DialectEvo: Bridging SQL Dialects with Execution-Guided Knowledge Refinement

Jianling Gao
jianlingg@buaa.edu.cn
SKLCCSE, Beihang University
Beijing, China

Chongyang Tao
chongyang@buaa.edu.cn
SKLCCSE, Beihang University
Beijing, China

Liu Yang
yangliu26@buaa.edu.cn
SKLCCSE, Beihang University
Beijing, China

Xiya Jiang
jiangxiya@bupt.edu.cn
Beijing University of Posts and Telecommunications
Beijing, China

Shuai Ma*
mashuai@buaa.edu.cn
SKLCCSE, Beihang University
Beijing, China

ABSTRACT

SQL dialect translation is essential for database interoperability, yet it remains challenging in practice: rule-based transpilers are brittle to long-tail dialect discrepancies, while large language models (LLMs) are flexible but prone to non-executable or semantically incorrect outputs. Existing systems also mainly operate at the query level, without accumulating reusable knowledge from past failures. In this paper, we present DialectEvo, an execution-guided framework for SQL dialect translation with iterative rule refinement. DialectEvo first performs deterministic translation, then invokes an LLM with bounded self-reflection when needed, and verifies translations through actual database execution. Crucially, DialectEvo does not treat failures as isolated query-level errors; instead, it converts them into reusable cross-dialect transformation rules maintained in an evolving rule base. Experiments across six dialect pairs show that DialectEvo consistently improves translation reliability over existing methods. These results suggest that execution-guided knowledge refinement provides an effective path from query-level correction to robust and reusable SQL dialect translation.

PVLDB Reference Format:
Jianling Gao, Chongyang Tao, Liu Yang, Xiya Jiang, and Shuai Ma. DialectEvo: Bridging SQL Dialects with Execution-Guided Knowledge Refinement. PVLDB, 19(12): 4850 - 4853, 2026.
doi:10.14778/3827998.3828138

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/JerryGao818/DialectEvo.

1 INTRODUCTION

Modern data systems exhibit a rich ecosystem of SQL dialects, including MySQL, PostgreSQL, Oracle Database, Hive, and emerging cloud-native engines. Although these systems share a common SQL foundation, they diverge substantially in syntax, built-in functions, data types, and semantic behaviors. As a result, migrating queries across database systems remains a challenging and error-prone task, particularly in scenarios such as system modernization, multi-engine benchmarking, and cloud portability. SQL dialect translation is therefore a fundamental problem in database interoperability.

In practice, organizations often rely on manual rewriting, rule-based transpilers [1, 3, 4], or large language models (LLMs) [6] to translate queries. Rule-based transpilers rely on manually crafted transformation rules derived from expert knowledge. While deterministic and interpretable, such systems require significant engineering effort and often struggle to cover long-tail dialect variations. On the other hand, LLMs offer flexible and powerful translation capabilities, but may generate syntactically invalid or semantically incorrect queries, especially when dialect-specific constraints are subtle. More importantly, LLM-based approaches typically treat translation as an isolated generation task: when errors occur, the correction process does not naturally evolve into reusable knowledge. Recent systems such as CrackSQL [7] incorporate execution feedback to iteratively repair SQL queries. While effective for improving translation correctness through query-level refinement, these approaches do not explicitly model or accumulate reusable dialect transformation knowledge across queries.

In this paper, we present DialectEvo, an execution-guided framework that transforms translation failures into reusable dialect-level rules. Instead of treating translation as a one-shot generation or per-query repair problem, DialectEvo models cross-dialect differences as an evolving knowledge base of transformation rules. Starting from an initial rule set automatically generated from dialect tutorials and documentation, DialectEvo performs rule-based translation with fallback generation and validates results through real database execution. When translation failures occur, the system does not merely repair the query; rather, it analyzes error patterns, abstracts dialect discrepancies, and updates its rule repository. Over time, this execution-driven feedback loop enables the system to accumulate reusable dialect knowledge, progressively reducing translation errors and improving robustness across diverse queries. This design shifts the focus from query-level correction to knowledge-level evolution. By continuously discovering and refining transformation rules, DialectEvo builds a self-improving translation framework that generalizes beyond individual query fixes.

