent hallucinations caused by underspecified or ambiguous expressions.
- **QMR-4: Error-Type Reflection.** This category integrates meta-level reminders or preemptive warnings based on known hallucination types. It simulates user feedback or domain knowledge to guide the model away from historically frequent failure modes.

To detect hallucinations, each prompt transformation is linked to a MR—such as equivalence, subset, or superset—which specifies the expected consistency or deviation in SQL execution results. Given a selected MR rule and few-shot exemplars illustrating its transformation pattern, we employ an auxiliary LLM to generate the corresponding metamorphic prompt. The original and metamorphic queries are then compared using these predefined relationships in the Cross Validation module to determine whether the original query exhibits intent-violating hallucination. Due to space limitations, we provide detailed descriptions and concrete examples of HKB construction and MR rule usage in supplementary material [52].

6 Cross-Validation

Once the original and metamorphic SQL queries are generated and executed against the same database, MRSQLGen performs cross validation to detect hallucinations based on behavioral consistency. A straightforward strategy might flag a hallucination whenever any metamorphic query yields a different result from the original. However, due to the inherent uncertainty in LLM reasoning, such strict consistency can lead to high false positives. To reduce sensitivity to such noise, we adopt a majority-based voting strategy inspired by prior work [22], and tailored to the metamorphic prompting paradigm. Specifically, each prompt transformation is associated with a predefined MR that defines the expected relation between the output of the original and the transformed queries (e.g., equivalence, subset, superset). MRSQLGen checks whether each metamorphic query satisfies its corresponding MR with the original result. A hallucination is flagged only if the proportion of violated relationships exceeds a threshold, which our experiments identify as 80% to best balance precision and recall. This approach tolerates minor deviations while still identifying cases where the original query is semantically unstable under controlled intent-preserving variations—indicating an intent-violating hallucination.

7 Evaluation

Our evaluation is designed to answer the following research questions (RQs):

- **RQ1:** Can MRSQLGen effectively detect intent-violating hallucinations in LLM-generated SQL? How does MRSQLGen compare to state-of-the-art hallucination detection methods?
- **RQ2:** How do the Prompt Paraphrasing and Cross Validation modules contribute to detection performance?

7.1 Experimental Setup

7.1.1 Dataset. We conduct hallucination detection experiments on two widely-used NL2SQL benchmarks: Spider [11] and Bird [23]. Spider is a cross-domain dataset comprising 10,181 natural language questions over 200 databases. In our empirical study (Section 4), we used half of it to build the HKB. To ensure unbiased evaluation and demonstrate the generalizability of our method, we use the remaining half for hallucination detection experiments. To further alleviate distribution-specific concerns, we evaluate MRSQLGen not only on Spider but also on Bird, a substantially different benchmark in terms of schema, domain, and linguistic characteristics, which consists of 10,962 text-SQL pairs across 95 databases and features more complex SQL structures [23].

Both datasets provide ground truth SQL annotations, allowing us to determine whether an LLM-generated query constitutes a hallucination. Specifically, we use the Execution Accuracy (EX) metric—whether the generated query produces the same result as the ground-truth query on the database—to label hallucinated outputs. A generated SQL is considered intent-violating hallucinated if it produces a valid output but deviates semantically from the ground-truth (i.e., fails EX while being executable). For each question, we use an LLM to generate a corresponding SQL query. We then execute the query and discard cases with syntax or runtime errors (non-executable queries), since they fall outside the scope of intent-violating hallucination detection. The remaining executable-but-possibly-hallucinated queries form our hallucination detection dataset.

7.1.2 Evaluation Metrics. We use the following metrics to evaluate the effectiveness of intent-violating hallucination detection in MRSQLGen:

- **Precision.** Precision is the percentage of hallucinations correctly identified by MRSQLGen out of all queries it classifies as hallucinations. It measures how accurately the system detects hallucinations. A higher precision indicates fewer false positives [28].
- **Recall.** Recall is the percentage of actual hallucinations detected by MRSQLGen. It measures the system's ability to find all hallucinations. A higher recall indicates fewer false negatives [28].
- **F1 Score.** The F1 score is the harmonic mean of precision and recall, providing a balanced evaluation when both accuracy and completeness are important. It reflects the overall effectiveness of the hallucination detection method [28].

