Structured EXPLAIN Feedback Improves SQL Generation with a 4B SLM on Industrial Time-Series Data

Byounghoon Son
Department of Computer Engineering, Hanbat National University
Daejeon, Republic of Korea

Minsung Jung
Department of Computer Engineering, Hanbat National University
Daejeon, Republic of Korea

Hyeonseok Jang
Department of Computer Engineering, Hanbat National University
Daejeon, Republic of Korea

Beomdo Park
Department of Computer Engineering, Hanbat National University
Daejeon, Republic of Korea

Gihun Gil
Department of Computer Engineering, Hanbat National University
Daejeon, Republic of Korea

Junseong Park
Department of Computer Engineering, Hanbat National University
Daejeon, Republic of Korea

Minu Baek
Department of Computer Engineering, Hanbat National University
Daejeon, Republic of Korea

Yeojin Jang
Department of Computer Engineering, Hanbat National University
Daejeon, Republic of Korea

Sangkeum Lee
Department of Computer Engineering, Hanbat National University
Daejeon, Republic of Korea

Abstract

Iterative self-correction with external feedback is a common strategy for improving LLM-generated SQL, yet prior work only compares feedback *presence* against *absence*, leaving the effect of feedback *granularity* unexplored. We present a controlled study on a fine-tuned 4B SLM generating SQL over industrial electricity data (3M records, 8 sectors). We combine LoRA fine-tuning with EXPLAIN-based iterative correction and compare three feedback levels: generic failure signal, database error message, and structured EXPLAIN output with schema hints.

Our central finding is that feedback granularity controls **convergence speed**, not **accuracy ceiling**: all three levels reach statistically indistinguishable format compliance (86–88%, McNemar $p \geq 0.75$), but structured feedback requires 20% fewer inference calls (71 vs. 89 per 100 queries). Fine-tuning provides the largest gain (38% → 80% ID, 34% → 69% OOT), with EXPLAIN retry adding a further +10/+8 percentage points.

We additionally show that a learned 270M critic fails as a SQL verifier—passing 80% of invalid queries even with clean training data—while deterministic EXPLAIN validation has zero false negatives for syntactic errors. A preliminary cross-model check on Qwen-2.5 3B shows the same OOT convergence point (.69) despite a large base performance gap, suggesting the generalization ceiling is data-constrained rather than model-constrained. Our results suggest that, at least in this setting, feedback design primarily affects compute efficiency rather than accuracy, offering initial evidence toward practical SLM-based SQL generation in on-premise industrial settings—a core capability for AI Data Scientists deployed under data-sensitivity constraints.

CCS Concepts

• **Computing methodologies** → **Natural language processing**; *Transfer learning*; • **Information systems** → **Structured Query Language**.

Keywords

text-to-SQL, small language model, self-correction, EXPLAIN feedback, LoRA fine-tuning, AI data scientist, industrial time-series data

1 Introduction

Natural language interfaces to databases promise to democratize data access for non-technical users, and they form a foundational capability for the emerging class of *AI Data Scientists*—intelligent systems that automate or assist with end-to-end data workflows over tabular and time-series data [6]. In industrial settings, such as monitoring electricity consumption anomalies across manufacturing sectors, queries must often be answered on-premise due to data sensitivity, ruling out cloud-based LLM APIs. Small language models (SLMs, ≤4B parameters) offer a viable alternative, but their base SQL generation accuracy is low: in our setting, a 4B model achieves only 38% format compliance out of the box.

Prior work addresses this through iterative self-correction: if the generated SQL is invalid, feed the error back and retry [1, 2, 23]. These systems consistently demonstrate that *any* feedback improves over no feedback. However, a fundamental question remains unanswered: **does the diagnostic richness of feedback matter, or is the mere signal of failure sufficient?**

This question has practical consequences. Enterprise SQL deployments face a design trade-off between feedback pipeline complexity and inference cost. If richer feedback yields higher accuracy, the engineering effort is justified. If it only accelerates convergence to the same ceiling, the decision reduces to a compute budget allocation.

We investigate this question through a controlled study on a fine-tuned Gemma-3 4B model generating SQL over Korean industrial electricity data (3M records, 8 sectors). We combine LoRA fine-tuning with EXPLAIN-based iterative correction at three levels of feedback granularity: generic failure signal, database error message, and structured EXPLAIN output with schema hints.

Our main finding is that **feedback granularity controls convergence speed, not accuracy ceiling**. All three feedback levels reach statistically indistinguishable final accuracy (86–88%, McNemar $p \geq 0.75$), but structured feedback requires 20% fewer inference calls to converge (71 vs. 89). In our setting, this suggests feedback design is primarily a test-time compute efficiency question [25] rather than an accuracy question.