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828138

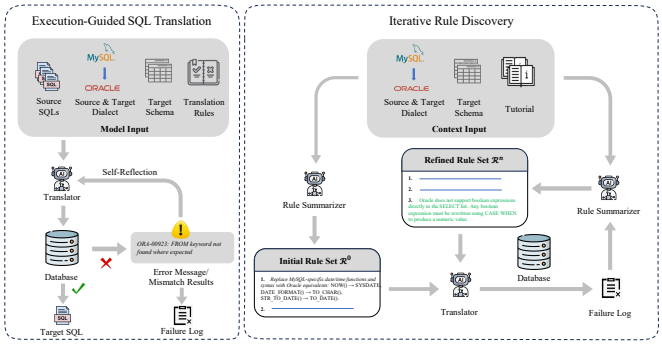

Figure 1: DialectEvo System Architecture

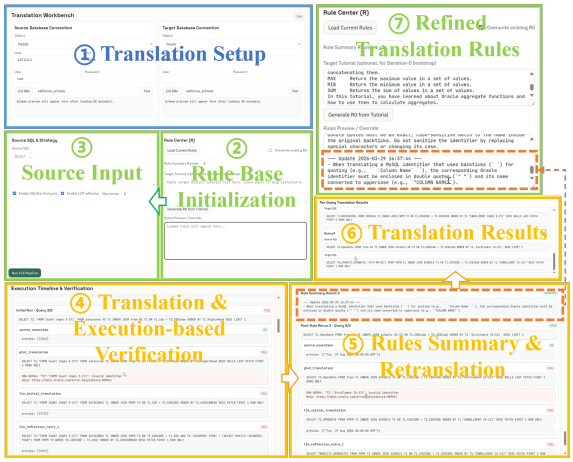

Figure 2: Interactive Interface of DialectEvo

2 METHODOLOGY

2.1 Problem Formulation & System Overview

The goal of DialectEvo is to translate a query Q_s written in a source dialect $\mathcal{L}_s$ into an equivalent query Q_t in a target dialect $\mathcal{L}_t$. We formally denote the translation function as $Q_t = \mathcal{T}(Q_s, C, \mathcal{R})$, where C represents the contextual schema information and $\mathcal{R}$ denotes the evolving set of dialect transformation rules.

As illustrated in Figure 1, DialectEvo operates through two synergistic processes: (1) *Execution-Guided Translation*, which generates and validates queries using a hybrid of deterministic rules and LLM-based reasoning; (2) *Iterative Rule Discovery*, which refines $\mathcal{R}$ based on execution feedback.

2.2 Execution-Guided Translation

2.2.1 Hybrid Translation Pipeline. The translation module adopts a hybrid strategy to balance efficiency and flexibility. Given an input query Q_s, the system first attempts a deterministic translation using a rule-based transpiler (e.g., SQLGlot [3]), denoted as $f_{rule}(Q_s)$. This handles standard syntactic mappings efficiently. If f_{rule} fails or if the generated query fails the verification phase, the system invokes the LLM-based agent. The LLM agent generates the target query Q_t conditioned on a prompt $\mathcal{P}$:

$$Q_t = \text{LLM}_{Translator}(\mathcal{P} \mid Q_s, \mathcal{S}_{tgt}, \mathcal{R}) \tag{1}$$

where $\mathcal{S}_{tgt}$ is the target schema, and $\mathcal{R}$ is the retrieved subset of translation rules relevant to the query's operators.

2.2.2 Execution-based Verification. DialectEvo employs an execution-based verification mechanism. Let $E(Q, D)$ denote the execution result of query Q on database D. A translation is deemed successful if and only if: (1) Executability: $E(Q_t, D_t)$ does not return a runtime error; and (2) Data Consistency: the target execution result matches the reference result under our consistency protocol.

Remark on Consistency Constraints. Existing Text-to-SQL benchmarks such as BIRD [5] commonly compute EX by comparing result sets. For SQL dialect translation, this conventional result matching can be too permissive, because it may ignore duplicate-row multiplicity and the output order required by explicit ordering clauses (e.g., `ORDER BY`). DialectEvo therefore uses order-insensitive multiset equivalence for unordered queries, which preserves duplicate rows, and list equivalence when explicit ordering is present. This ordered-aware protocol reduces false positives relative to conventional result-set matching. Nevertheless, it remains an execution-based criterion rather than a formal test of semantic equivalence: logically different queries may coincide on the current instances, while semantically valid translations may fail due to DBMS-specific behaviors such as default ordering, NULL handling, type coercion, or function semantics. Thus, reported EX should be interpreted as empirical evidence under our benchmark databases and consistency protocol, not as a complete guarantee of semantic equivalence.