7.1.3 Models. Our primary test subject is GPT-4o. OpenAI's GPT-4o is a black-box LLM and represents a noteworthy milestone in the progress of transformer-based language models [53]. We also evaluate four representative LLMs, including three open-source models (CodeLlama-13B-Instruct [54], Deepseek-Coder-V2:16B [45], and Qwen2.5-Coder-14B [55]) and one closed-source models (GPT-4o-mini [53]). These models are selected for their strong performance on NL2SQL tasks, as well as their widespread adoption in both academic and industrial settings. The temperature of all LLMs was set to 0.0 in all experiments to reduce randomness.

7.1.4 Baselines. We compare MRSQLGen against two representative baselines.

- **SelfCheckGPT** [19] is a state-of-the-art confidence-based hallucination detector originally designed for open-ended text generation. It estimates the likelihood of hallucination by sampling multiple responses from the LLM and measuring their consistency with the original output. To adapt SelfCheckGPT to SQL generation, we treat the LLM-generated SQL as the base query and compute token-level n-gram overlap between the base and sampled queries. Following the original setup, we use a sampling temperature of 1.0 and apply a threshold-based rule to classify a query as hallucinated if its sampled variants show low consistency.
- **LLM-as-a-judge** [56] leverages the LLM to reflect on and assess the hallucination of its generated output. In our evaluation, we consider two LLM-as-a-judge settings. **LLM-as-a-Judge Self**

(**LAJ-Self**), which uses the same model that generates the SQL to judge its own output, and **LLM-as-a-Judge GPT-5.1 (LAJ-G5)**, which employs a stronger external judge, GPT-5.1, to assess the generated SQL. In both settings, the judge model is prompted to determine whether the SQL accurately captures the user intent expressed in the input natural language query, without access to any external ground truth or execution feedback. Specifically, the model is instructed to assign a binary label: 1 if the query is considered hallucinated, and 0 otherwise. To maintain diversity, we set temperature 0.5 and perform multiple sampling rounds and apply majority voting over the predicted labels to make the final decision.

7.1.5 Environment. We perform our experiments on a workstation an 8-core "11th Gen Intel(R) Core(TM) i7-11700@2.50GHz" processor, 64GB of memory, and NVIDIA RTX A6000 with 48GB of VRAM running Ubuntu 22.04.1 LTS. For open-source LLMs, we deploy a local API server based on vLLM [57] which is a unified library for LLM serving and inference. To ensure a fair comparison, all methods use the same underlying LLM across experiments.

7.2 RQ1: Effectiveness and Comparison with Baselines

Table 3 summarizes the results of MRSQLGen against SelfCheckGPT (SCG) and two *LLM-as-a-judge* variants, LAJ-Self and LAJ-G5, on the Spider and Bird benchmarks. Overall, MRSQLGen consistently delivers superior precision, recall, and F1 scores across all evaluated models.

Table 3. **Comparison between MRSQLGen, SelfCheckGPT, and LLM-as-a-judge**

Dataset	GPT-4o				GPT-4o-mini				CodeLlama				Deepseek-Coder				Qwen2.5-Coder			
Metric	MRS	SCG	LAJ-Self	LAJ-G5	MRS	SCG	LAJ-Self	LAJ-G5	MRS	SCG	LAJ-Self	LAJ-G5	MRS	SCG	LAJ-Self	LAJ-G5	MRS	SCG	LAJ-Self	LAJ-G5
Spider (P)	0.782	0.200	0.357	0.716	0.651	0.294	0.577	0.571	0.687	0.358	0.276	0.257	0.478	0.454	0.366	0.205	0.651	0.291	0.666	0.167
Spider (R)	0.560	0.083	0.074	0.369	0.690	0.059	0.180	0.400	0.973	0.372	0.225	1.000	0.883	0.187	0.317	1.000	0.754	0.112	0.089	1.000
Spider (F1)	0.653	0.117	0.123	0.487	0.670	0.099	0.275	0.470	0.805	0.365	0.248	0.409	0.620	0.265	0.339	0.340	0.699	0.162	0.157	0.286
BIRD (P)	0.696	0.666	0.619	0.604	0.803	0.564	0.562	0.623	0.905	0.280	0.302	0.351	0.816	0.583	0.565	0.519	0.701	0.621	0.545	0.531
BIRD (R)	0.692	0.124	0.349	0.454	0.714	0.108	0.407	0.617	0.986	0.421	0.220	0.791	0.933	0.241	0.278	0.717	0.818	0.250	0.152	0.558
BIRD (F1)	0.694	0.209	0.446	0.518	0.756	0.182	0.472	0.620	0.944	0.336	0.254	0.486	0.870	0.341	0.373	0.602	0.755	0.356	0.238	0.544

Note: MRS = MR-SQLGen, SCG = SelfCheckGPT, LAJ-Self = model self-judging, LAJ-G5 = judged by GPT-5.1. P = Precision, R = Recall, F1 = F1 Score.