Our contributions are:

(1) A **three-level feedback ablation** showing that granularity affects convergence speed but not accuracy ceiling—to our knowledge, the first such comparison in text-to-SQL self-correction.
(2) An **EXPLAIN-based correction pipeline** achieving 90%/77% (ID/OOT) FCR with a 4B SLM on industrial data, suitable for on-premise AI Data Scientist deployment.
(3) An **ablation** demonstrating that a learned 270M critic cannot match deterministic EXPLAIN validation, even with clean training data.
(4) **Cross-model validation** on Qwen-2.5 3B showing identical OOT convergence after fine-tuning.

2 Related Work

LLM self-correction. Self-Refine [17] and Reflexion [24] demonstrate that LLMs can improve outputs through iterative feedback loops. However, Huang et al. [9] show that LLMs cannot self-correct reasoning without external signals—a finding corroborated by two recent surveys [11, 19]. Self-Debug [2] uses execution results as external feedback for code generation, and CRITIC [7] incorporates tool interaction. Kumar et al. [12] train self-correction via reinforcement learning but target math and code, not SQL. Crucially, all prior work compares feedback *presence* vs. *absence*; none systematically varies feedback *granularity* to measure its effect on convergence.

Text-to-SQL. The field has advanced from supervised models on Spider [29] and BIRD [14] to enterprise-scale evaluation with Spider 2.0 [13] and LLM-based approaches including DIN-SQL [21] and DAIL-SQL [5]. Recent systems incorporate retry mechanisms: RetrySQL [23] trains on retry trajectories with corrupted SQL, MAGIC [1] generates self-correction guidelines from error patterns, and SHARE [22] uses hierarchical SLM-based correction. CHASE-SQL [20] uses execution plans for generation-time reasoning, whereas we use EXPLAIN output as *post-hoc correction feedback*—a complementary approach. Concurrent work models SQL errors via AST-based structural representations [27], whereas our EXPLAIN-based approach requires no external parser and leverages the database engine directly. None of these systems ablate feedback granularity.

SLM fine-tuning and deployment. LoRA [8] and QLoRA [3] enable parameter-efficient fine-tuning of small models. Synthetic training data improves text-to-SQL fine-tuning [28], and practical deployment systems such as SQLGenie [6] demonstrate the growing demand for production-grade SQL interfaces—a defining workload for AI Data Scientist platforms. We apply LoRA to a 4B model for domain-specific SQL generation, targeting on-premise deployment where API-based LLMs are impractical.

Learned verifiers. Step-level verification [15], execution-based reranking [18], and process reward models [26] have improved LLM outputs in math and code domains. Our critic ablation (§4.6) tests whether a learned verifier can substitute for deterministic SQL validation, finding that a 270M model lacks sufficient capacity for reliable SQL discrimination.

3 Method

3.1 Task Definition

Given a natural language query q about industrial electricity consumption patterns, the task is to generate a syntactically valid SQL query s over a single-table SQLite database. The target schema consists of one table (`greenbutton_data`) with 27 columns: a categorical industry identifier (`industry_en`), temporal fields (`year`, `month`, `day`), and 24 hourly consumption values (`elcp_use_0000` through `elcp_use_2300`) in kWh. Queries range from simple filtering ("Find days where Metal sector usage at 06:00 exceeds 10,000 kWh") to complex aggregations involving cross-industry comparisons, year-over-year changes, and statistical operations.

3.2 Fine-tuning

We generate 2,400 synthetic question–SQL pairs using template-based generation across 6 query families: threshold, z-score, pattern detection, comparative, temporal spike, and aggregate. For each sample, we randomly select an industry sector, year, hour, and relevant parameters. An example template instantiation for the threshold family:

> *"Find days where Metal power usage at 14:00 exceeds 2.0 times the average in 2021."*

Each generated SQL is validated by executing it against the database; pairs that fail are discarded and regenerated. The prompt format follows a structured template:

```
[DB Schema]
Table: greenbutton_data
Columns: industry_en (TEXT), year (INT),
month (INT), day (INT),
elcp_use_0000~elcp_use_2300 (FLOAT)

Instruction: {query}
Generate a valid SQL query...
```

We fine-tune Gemma-3 4B using LoRA [8] with rank r=16, scaling factor α=32, and dropout 0.05. Training uses a learning rate of 2×10^{-4} with cosine schedule, batch size 4 with gradient accumulation 4, for 3 epochs.

We define two evaluation splits: **in-distribution (ID)** queries (n=100) use the same 6 template families seen during training, while **out-of-template (OOT)** queries (n=100) use 10 held-out families (e.g., variance analysis, consecutive pattern detection, range difference) that share the same schema but require unseen SQL structures.