2.2.3 Self-Reflection with Error Feedback. The system enters a multi-turn self-reflection loop when $E(Q_t, D_t) \neq E(Q_s, D_s)$. We construct a feedback object F_k at iteration k, containing the generated SQL, the error message, or a diff of the result sets. The LLM is then prompted by $\mathcal{P}^{(k)}$ to refine the query:

$$Q_t^{(k+1)} = \text{LLM}_{Translator}(\mathcal{P}^{(k)} \mid Q_s, Q_t^{(k)}, F_k, \mathcal{R}) \tag{2}$$

This process repeats until the verification passes or the maximum retry limit K is reached. If the translation ultimately fails, the tuple $(Q_s, Q_t^{(K)}, F_K)$ is recorded in the *Failure Log* $\mathcal{L}_{fail}$ for rule discovery.

2.3 Iterative Rule Discovery

To avoid repeatedly repairing similar failures at the query level, DialectEvo accumulates failures into reusable dialect rules through an iterative rule discovery process.

2.3.1 Initial Rule Generation ($\mathcal{R}^0$). Before processing queries, a Rule Summarizer Agent initializes the knowledge base by analyzing official documentation and tutorials for the source and target dialects. It identifies fundamental syntactic and semantic discrepancies of source and target dialects (e.g., `MySQL:DATE_ADD` → `Oracle: + INTERVAL`), forming the initial rule set $\mathcal{R}^0$.

2.3.2 Feedback-Driven Rule Refinement. The key idea of DialectEvo is failure-to-rule accumulation: rather than treating failed translations as isolated cases, it clusters the failure logs $\mathcal{L}_{fail}$, identifies uncovered error patterns, and abstracts them into reusable transformation rules. The refinement process is modeled as:

$$\mathcal{R}^{n+1} = \text{LLM}_{Summarizer}(\mathcal{R}^n, \text{Cluster}(\mathcal{L}_{fail}), \text{Docs}) \tag{3}$$

Table 1: Comparison of Execution Accuracy (%) across different dialect pairs. For each dialect pair "X ↔ Y", results are reported as "X→Y / Y→X". Bold indicates the best performance. "DialectEvo (Static)" refers to using pre-summarized rules without runtime updates. "–" denotes unsupported dialect pairs.

Method *#SQL*	PG ↔ MySQL 1501	PG ↔ Oracle 1393	MySQL ↔ Oracle 1491	Hive ↔ PG 1403	Hive ↔ MySQL 1403	Hive ↔ Oracle 1392
SQLGlot	97.20 / 83.68	78.98 / 78.71	72.50 / 92.49	79.69 / 78.69	82.89 / 79.33	76.51 / 77.73
jOOQ	85.21 / 77.48	80.46 / 85.56	71.16 / 81.89	–	–	–
Ora2Pg	– / 70.75	– / 80.79	–	–	–	–
SQLines	90.61 / 79.08	20.15 / 79.92	21.66 / 95.64	83.11 / 64.22	89.52 / 72.27	19.90 / 68.61
GPT	95.27 / 87.80	89.72 / 97.25	85.85 / 94.70	93.73 / 78.62	98.22 / 79.12	87.72 / 79.24
CrackSQL	99.13 / 91.74	91.20 / 98.12	87.59 / 96.24	–	–	–
DialectEvo (Static)	99.40 / 98.53	99.06 / 98.79	98.46 / 99.40	99.64 / 98.65	99.00 / 98.22	99.07 / 99.07
DialectEvo	**99.47** / **98.67**	**99.60** / **99.40**	**98.73** / **99.80**	**99.79** / **99.07**	**99.22** / **98.93**	**99.57** / **99.14**

The system clusters failed queries by error type (e.g., "Timestamp Formatting", "Join Semantics"). For each cluster, the LLM analyzes the gap between the source intent and the failed target execution, consults the documentation, and synthesizes a new or corrected rule. This updated rule set $\mathcal{R}^{n+1}$ is then injected into the translation agent's prompt context for subsequent queries, enabling the system to self-improve and avoid repeating past mistakes.

3 DEMONSTRATION

Our demo presents the full DialectEvo workflow in a single interactive page, allowing users to observe both end-to-end SQL translation and the evolution of reusable cross-dialect knowledge. As shown in Figure 2, the interface consists of a system overview, a *Translation Workbench*, and a *Results* panel.