On the Spider dataset, MRSQLGen achieves the highest F1 scores across all models. For example, with GPT-4o, MRSQLGen obtains an F1 score of 0.653, substantially outperforming SelfCheckGPT (0.117) as well as both *LLM-as-a-judge* variants, including LAJ-Self (0.123) and the stronger external-judge setting LAJ-G5 (0.487). Similar trends are observed with open-source models such as CodeLlama and Deepseek-Coder, where MRSQLGen consistently yields higher precision and recall, indicating that it can detect a broader range of intent-violating hallucinations while maintaining a low false positive rate. On the Bird dataset, which contains more complex NL2SQL tasks, MRSQLGen still demonstrates strong generalization capability. For instance, with CodeLlama, MRSQLGen achieves an F1 score of 0.944, compared to 0.336 for SelfCheckGPT, 0.254 for LAJ-Self, and 0.486 for LAJ-G5. These results indicate that although our hallucination taxonomy is derived from Spider—a widely recognized and representative NL2SQL benchmark—we intentionally define hallucination types at the level of general SQL semantics (e.g., aggregation granularity and condition-logic correctness), rather than schema- or dataset-specific patterns. The strong performance on the more complex Bird dataset demonstrates that the taxonomy and the derived metamorphic rules generalize well beyond the dataset used for annotation. Paired bootstrap significance tests ($p < 0.05$) further validate that the observed improvements of MRSQLGen over both baselines are statistically significant.

When compared with existing approaches, MRSQLGen shows clear advantages. Across both datasets, MRSQLGen achieves consistent improvements in F1 scores, with gains ranging from 112.1% to 576.8% over SelfCheckGPT, from 55.6% to 430.9% over the self-judging setting (LAJ-Self), and from 21.9% to 144.4% over the stronger external-judge setting (LAJ-G5). The weaker

performance of SelfCheckGPT mainly stems from its reliance on sampling multiple variants of the same prompt. LLMs often generate nearly identical SQL queries across these samples, limiting diversity and preventing the method from exposing subtle semantic inconsistencies, which leads to low recall. Similarly, LLM-as-a-judge exhibits inherent limitations on structured tasks such as NL2SQL. In the self-judging setting (LAJ-Self), the model tends to suffer from self-confirmation bias, overestimating the correctness of its own generated SQL and remaining insensitive to fine-grained intent deviations. Using a stronger external judge (LAJ-G5) partially alleviates this issue by improving recall and recovering many previously missed hallucinations; however, it still lacks direct access to execution semantics and relies primarily on post-hoc semantic reasoning. In contrast, MRSQLGen leverages metamorphic prompting to generate intent-preserving transformations that guide the model along diverse reasoning paths, while the cross-validation module ensures consistency checks at the execution level. This combination allows MRSQLGen to achieve higher recall without sacrificing precision, delivering more reliable and accurate hallucination detection in NL2SQL scenarios.

Answer to RQ1: MRSQLGen can effectively detect intent-violating hallucinations in LLM-generated SQL, achieving consistently higher precision, recall, and F1 across all baselines.

7.3 RQ2: Ablation Study

To quantify the benefit of our designs, we conduct an ablation study focusing on the Prompt Paraphrasing and Cross Validation modules. In the first variant, MRSQLGen-PP, we replace the Prompt Paraphrasing module with a naive strategy that prompts the LLM to generate ten equivalent metamorphic prompts, without using the HKB to retrieve specific hallucination types for tailoring MR rules. In the second variant, MRSQLGen-CV, we adjust the cross-validation strategy to flag hallucinations immediately upon detecting any inconsistency, instead of requiring a majority violation. Due to computational constraints, all ablation experiments are evaluated only on GPT-4o.

Table 4. **Ablation study of MRSQLGen on Spider and BIRD using GPT-4o**

Method	Spider			Improvement (F1)	BIRD			Improvement (F1)
	P	R	F1		P	R	F1	
MRSQLGen	0.782	0.560	0.653	–	0.696	0.692	0.694	–
MRSQLGen-PP	0.800	0.454	0.579	+12.8%	0.600	0.540	0.568	+22.2%
MRSQLGen-CV	0.726	0.572	0.640	+2.0%	0.491	0.872	0.628	+10.5%

Note: P = Precision, R = Recall, F1 = F1 Score. Improvement (F1) is the percentage gain of MRSQLGen over each ablation.