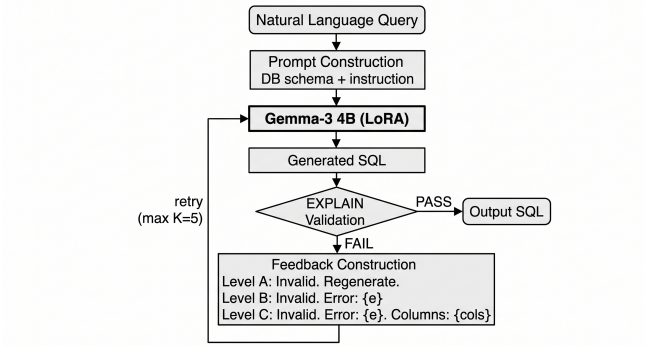

Figure 1: System architecture. The model generates SQL, which is validated via EXPLAIN. On failure, feedback at one of three granularity levels is appended to the prompt for re-generation (max K=5 retries).

3.3 EXPLAIN-Based Iterative Correction

After the model generates a candidate SQL s_i, we validate it by executing `EXPLAIN` s_i against the target database. If EXPLAIN returns an error, we construct a feedback prompt and request a new generation s_{i+1}. This loop continues for up to K=5 retries or until EXPLAIN succeeds.

We compare three feedback strategies of increasing diagnostic specificity. The exact feedback strings appended to the retry prompt are:

- **Level A** (generic): "`The SQL is invalid. Regenerate.`"
- **Level B** (error message): "`The SQL is invalid. Error: {e}`" where `{e}` is the verbatim error string from EXPLAIN (e.g., "`no such column: hour`").
- **Level C** (structured EXPLAIN): "`The SQL is invalid. Error: {e}. Available columns: {cols}`" where `{cols}` lists the valid column names from the target table schema.

This three-level design isolates the effect of feedback informativeness: all conditions provide the same *signal* (invalid SQL), but differ in the *diagnostic content* available for correction.

4 Experiments

4.1 Setup

Dataset. We use the Korea Electric Power Corporation (KEPCO) Green Button dataset, which records hourly electricity consumption across eight industrial sectors (Metal, Chemical, Food, Energy, Ceramics, Paper&Wood, Textile, and Industry_Etc) from 2020 to 2022, totaling 3,046,368 records. The data resides in a single SQLite table (`greenbutton_data`) with 27 columns.

Evaluation queries. We construct 100 in-distribution (ID) queries drawn from the 6 training template families, and 100 out-of-template (OOT) queries spanning 10 unseen categories: absolute threshold, complex aggregation, consecutive pattern, cross-industry comparison, multi-condition, percentile, range, top-n, variance, and year-over-year. Each OOT category contains 10 queries. OOT queries test generalization to query structures not seen during fine-tuning.

Table 1: Format Compliance Rate across configurations (n=100 per split, max 5 retries). Wilson 95% CIs reported. Feedback granularity ablation is detailed in Table 2; critic configurations are discussed in §4.6.

Configuration	ID	95% CI	OOT	95% CI
Base (no FT)	.38	[.29, .48]	.34	[.25, .44]
FT only	.80	[.71, .87]	.69	[.59, .77]
FT + Critic (noisy)	.33	[.25, .43]	.27	[.19, .36]
FT + Critic (clean)	.82	[.73, .88]	.65	[.55, .74]
FT + EXPLAIN retry	.90	[.83, .94]	.77	[.68, .84]

Models. Our primary generator is Gemma-3 4B, fine-tuned with LoRA (r=16, α=32) on 2,400 synthetic question–SQL pairs. For the critic ablation (§4.6), we train Gemma-3 270M as a binary classifier under two data conditions. We additionally validate on Qwen-2.5 3B (§4.7) with identical training data and LoRA configuration. All experiments run on a single NVIDIA RTX 5090 with Flash Attention 2.

Metrics. We report **Format Compliance Rate (FCR)**: the proportion of generated SQL queries that pass syntactic validation via SQLite's `EXPLAIN` command. We choose FCR over execution accuracy because our industrial setting lacks gold-standard SQL annotations; FCR provides a fully automatable, objective proxy for syntactic correctness. We compute 95% Wilson confidence intervals for all FCR values and apply McNemar's exact test [4] for pairwise comparisons between feedback conditions. We additionally report execution success rate (proportion of FCR-valid queries that execute without error or timeout).

4.2 System Recipe Results (C1)

Table 1 presents FCR across five configurations. Fine-tuning alone yields the largest single improvement, raising ID FCR from .38 to .80 (+42%p) and OOT FCR from .34 to .69 (+35%p). Adding EXPLAIN-based retry provides a further +10%p on ID and +8%p on OOT, reaching .90 and .77 respectively. Configurations 3 and 4 (critic-based verification) are discussed in §4.6.