The workflow starts in the *Translation Workbench*, where users specify the source and target dialects, configure database connections, inspect schema metadata from both databases, and provide source SQL queries. Users can also control whether to enable the SQLGlot first pass, the LLM-based reflection stage, and the maximum number of retry rounds. A central component is the *Rule Center*, which exposes the knowledge evolution process of DialectEvo. Before translation, users may initialize the rule base $\mathcal{R}^0$ by pasting a target-dialect tutorial and invoking the rule summarizer. The interface supports previewing, editing, and overriding the current rules, as well as setting the number of rule summarization rounds. After the pipeline is launched, DialectEvo performs batch translation and verification. Each query is first handled by deterministic translation, then refined by the LLM with bounded self-reflection when necessary, and finally checked through execution-based verification. The *Execution Timeline & Verification* panel visualizes stage-level traces such as failures, mismatches, and retries, while the metric cards and stage-wise chart summarize how many queries are resolved by deterministic translation, LLM refinement, and each rule-summary round. When failures remain, the demo highlights the key distinction of DialectEvo: it aggregates the failures into reusable rules, updates the rule base, and applies the refined knowledge in subsequent translation attempts. Finally, the *Per-Query Translation Results* panel presents the translated SQL or remaining failure reason for each query. Overall, the demo makes the process of failure-to-rule accumulation directly observable: users can inspect failed translations, track how they are summarized into rules, and rerun queries with a refined rule base.

4 MODEL PERFORMANCE

4.1 Experimental Setup

Dataset and Ground Truth. We evaluate DialectEvo on the development set of the BIRD benchmark [5], which contains real-world SQL queries. Since BIRD is originally grounded in SQLite, extending it to other dialects (PostgreSQL, MySQL, Oracle, and Hive) is non-trivial. To obtain a reliable multi-dialect evaluation set, we adopt a consensus-based curation process that combines automatic execution checking with manual inspection. Specifically, we retain only those queries whose translated forms can be validated across dialects and yield equivalent outputs under our verification protocol. This filtering process is intended to identify query instances with reliable cross-dialect semantics for fair comparison, resulting in approximately 1,400 to 1,500 query pairs per dialect mapping.

Baselines. We compare our system against three categories of baselines: (1) Rule-based Transpilers: We include industry-standard tools SQLGlot [3], jOOQ [1], SQLines [4], and Ora2Pg [2] (specialized for Oracle/MySQL to PostgreSQL). (2) LLM-based Translation: We use GPT-5-mini with a direct translation prompt as a strong LLM baseline. (3) Hybrid Systems: We compare with CrackSQL [7], a recent neuro-symbolic approach.

Implementation Details. For DialectEvo, we employ GPT-5-mini as the backend for the Translation Agent and Gemini-3-pro-preview for the Rule Summarizer Agent. The maximum number of reflection retries is set to 3. We report Execution Accuracy (EX) under our verification protocol detailed in Section 2.2.2.

4.2 Main Results

Table 1 summarizes the translation accuracy across 6 dialect pairs. Our analysis yields three key observations:

(1) Robustness Across Heterogeneous Dialects. Rule-based systems (e.g., SQLGlot, SQLines) exhibit significant performance volatility. While they achieve reasonable accuracy on common pairs like PG→MySQL (~90-97%), their performance degrades sharply on complex pairs involving Oracle or Hive (e.g., SQLines drops

Table 2: Ablation study of DialectEvo. Execution accuracies (%) are reported in the format "X→Y / Y→X" for each dialect pair.

Method	PG ↔ MySQL	PG ↔ Oracle	MySQL ↔ Oracle	Hive ↔ PG	Hive ↔ MySQL	Hive ↔ Oracle
#SQL	1501	1393	1491	1403	1403	1392
SQLGlot	97.20 / 83.68	78.98 / 78.71	72.50 / 92.49	79.69 / 78.69	82.89 / 79.33	76.51 / 77.73
+ LLM	98.73 / 94.00	86.10 / 98.59	87.32 / 98.93	95.15 / 81.11	98.43 / 82.18	91.74 / 82.40
+ Feedback	99.40 / 98.27	98.19 / 99.33	96.58 / 99.73	98.86 / 91.52	99.07 / 92.23	98.71 / 92.74
+ Rule (1)	99.47 / 98.53	99.33 / 99.40	97.18 / 99.80	99.64 / 97.65	99.22 / 98.50	99.50 / 98.99
+ Rule (2)	99.47 / 98.67	99.53 / 99.40	98.73 / 99.80	99.79 / 98.43	99.22 / 98.50	99.50 / 98.99
+ Rule (3)	99.47 / 98.67	99.60 / 99.40	98.73 / 99.80	99.79 / 99.07	99.22 / 98.93	99.57 / 99.14

to 20.15% for PG→Oracle). This is primarily because handcrafted rules struggle to cover the semantic gaps in non-standard functions (e.g., date arithmetic or window functions). In contrast, DialectEvo consistently achieves high execution accuracy (around 99%) across all 12 scenarios. This demonstrates that our execution-guided evolution mechanism effectively bridges the gap between syntactically diverse dialects that traditional transpilers fail to handle.