Table 4 presents the ablation study results. When the Prompt Paraphrasing module is replaced with a naive strategy (MRSQLGen-PP), the F1 score of MRSQLGen improves by approximately 12.8% on Spider and 22.2% on Bird. This indicates that the HKB-guided retrieval mechanism plays a critical role in generating targeted metamorphic prompts that effectively expose intent-violating hallucinations. Without this targeted paraphrasing, the transformations become less diverse, reducing the ability to detect subtle hallucinations. Similarly, when the Cross Validation strategy is relaxed, MRSQLGen shows an improvement of approximately 2.0% on Spider and 10.5% on Bird. This finding suggests that a majority agreement across transformed prompts is crucial for improving robustness and reducing false positives. Without this vote mechanism, the framework tends to over-flag queries, leading to degraded precision and overall detection reliability.

Answer to RQ2: Both the Prompt Paraphrasing and Cross Validation modules are essential, as removing either notably reduces F1 and weakens hallucination detection.

8 Discussion

8.1 System Overhead, Complexity and Practical Deployment

We evaluate the computational overhead of MRSQLGen in comparison with SelfCheckGPT and LLM-as-a-judge, using GPT-4o as the underlying model for all methods. We focus on the self-judging setting (LAJ-Self), which ensures a fair and model-agnostic comparison. As shown in Table 5, MRSQLGen incurs higher token usage due to metamorphic prompt generation, but its overall cost remains comparable to LAJ-Self and within a practical range. On Spider, MRSQLGen requires 7.9k tokens and 53.9 seconds per task, introducing about 13 seconds of additional latency over LAJ-Self (40.9s) while improving F1 from 0.123 to 0.653 (+0.53). On Bird, MRSQLGen incurs a similar overhead (73.4s vs. 62.4s) and achieves an F1 gain of +0.248 (0.694 vs. 0.446). In contrast, SelfCheckGPT reduces computational cost but suffers substantially lower accuracy. Overall, MRSQLGen offers a favorable accuracy–efficiency trade-off for practical NL2SQL deployment.

In practice, the engineering overhead of MRSQLGen is modest. First, maintaining the hallucination knowledge base (HKB) is lightweight, as it consists of a compact and largely static set of hallucination patterns and transformation templates. Importantly, the same HKB generalizes well across both Spider and Bird without dataset-specific re-engineering. Second, prompt/SQL normalization as well as hybrid embedding-based retrieval are low-cost preprocessing steps that execute within milliseconds and contribute negligible additional runtime overhead. As a result, MRSQLGen can be integrated as an online validation layer in existing NL2SQL systems with minimal engineering effort.

Table 5. **Token Usage and Runtime Comparison on GPT-4o**

Dataset	MRSQLGen		SelfCheckGPT		LAJ-Self	
	Spider	BIRD	Spider	BIRD	Spider	BIRD
token	7,925	13,489	2,389	4,636	7,725	26,173
time(s)	53.92	73.39	29.45	48.01	40.88	62.40

Note: time(s) reports the average inference time per task.

8.2 Failure Case Analysis

We conduct a qualitative analysis of 100 randomly sampled cases where MRSQLGen fails to correctly detect hallucinations, and identify two dominant failure modes. First, inaccurate hallucination-type retrieval from the HKB can lead to misaligned MR rule selection, causing MRSQLGen to apply transformations that do not target the actual faulty logic in the original query and resulting in false negatives. Second, even when the correct hallucination type is identified, the auxiliary LLM may introduce new hallucinations during metamorphic query generation, especially under structurally complex transformations, which can violate the intended MR constraints and lead to false positives. Concrete examples illustrating both failure modes are provided in the supplementary material [58].

8.3 Threats to Validity

Internal Validity. Our empirical study relies on manual annotation of hallucination types, which may introduce subjective bias. To mitigate this, three authors independently labeled the data and resolved disagreements through discussion to ensure consistency.

External Validity. The external threats to validity stem from the limited number of LLMs used in our evaluation. While more test subjects could potentially improve the generalizability of our findings, we have selected five widely adopted models—spanning both proprietary and open-source variants—that are representative of current LLM development. In future work, we plan to extend our evaluation to additional models to further validate the robustness of MRSQLGen.