The ID–OOT gap persists across all configurations: 4%p at base, 11%p after fine-tuning, and 13%p after retry. This widening gap likely reflects that OOT residual failures involve qualitatively different error types (unsupported functions, compositional complexity) that retry cannot address, as confirmed by the error taxonomy in §4.5.

4.3 Feedback Granularity and Convergence (C2)

Our central experiment examines whether the *content* of correction feedback affects self-correction outcomes. We apply the three feedback levels to OOT queries (n=100, max 5 retries), starting from the FT-only baseline (.69).

Accuracy ceiling is feedback-invariant. Table 2 shows that all three conditions reach statistically indistinguishable final FCR: .86, .88, and .87 for A, B, and C respectively. McNemar's exact test yields $p \geq 0.75$ for all pairwise comparisons (A vs. B: p=0.75, n_{disc}=10;

Table 2: Feedback granularity ablation on OOT queries (n=100, max 5 retries). McNemar's exact test: $p \geq 0.75$ for all pairwise comparisons. Calls = total retry calls across 100 queries. Avg Att. = mean attempts per query.

Level	FCR	95% CI	Avg Att.	Calls
No retry	.69	[.59, .77]	1.00	0
A (generic)	.86	[.78, .91]	1.89	89
B (error msg)	.88	[.80, .93]	1.79	79
C (EXPLAIN)	.87	[.79, .92]	1.71	71

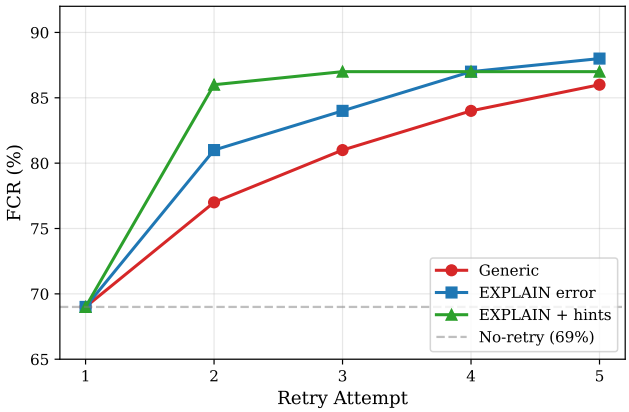

Figure 2: FCR convergence curves by feedback granularity (OOT, n=100). All conditions start at .69 (FT-only baseline). Level C reaches .87 by retry 2; Level A requires 4 retries to reach .86.

A vs. C: p=1.0, n_{disc}=9; B vs. C: p=1.0, n_{disc}=9), confirming that feedback granularity does not raise the accuracy ceiling.

Convergence speed is feedback-dependent. The conditions differ markedly in how quickly they reach this ceiling. Figure 2 shows the per-attempt progression. Level C reaches .86 after just one retry and plateaus at .87 by retry 2. Level B reaches .81 at retry 1 and .87 by retry 3. Level A climbs gradually: .77, .81, .84, .86 across four retries. In total, Level C requires 71 retry calls versus 89 for Level A—a 20% reduction in inference cost for equivalent final accuracy.

We note that Level B (error message) achieves the highest point estimate (.88) despite being simpler than Level C (.87), reinforcing that structured feedback's advantage is limited to convergence speed, not final accuracy. This finding frames feedback quality as a *compute efficiency* factor rather than an accuracy factor, connecting to recent work on scaling test-time compute [25].

4.4 Per-Category OOT Analysis

The aggregate OOT numbers mask variation across query categories. Table 3 breaks down FCR by the 10 OOT query families (n=10 each). Given the small per-category sample size, we highlight three tentative observations that require larger-scale confirmation:

Categories solved by fine-tuning. Consecutive pattern, cross-industry comparison, and percentile queries reach 1.0 FCR after fine-tuning alone, leaving no room for retry improvement. These categories require SQL patterns (e.g., `LAG()`, cross-table comparisons) that, while absent from training templates, are structurally close to training examples.

Table 3: Per-category OOT FCR (n=10 per category; results are illustrative given wide confidence intervals at this sample size). EXPLAIN retry provides the largest gains for range (+.4) and multi-condition (+.2) queries, but cannot help categories limited by unsupported SQL functions (variance) or compositional complexity (complex aggregation).

OOT Category	Base	FT	+EXPLAIN
Absolute threshold	.8	.8	.9
Complex aggregation	.1	.4	.4
Consecutive pattern	.4	1.0	1.0
Cross-industry	.6	1.0	1.0
Multi-condition	.5	.7	.9
Percentile	.5	1.0	1.0
Range difference	.0	.4	.8
Top-n	.3	.7	.7
Variance	.0	.3	.3
Year-over-year	.2	.6	.7
Overall	**.34**	**.69**	**.77**

Categories where EXPLAIN retry helps. Range difference and multi-condition queries show the largest retry gains (+40%p and +20%p respectively). These categories generate SQL with column name errors that EXPLAIN feedback directly addresses: the error message identifies the invalid column, and Level C additionally provides the correct column list.