(2) Superiority over Direct LLM Generation. Although GPT-5-mini shows strong generalization with accuracies ranging from 78% to 98%, it still suffers from hallucination issues, often generating plausible but non-executable SQLs for edge cases. DialectEvo outperforms the direct LLM baseline by a substantial margin (improving accuracy by up to 19.90 percentage points in pairs like Oracle→Hive). This gain validates the necessity of our hybrid architecture: the rule summarizer effectively "grounds" the LLM with verified dialect knowledge, while the reflection loop catches and corrects runtime errors that a single-pass generation would miss.

(3) Efficacy of Learned Rules (Static vs. Dynamic). To assess the quality of the accumulated rules, we evaluate the DialectEvo (Static) configuration, which uses the finalized rule set $\mathcal{R}^n$ without updates during testing. As shown in Table 1, the Static variant performs comparably to the fully dynamic system, with less than 1% accuracy drop. This result suggests that the gains of DialectEvo do not come solely from online refinement; rather, repeated failures can be distilled into reusable rules. It also indicates that once such rules are accumulated, DialectEvo can maintain high translation reliability with much less dependence on continuous rule discovery.

4.3 Ablation Study

To quantify the contribution of each component in DialectEvo, we conduct a stepwise ablation study. Table 2 details the performance evolution as we incrementally enable the LLM fallback, the execution feedback loop, and the iterative rule discovery mechanism.

Effect of Hybrid Translation (+LLM). The base rule-based transpiler (SQLGlot) struggles with complex dialect pairs, such as MySQL→Oracle (72.50%). Introducing the LLM as a fallback for failed rule-based attempts significantly improves coverage. For instance, in Oracle→PG, the accuracy jumps from 78.71% to 98.59%. This confirms that LLMs successfully handle syntactic variations that are too rigid or ambiguous for static transpilers to process.

Impact of Execution Feedback (+Feedback). While the LLM improves initial generation, it is prone to subtle semantic errors. Enabling the self-reflection loop with execution feedback yields substantial gains, particularly for difficult targets. Notably, in PG→Oracle, accuracy surges from 86.10% to 98.19%, and in Hive→PG, it improves from 95.15% to 98.86%. This validates that error messages serve as a critical signal for the agent to correct hallucinations and align with the target database's strict execution requirements.

Convergence of Rule Discovery (+Rule 1–3). The final rows of Table 2 show the efficacy of our *Iterative Rule Discovery* module:

- **Rapid Learning (Round 1):** After a round of rule summarization, we observe a sharp reduction in remaining errors. For example, Oracle→Hive accuracy increases from 92.74% to 98.99%. This indicates that the Rule Summarizer successfully abstracts high-frequency error patterns into reusable knowledge.
- **Knowledge Convergence (Rounds 2–3):** Subsequent iterations show diminishing returns but continue to fix long-tail edge cases. By Round 3, performance stabilizes across most dialect pairs, suggesting that DialectEvo captures most reusable dialect rules with only a limited number of refinement rounds. The performance of DialectEvo (static) method in Table 1 also confirms the effectiveness of the learned knowledge.

In summary, while the hybrid translation and feedback loop provide a strong foundation, the iterative rule discovery is the decisive factor that enables DialectEvo to achieve high reliability by transforming transient errors into permanent knowledge.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Grant No. U22B2021, U24B20143, and 62572034), with support from the State Key Laboratory of Complex & Critical Software Environment (SKLCCSE).

REFERENCES

[1] 2026. *JOOQ. (Tool).* https://www.jooq.org/ Accessed: 2026-03-01.
[2] 2026. *Ora2Pg. (Tool).* https://ora2pg.darold.net/ Accessed: 2026-03-01.
[3] 2026. *SQLGlot. (Tool).* https://sqlglot.com/ Accessed: 2026-03-01.
[4] 2026. *SQLines. (Tool).* https://www.sqlines.com/ Accessed: 2026-03-01.
[5] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2023. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems* 36 (2023), 42330–42357.
[6] Amadou Latyr Ngom and Tim Kraska. 2024. MALLET: SQL dialect translation with llm rule generation. In *Proceedings of the Seventh International Workshop on Exploiting Artificial Intelligence Techniques for Data Management.* 1–5.
[7] Wei Zhou, Yuyang Gao, Xuanhe Zhou, and Guoliang Li. 2025. Cracking SQL barriers: An llm-based dialect translation system. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–26.