9 Related Work

NL2SQL Systems and Benchmarks. NL2SQL automatically translates natural language questions to SQL queries, bridging the gap between non-expert users and databases and enabling applications like intelligent database services, automated data analysis, and database question-answering. Recent LLM-based approaches have improved SQL generation accuracy via in-context learning and fine-tuning [59–63]. These models are typically evaluated on benchmark datasets such as WikiSQL [64] and Spider [11]. However, these benchmarks rely on ground-truth queries and metrics like Exact Match (EM) and Execution Accuracy (EX), limiting evaluation to controlled settings and failing to address real scenarios with no canonical solution. In contrast, we propose a comprehensive testing methodology and evaluation tool to validate LLM-generated SQL queries in real time without external resources.

Hallucinations in LLMs. The term "hallucination" in NLP describes generated text that is nonsensical or deviates from the source content [65]. Hallucinations are classified as intrinsic (contradicting the source) and extrinsic (unverifiable from the source) [66]. This concept extends to code generation, where LLMs may produce code beyond intended instructions. A recent taxonomy defines three types of code hallucinations: (i) task requirement conflicts (code failing the prompt intent), (ii) factual knowledge conflicts (violating programming knowledge, e.g., API misuse), and (iii) project context conflicts (code incompatible with the project environment) [67]. These align with input, knowledge, and context misalignments akin to NLP. However, there is a lack of research on hallucination phenomena in the field of NL2SQL systems. To bridge this gap, we conduct a comprehensive empirical study of hallucinations in NL2SQL systems, with the goal of informing the design of our approach for efficiently detecting hallucinations in real-world development scenarios.

Hallucination Detection Techniques. Existing methods mainly fall into two categories: *confidence estimation*, which relies on model-internal signals [68–72], and *LLM-as-a-judge*, which uses either smaller detectors or external LLMs as evaluators [65, 73–76]. While effective in NLG tasks, these approaches are less practical for code-related tasks such as NL2SQL, where reliance on internal signals or external evaluators limits scalability and adaptability. In this paper, we propose a hallucination detection method that requires no external resources or model confidence scores, enabling real-time identification of SQL hallucinations in practice.

10 Conclusion

In this paper, we present MRSQLGen, a novel hallucination detection framework for NL2SQL tasks based on the metamorphic prompting paradigm. MRSQLGen systematically rewrites input prompts using transformation rules and detects intent-violating hallucinations by comparing execution results under predefined metamorphic relationships. Experimental results on Spider and Bird benchmarks demonstrate that MRSQLGen consistently outperforms strong baselines, including SelfCheckGPT and LLM-as-a-judge, across multiple LLMs.

Data Availability

The source code of MRSQLGen is available at https://github.com/MRSQLGen/MRSQLGen.

Acknowledgement

We sincerely thank the anonymous reviewers for their valuable and insightful feedback. This research was supported by the Natural Science Foundation of China (Grant No. 62272400) and Fujian Provincial Natural Science Foundation of China (Grant No. 2025J010002). Li Lin began this work while affiliated with Xiamen University. Now he is a Ph.D. student at Zhejiang University. Rongxin Wu is the corresponding author and works as a member of Xiamen Key Laboratory of Intelligent Storage and Computing in Xiamen University.

References

[1] "pandas," https://github.com/pandas-dev/pandas, 2024, accessed: 2025-03-04.

[2] S. Agarwal, G. Y.-Y. Chan, S. Garg, T. Yu, and S. Mitra, "Fast natural language based data exploration with samples," in *Companion of the 2023 International Conference on Management of Data*, 2023, pp. 155–158.

[3] R. J. L. John, D. Bacon, J. Chen, U. Ramesh, J. Li, D. Das, R. Claus, A. Kendall, and J. M. Patel, "Datachat: An intuitive and collaborative data analytics platform," in *Companion of the 2023 International Conference on Management of Data*, 2023, pp. 203–215.

[4] X. V. Lin, R. Socher, and C. Xiong, "Bridging textual and tabular data for cross-domain text-to-sql semantic parsing," *arXiv preprint arXiv:2012.12627*, 2020.

[5] J. Jiang, H. Xie, Y. Shen, Z. Zhang, M. Lei, Y. Zheng, Y. Fang, C. Li, D. Huang, W. Zhang *et al.*, "Siriusbi: Building end-to-end business intelligence enhanced by large language models," *arXiv preprint arXiv:2411.06102*, 2024.

[6] J. Lian, X. Liu, Y. Shao, Y. Dong, M. Wang, Z. Wei, T. Wan, M. Dong, and H. Yan, "Chatbi: Towards natural language to complex business intelligence sql," *arXiv preprint arXiv:2405.00527*, 2024.