Categories resistant to correction. Complex aggregation (.4) and variance (.3) queries remain low even after retry. Variance queries frequently generate `VARIANCE()`, which SQLite does not support natively—EXPLAIN catches these as FCR failures, and no amount of retry feedback can teach the model an alternative implementation. Complex aggregation queries require multi-step CTEs with nested subqueries that exceed the model's compositional capacity at this scale.

4.5 Error Taxonomy (C3)

To understand what each pipeline stage fixes and what it leaves unsolved, we classify all OOT failures into mutually exclusive error types (Figure 3).

Base model failures. (n=66) are dominated by column hallucination (48/66, 73%). The most frequent subtype is *elcp_variant* hallucination (36 cases), where the model generates schema-similar but non-existent column names (e.g., `elcp_0500` instead of `elcp_use_0500`). Markdown fence errors (9/66) indicate the model outputs code-fenced text rather than raw SQL.

Fine-tuning. eliminates markdown fence errors entirely (9→0) and removes all elcp_variant hallucinations (36→0). The residual failures (n=31) shift to *invented* column hallucination (20/31, 65%), where the model generates entirely fictional column names with no schema resemblance. Additionally, 5 cases involve unsupported

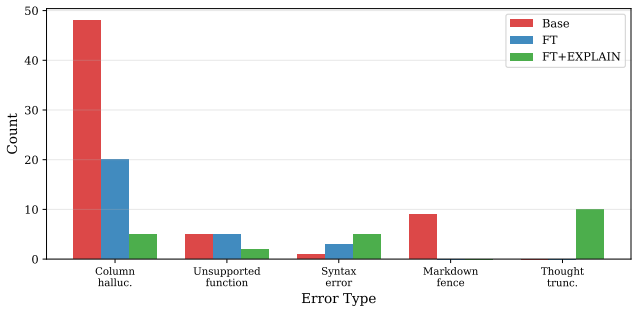

Figure 3: OOT failure distribution by error type across three configurations. FT eliminates markdown fence and elcp_variant errors; EXPLAIN retry reduces invented hallucination but introduces thought truncation as the primary residual failure mode.

SQLite functions (e.g., `VARIANCE()`) and 3 involve incomplete SQL where complex OOT queries caused the model to exceed the generation token budget.

EXPLAIN retry. further reduces column hallucination from 20 to 5 cases (those where the model persistently regenerates the same invalid column despite feedback). However, it introduces a new dominant failure mode: *thought truncation* (10/23, 43% of residual failures), where the growing prompt across retries exceeds the generation budget, producing incomplete SQL. This represents a structural limitation of iterative correction with fixed generation length.

4.6 Critic Ablation (C4)

Learned verifiers have shown promise in mathematical reasoning [15] and code generation [18]. We investigate whether a similar approach can substitute for deterministic `EXPLAIN` validation in SQL.

Training data and label noise. We initially collected 14,400 SQL samples with binary labels (PASS/FAIL) from the planner's outputs. A post-hoc audit via EXPLAIN validation revealed that 46.3% of FAIL-labeled samples were actually valid SQL—they had been mislabeled due to formatting issues in the original evaluation pipeline. The false-PASS rate was 0%. We constructed a clean dataset (11,059 samples) by relabeling all samples with EXPLAIN and trained a second critic on this corrected data.

Results. Table 4 evaluates both critics on a leak-free test set (n=300, 150 PASS + 150 FAIL, 0% overlap with training). The noisy critic performs at chance level (faithfulness .43, noise rejection .52). The clean critic achieves perfect faithfulness (1.00) but a noise rejection rate of only .20—it passes 80% of invalid SQL as valid.

This asymmetric failure is informative: the 270M model learns to recognize valid SQL patterns (high faithfulness) but cannot discriminate invalid SQL from valid SQL with sufficient resolution (low noise rejection). This appears to be a capacity limitation at this scale; whether larger learned verifiers (e.g., 1B+) could compete with EXPLAIN remains an open question.

Table 4: Verifier comparison on leak-free test set (n=300). Faithfulness: fraction of valid SQL correctly passed. Noise rejection: fraction of invalid SQL correctly rejected. Even with clean training data, the 270M critic passes 80% of invalid SQL.

Critic	Faithfulness	Noise Rej.
Noisy (14.4k, 46% label err.)	.43 [.36, .51]	.52 [.44, .60]
Clean (11.1k, 0% label err.)	1.00 [.97, 1.0]	.20 [.14, .27]
EXPLAIN (deterministic)	1.00	1.00

Table 5: Cross-model validation. Despite a large base performance gap, both models converge to identical OOT FCR (.69) after fine-tuning on the same 2,400 pairs, suggesting a data-constrained ceiling.