[7] J. Sen, F. Ozcan, A. Quamar, G. Stager, A. Mittal, M. Jammi, C. Lei, D. Saha, and K. Sankaranarayanan, "Natural language querying of complex business intelligence queries," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1997–2000.

[8] Y. Song, S. Ezzini, X. Tang, C. Lothritz, J. Klein, T. Bissyandé, A. Boytsov, U. Ble, and A. Goujon, "Enhancing text-to-sql translation for financial system design," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*, 2024, pp. 252–262.

[9] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models," in *Chi conference on human factors in computing systems extended abstracts*, 2022, pp. 1–7.

[10] H. Ding, V. Kumar, Y. Tian, Z. Wang, R. Kwiatkowski, X. Li, M. K. Ramanathan, B. Ray, P. Bhatia, S. Sengupta *et al.*, "A static evaluation of code completion by large language models," *arXiv preprint arXiv:2306.03203*, 2023.

[11] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," *arXiv preprint arXiv:1809.08887*, 2018.

[12] ANTLR Project, "ANTLR: ANother Tool for Language Recognition," https://www.antlr.org/, 2025, accessed: 2025-06-05.

[13] T. Mao, "Sqlglot: An open source sql parser, transpiler, and optimizer," https://github.com/tobymao/sqlglot, 2022, accessed: 2025-07-07.

[14] C. Li, B. Bi, M. Yan, W. Wang, and S. Huang, "Addressing semantic drift in generative question answering with auxiliary extraction," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 2021, pp. 942–947.

[15] J. Maynez, S. Narayan, B. Bohnet, and R. McDonald, "On faithfulness and factuality in abstractive summarization," *arXiv preprint arXiv:2005.00661*, 2020.

[16] M. Huang, X. Zhu, and J. Gao, "Challenges in building intelligent open-domain dialog systems," *ACM Transactions on Information Systems (TOIS)*, vol. 38, no. 3, pp. 1–32, 2020.

[17] N. Varshney, W. Yao, H. Zhang, J. Chen, and D. Yu, "A stitch in time saves nine: Detecting and mitigating hallucinations of llms by validating low-confidence generation," *arXiv preprint arXiv:2307.03987*, 2023.

[18] J.-Y. Yao, K.-P. Ning, Z.-H. Liu, M.-N. Ning, Y.-Y. Liu, and L. Yuan, "Llm lies: Hallucinations are not bugs, but features as adversarial examples," *arXiv preprint arXiv:2310.01469*, 2023.

[19] P. Manakul, A. Liusie, and M. J. Gales, "Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models," *arXiv preprint arXiv:2303.08896*, 2023.

[20] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin *et al.*, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," *ACM Transactions on Information Systems*, vol. 43, no. 2, pp. 1–55, 2025.

[21] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in Neural Information Processing Systems*, vol. 36, pp. 46 595–46 623, 2023.

[22] J. C. Yang, D. Dalisan, M. Korecki, C. I. Hausladen, and D. Helbing, "Llm voting: Human choices and ai collective decision-making," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, vol. 7, 2024, pp. 1696–1708.

[23] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo *et al.*, "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*, vol. 36, pp. 42 330–42 357, 2023.

[24] Z. Q. Zhou, D. Huang, T. Tse, Z. Yang, H. Huang, and T. Chen, "Metamorphic testing and its applications," in *Proceedings of the 8th International Symposium on Future Software Technology (ISFST 2004)*. Software Engineers Association Xian, China, 2004, pp. 346–351.

[25] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," *arXiv preprint arXiv:2002.12543*, 2020.

[26] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on software engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[27] N. Li, Y. Li, Y. Liu, L. Shi, K. Wang, and H. Wang, "Drowzee: Metamorphic testing for fact-conflicting hallucination detection in large language models," *Proceedings of the ACM on Programming Languages*, vol. 8, no. OOPSLA2, pp. 1843–1872, 2024.

[28] W. Wu, Y. Cao, N. Yi, R. Ou, and Z. Zheng, "Detecting and reducing the factual hallucinations of large language models with metamorphic testing," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 1432–1453, 2025.

[29] B. Yang, M. A. Al Mamun, J. M. Zhang, and G. Uddin, "Hallucination detection in large language models with metamorphic relations," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 425–445, 2025.

[30] P. Ma, S. Wang, and J. Liu, "Metamorphic testing and certified mitigation of fairness violations in nlp models." in *IJCAI*, vol. 20, 2020, pp. 458–465.

[31] C. Tsigkanos, P. Rani, S. Müller, and T. Kehrer, "Large language models: The next frontier for variable discovery within metamorphic testing?" in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2023, pp. 678–682.

[32] P. Xue, L. Wu, Z. Yang, X. Li, Z. Yu, Z. Jin, G. Li, Y. Xiao, and J. Wu, "Exploring and lifting the robustness of llm-powered automated program repair with metamorphic testing," *arXiv preprint arXiv:2410.07516*, 2024.

[33] S. Hyun, M. Guo, and M. A. Babar, "Metal: Metamorphic testing framework for analyzing large-language model qualities," in *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2024, pp. 117–128.

[34] X. Wang and D. Zhu, "Validating llm-generated programs with metamorphic prompt testing," *arXiv preprint arXiv:2406.06864*, 2024.

[35] D. Rai and Z. Yao, "An investigation of neuron activation as a unified lens to explain chain-of-thought eliciting arithmetic reasoning of llms," *arXiv preprint arXiv:2406.12288*, 2024.

[36] J. Dong, J. Sun, W. Zhang, J. S. Dong, and D. Hao, "Contested: Consistency-aided tested code generation with llm," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, pp. 596–617, 2025.

[37] M. A. Team, "Mistral-7b-instruct-v0.2," https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2, 2023, accessed: 2025-08-14.

[38] S. Zhou, T. Li, K. Wang, Y. Huang, L. Shi, Y. Liu, and H. Wang, "Understanding the effectiveness of coverage criteria for large language models: A special angle from jailbreak attacks," *arXiv preprint arXiv:2408.15207*, 2024.

[39] T. Y. Chen, P.-L. Poon, and X. Xie, "Metric: Metamorphic relation identification based on the category-choice framework," *Journal of Systems and Software*, vol. 116, pp. 177–190, 2016.

[40] C.-A. Sun, A. Fu, P.-L. Poon, X. Xie, H. Liu, and T. Y. Chen, "Metric: A metamorphic relation identification technique based on input plus output domains," *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1764–1785, 2019.

[41] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz-Corts, "Metamorphic testing of restful web apis," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 882–882.

[42] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey, "Metamorphic relations for enhancing system understanding and use," *IEEE Transactions on Software Engineering*, vol. 46, no. 10, pp. 1120–1154, 2018.

[43] C. Xu, V. Terragni, H. Zhu, J. Wu, and S.-C. Cheung, "Mr-scout: Automated synthesis of metamorphic relations from existing test cases," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 6, pp. 1–28, 2024.

[44] OpenAI, "Chatgpt: Optimizing language models for dialogue," https://openai.com/blog/chatgpt, 2022, accessed: 2025-03-09.

[45] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[46] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *arXiv preprint arXiv:2308.15363*, 2023.

[47] S. H. Khandkar, "Open coding," *University of Calgary*, vol. 23, no. 2009, 2009.

[48] unknown, "MRSQLGen," unknown. [Online]. Available: https://github.com/MRSQLGen/MRSQLGen

[49] F. Mu, L. Shi, S. Wang, Z. Yu, B. Zhang, C. Wang, S. Liu, and Q. Wang, "Clarifygpt: Empowering llm-based code generation with intention clarification," *arXiv preprint arXiv:2310.10996*, 2023.

[50] E. J. Smith, "Sqlglot: An open source python sql parser, transpiler, and engine," https://sqlglot.com/sqlglot.html, 2021, accessed: 2025-08-06.

[51] E. AI, "spacy: Industrial-strength natural language processing in python," https://spacy.io/, 2015, accessed: 2025-08-06.

[52] MRSQLGen Authors, "Detailed hkb construction and mr rules usage for mrsqlgen," https://github.com/MRSQLGen/FSE26-Rebuttal-Material/blob/main/Details%20of%20HKB%20Construction%20and%20MR%20Rules%20Usage.md, 2026, supplementary material.