	Base		FT	
Model	ID	OOT	ID	OOT
Gemma-3 4B	.38	.34	.80	.69
Qwen-2.5 3B	.10	.06	.77	.69

By contrast, `EXPLAIN` deterministically catches all syntactic errors and invalid column references with zero false negatives, at no training cost. This result motivates our design choice: a deterministic verifier outperforms a learned one for this task, and the engineering effort of training a critic is better spent on improving the generator.

The effect on downstream performance is visible in Table 1: FT + Critic (noisy) degrades FCR below the FT-only baseline (.33/.27 vs. .80/.69) because the noisy critic rejects valid SQL. FT + Critic (clean) achieves .82/.65, comparable to FT-only, because it passes most queries through without meaningful filtering.

4.7 Cross-Model Validation (C5)

To assess whether our findings are specific to Gemma, we repeat the fine-tuning pipeline on Qwen-2.5 3B with identical training data and LoRA configuration (Table 5).

Despite a large gap in base performance (Gemma .38/.34 vs. Qwen .10/.06 on ID/OOT), both models converge to the same OOT FCR of .69 after fine-tuning. Gemma retains a modest ID advantage (.80 vs. .77). The shared OOT convergence point suggests that the generalization ceiling is constrained by the training data (2,400 pairs over a single schema) rather than by model architecture at this scale. We note that two models are insufficient to establish a general claim; this result is suggestive rather than conclusive.

4.8 Semantic Validation

FCR measures syntactic validity, not whether the SQL returns meaningful results. We execute all FCR-valid queries against the database with a 30-second timeout to assess execution success (Table 6).

The FT-only ID configuration has the lowest execution rate (79%) due to 17 timeouts from z-score queries. While `VARIANCE()` calls are caught by EXPLAIN at the FCR stage (accounting for 16 of 20 FT ID

Table 6: Execution success rates for FCR-valid queries (30s timeout). FT ID timeouts are from z-score queries computing standard deviation via nested subqueries over the full 3M-record table.

Config	Valid	Executed	Exec %	Timeouts
Base ID	38	37	97%	1
Base OOT	34	32	94%	2
FT ID	80	63	79%	17
FT OOT	69	68	99%	1
FT+EXPLAIN ID	90	73	81%	17
FT+EXPLAIN OOT	77	71	92%	6

FCR failures), the FCR-valid z-score queries compute standard deviation via nested subqueries over the full 3M-record table, causing execution timeouts. OOT execution rates are consistently higher (92–99%) because OOT queries less frequently require expensive statistical computations. The FT+EXPLAIN configuration shows 92% OOT execution (6 timeouts from complex window function queries).

Execution success indicates the query runs without error but does not guarantee semantic correctness—a query that returns an empty result set technically "succeeds." Full semantic evaluation via human annotation remains future work.

5 Discussion

Feedback as compute allocation. Our central finding—that feedback granularity controls convergence speed but not accuracy ceiling—has direct implications for AI Data Scientist system design. In latency-constrained deployments (e.g., on-premise industrial monitoring), structured EXPLAIN feedback reduces the expected number of inference calls by 20%, translating to proportional savings in latency and compute cost. Conversely, when compute is cheap but engineering effort is scarce, even a generic "retry" signal achieves comparable accuracy, simplifying the feedback pipeline.

This connects to the test-time compute scaling framework of Snell et al. [25]: our results suggest that for SLM self-correction, the *quality* of test-time computation (via richer feedback) matters alongside its *quantity* (number of retries).

Where EXPLAIN retry helps—and where it does not. The per-category analysis (Table 3) reveals that EXPLAIN retry is most effective for *column-level errors* in structurally simple queries (range: +.4, multi-condition: +.2). The EXPLAIN error message directly identifies the problematic column, and Level C provides the valid alternatives. For *function-level errors* (variance: .3 unchanged) and *structural complexity* (complex aggregation: .4 unchanged), EXPLAIN feedback is insufficient because the error is semantic rather than syntactic.

Thought truncation as a test-time compute ceiling. The error taxonomy reveals that EXPLAIN retry *creates* failures even as it fixes others: thought truncation accounts for 43% of residual failures (10/23). Each retry appends the previous SQL and feedback to the prompt, consuming generation budget. After 3–4 retries, the accumulated context leaves insufficient tokens for a complete SQL statement. This represents a fundamental tension in iterative correction: more retries increase the probability of success but also increase the probability of truncation. Longer context windows or summarization-based feedback compression could mitigate this trade-off.

Distinction from CHASE-SQL. Pourreza et al. [20] also leverage execution plans, but as a *generation-time reasoning strategy*—the model conditions on EXPLAIN output when producing its initial SQL. In contrast, we use EXPLAIN output as *post-hoc diagnostic feedback* within an iterative correction loop. These are complementary: CHASE-SQL's approach could be combined with ours to improve both initial generation and subsequent correction.

Label noise and verifier training. The 46.3% false-FAIL rate in our initial critic training data (discovered via EXPLAIN audit) highlights a broader issue for AI Data Scientist evaluation pipelines: SQL validity labels derived from execution-based evaluation are unreliable. Formatting artifacts, timeout limits, and evaluation pipeline bugs can mislabel valid SQL as invalid. This finding suggests that text-to-SQL systems that rely on execution-based training signals [23] may benefit from EXPLAIN-based label verification as a preprocessing step.

Limitations

Scale. All experiments use n=100 queries per condition (10 per OOT category), yielding wide confidence intervals (typically ±8–10%p). The feedback granularity differences (86–88%) fall within overlapping CIs, and while McNemar's test confirms no significant difference, a larger sample might reveal small effects our study is underpowered to detect.

Schema diversity. We evaluate on a single table with a fixed schema. Real-world text-to-SQL requires multi-table joins, schema linking, and diverse column types. Our results do not generalize to these settings without further validation.

FCR vs. semantic correctness. FCR measures syntactic validity, not whether the SQL returns the intended result. Our execution analysis (§4.8) shows that 79–99% of FCR-valid queries execute successfully, but execution success does not guarantee semantic correctness. Manual evaluation of output semantics remains future work.

Benchmark coverage. We do not evaluate on standard benchmarks (Spider, BIRD). While recent work questions the ecological validity of such benchmarks [10, 16], their absence limits direct comparison with prior systems. Moreover, systems evaluated on Spider/BIRD use execution accuracy over multi-table schemas with gold SQL, making direct numerical comparison with our single-table FCR-based setting uninformative.

Model coverage. Cross-model validation uses only two model families (Gemma, Qwen). The shared OOT convergence point is suggestive but not generalizable from two data points.

Ethics Statement

The KEPCO Green Button dataset used in this study contains aggregate industrial electricity consumption statistics with no personally

identifiable information. The data is publicly available through the Korea Electric Power Corporation's open data portal. Our synthetic evaluation queries do not target individual facilities or employees. All experiments run locally on a single GPU with no external API calls, consistent with the on-premise deployment scenario motivating this work.

6 Conclusion

We presented an empirical study of EXPLAIN-based iterative correction for SQL generation with a fine-tuned 4B SLM on industrial time-series data—a representative workload for AI Data Scientists deployed under data-sensitivity constraints. Our system achieves 90% ID and 77% OOT format compliance through the combination of LoRA fine-tuning and structured retry.

Our main finding is that feedback granularity controls convergence efficiency rather than accuracy ceiling: generic, error-only, and structured EXPLAIN feedback all reach equivalent final accuracy (86–88%, $p \geq 0.75$), but structured feedback requires 20% fewer inference calls. In our setting, this suggests feedback design functions as a compute optimization problem rather than an accuracy problem.

Per-category analysis reveals that EXPLAIN retry is most effective for column-level errors in simple queries (range: +40%p) but cannot address function-level gaps (variance) or compositional complexity (complex aggregation). The dominant residual failure mode—thought truncation from accumulated retry context—points to context management as the key bottleneck for iterative correction with SLMs.

Additionally, we show that a learned 270M critic fails to match the reliability of deterministic EXPLAIN validation, even with clean training data, and that fine-tuning gains replicate across model families with a shared OOT convergence point. These results suggest directions for SLM-based AI Data Scientist platforms that warrant further validation on more complex schemas: invest in fine-tuning data quality, prefer deterministic validation over learned critics at this model scale, and consider feedback granularity as a compute budget decision rather than an accuracy decision.

References

[1] Arian Askari, Christian Poelitz, and Xinye Tang. 2025. MAGIC: Generating Self-Correction Guideline for In-Context Text-to-SQL. In *Proceedings of AAAI*. https://ojs.aaai.org/index.php/AAAI/article/view/34511

[2] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching Large Language Models to Self-Debug. In *Proceedings of ICLR*. https://openreview.net/forum?id=KuPixIqPiq

[3] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. In *Proceedings of NeurIPS*. https://papers.nips.cc/paper_files/paper/2023/hash/1feb87871436031bdc0f2beaa62a049b-Abstract-Conference.html

[4] Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. 2018. The Hitchhiker's Guide to Testing Statistical Significance in Natural Language Processing. In *Proceedings of ACL*. https://aclanthology.org/P18-1128/

[5] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proceedings of the VLDB Endowment* 17, 5 (2024), 1132–1145. https://www.vldb.org/pvldb/vol17/p1132-gao.pdf