[53] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.
[54] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.
[55] "Qwen2.5-coder-14b," 2024. [Online]. Available: https://huggingface.co/Qwen/Qwen2.5-Coder-14B
[56] R. Wang, J. Guo, C. Gao, G. Fan, C. Y. Chong, and X. Xia, "Can llms replace human evaluators? an empirical study of llm-as-a-judge in software engineering," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, pp. 1955–1977, 2025.
[57] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.
[58] MRSQLGen Authors, "Failure case analysis for mrsqlgen," https://github.com/MRSQLGen/FSE26-Rebuttal-Material/blob/main/Failure%20Cases%20Analysis.md, 2026, supplementary material.
[59] M. Pourreza and D. Rafiei, "Din-sql: Decomposed in-context learning of text-to-sql with self-correction," *Advances in Neural Information Processing Systems*, vol. 36, pp. 36 339–36 348, 2023.
[60] Y. Xie, X. Jin, T. Xie, M. Lin, L. Chen, C. Yu, L. Cheng, C. Zhuo, B. Hu, and Z. Li, "Decomposition for enhancing attention: Improving llm-based text-to-sql through workflow paradigm," *arXiv preprint arXiv:2402.10671*, 2024.
[61] B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, L. Chai, Z. Yan, Q.-W. Zhang, D. Yin, X. Sun *et al.*, "Mac-sql: A multi-agent collaborative framework for text-to-sql," *arXiv preprint arXiv:2312.11242*, 2023.
[62] 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.
[63] R. Sun, S. Ö. Arik, A. Muzio, L. Miculicich, S. Gundabathula, P. Yin, H. Dai, H. Nakhost, R. Sinha, Z. Wang *et al.*, "Sql-palm: Improved large language model adaptation for text-to-sql (extended)," *arXiv preprint arXiv:2306.00739*, 2023.
[64] W. Hwang, J. Yim, S. Park, and M. Seo, "A comprehensive exploration on wikisql with table-aware word contextualization," *arXiv preprint arXiv:1902.01069*, 2019.
[65] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, "Survey of hallucination in natural language generation," *ACM computing surveys*, vol. 55, no. 12, pp. 1–38, 2023.
[66] F. Liu, Y. Liu, L. Shi, H. Huang, R. Wang, Z. Yang, L. Zhang, Z. Li, and Y. Ma, "Exploring and evaluating hallucinations in llm-powered code generation," *arXiv preprint arXiv:2404.00971*, 2024.
[67] Z. Zhang, C. Wang, Y. Wang, E. Shi, Y. Ma, W. Zhong, J. Chen, M. Mao, and Z. Zheng, "Llm hallucinations in practical code generation: Phenomena, mechanism, and mitigation," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, pp. 481–503, 2025.
[68] D. D. Johnson, D. Tarlow, D. Duvenaud, and C. J. Maddison, "Experts don't cheat: learning what you don't know by predicting pairs," *arXiv preprint arXiv:2402.08733*, 2024.
[69] T. Xia, B. Yu, Y. Wu, Y. Chang, and C. Zhou, "Language models can evaluate themselves via probability discrepancy," *arXiv preprint arXiv:2405.10516*, 2024.
[70] S. Lin, J. Hilton, and O. Evans, "Teaching models to express their uncertainty in words," *arXiv preprint arXiv:2205.14334*, 2022.
[71] Y. Abbasi Yadkori, I. Kuzborskij, A. György, and C. Szepesvari, "To believe or not to believe your llm: Iterative prompting for estimating epistemic uncertainty," *Advances in Neural Information Processing Systems*, vol. 37, pp. 58 077–58 117, 2024.
[72] L. Kuhn, Y. Gal, and S. Farquhar, "Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation," *arXiv preprint arXiv:2302.09664*, 2023.
[73] Y. Wang, Z. Yu, Z. Zeng, L. Yang, C. Wang, H. Chen, C. Jiang, R. Xie, J. Wang, X. Xie *et al.*, "Pandalm: An automatic evaluation benchmark for llm instruction tuning optimization," *arXiv preprint arXiv:2306.05087*, 2023.
[74] T. Wang, P. Yu, X. E. Tan, S. O'Brien, R. Pasunuru, J. Dwivedi-Yu, O. Golovneva, L. Zettlemoyer, M. Fazel-Zarandi, and A. Celikyilmaz, "Shepherd: A critic for language model generation," *arXiv preprint arXiv:2308.04592*, 2023.
[75] J. Wang, Y. Liang, F. Meng, Z. Sun, H. Shi, Z. Li, J. Xu, J. Qu, and J. Zhou, "Is chatgpt a good nlg evaluator? a preliminary study," *arXiv preprint arXiv:2303.04048*, 2023.
[76] T. Kocmi and C. Federmann, "Large language models are state-of-the-art evaluators of translation quality," *arXiv preprint arXiv:2302.14520*, 2023.

Received 2025-09-10; accepted 2025-12-22