[6] Pushpendu Ghosh, Aryan Jain, and Promod Yenigalla. 2025. SQLGenie: A Practical LLM based System for Reliable and Efficient SQL Generation. In *Proceedings of ACL Industry*. https://aclanthology.org/2025.acl-industry.71/

[7] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing. In *Proceedings of ICLR*. https://openreview.net/forum?id=Sx038qxjek

[8] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *Proceedings of ICLR*. https://openreview.net/forum?id=nZeVKeeFYf9

[9] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. In *Proceedings of ICLR*. https://openreview.net/forum?id=IkmD3fKBPQ

[10] Tengjun Jin, Yoojin Choi, Yuxuan Zhu, and Daniel Kang. 2026. Text-to-SQL Benchmarks are Broken: An In-Depth Analysis of Annotation Errors. In *Proceedings of CIDR*. https://www.cidrdb.org/cidr2026/papers/p5-jin.pdf

[11] Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. When Can LLMs Actually Correct Their Own Mistakes? A Critical Survey of Self-Correction of LLMs. *Transactions of the ACL* 12 (2024), 1417–1440. https://aclanthology.org/2024.tacl-1.78/

[12] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. 2025. Training Language Models to Self-Correct via Reinforcement Learning. In *Proceedings of ICLR*. https://openreview.net/forum?id=CjwERcAU7w

[13] Fangyu Lei, Jixuan Chen, Yuxiao Ye, et al. 2025. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. In *Proceedings of ICLR*. https://openreview.net/forum?id=XmProj9cPs

[14] 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. In *Proceedings of NeurIPS*. https://papers.nips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html

[15] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's Verify Step by Step. In *Proceedings of ICLR*. https://openreview.net/forum?id=v8L0pN6EOi

[16] Xinyu Liu, Shuyu Shen, Boyan Li, Nan Tang, and Yuyu Luo. 2025. NL2SQL-BUGs: A Benchmark for Detecting Semantic Errors in NL2SQL Translation. In *Proceedings of KDD*. https://doi.org/10.1145/3711896.3737427

[17] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In *Proceedings of NeurIPS*. https://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html

[18] Ansong Ni, Srini Iyer, Dragomir Radev, Ves Stoyanov, Wen-Tau Yih, Sida I. Wang, and Xi Victoria Lin. 2023. LEVER: Learning to Verify Language-to-Code Generation with Execution. In *Proceedings of ICML*. https://openreview.net/forum?id=Gj3zN9zs4v

[19] Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2024. Automatically Correcting Large Language Models: Surveying the Landscape of Diverse Automated Correction Strategies. *Transactions of the ACL* 12 (2024), 484–506. https://aclanthology.org/2024.tacl-1.27/

[20] Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O. Arik. 2025. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. In *Proceedings of ICLR*. https://openreview.net/forum?id=CvGqMD5OtX

[21] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Proceedings of NeurIPS*. https://papers.nips.cc/paper_files/paper/2023/hash/72223cc66f63ca1aa59edaec1b3670e6-Abstract-Conference.html

[22] Ge Qu, Jinyang Li, Bowen Qin, Xiaolong Li, Nan Huo, Chenhao Ma, and Reynold Cheng. 2025. SHARE: An SLM-based Hierarchical Action CorREction Assistant for Text-to-SQL. In *Proceedings of ACL*. https://aclanthology.org/2025.acl-long.552/

[23] Alicja Rączkowska, Riccardo Belluzzo, Piotr Zieliński, et al. 2026. RetrySQL: Text-to-SQL Training with Retry Data for Self-Correcting Query Generation. In *Proceedings of AAAI*. https://ojs.aaai.org/index.php/AAAI/article/view/40556

[24] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Proceedings of NeurIPS*. https://openreview.net/forum?id=vAElhFcKW6

[25] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning. In *Proceedings of ICLR*. https://openreview.net/forum?id=4FWAwZtd2n

[26] Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations. In *Proceedings of ACL*. https://aclanthology.org/2024.acl-long.510/

[27] Weinan Wang, Yuren Li, Lei Wang, and Bing Qin. 2026. ErrorLLM: Leveraging Error Structure in SQL for Self-Correcting Large Language Models. *arXiv preprint arXiv:2603.03742* (2026). https://arxiv.org/abs/2603.03742

[28] Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing Text-to-SQL Data from Weak and Strong LLMs. In *Proceedings of ACL*. https://aclanthology.org/2024.acl-long.425/

[29] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of EMNLP*. https://aclanthology.org/D18-1425/

A Dataset Details

Schema. The `greenbutton_data` table contains 3,046,368 records with the following columns: `industry_en` (categorical, 8 values), `year` (2020–2022), `month`, `day`, and `elcp_use_0000` through `elcp_use_2300` (24 hourly electricity consumption values in kWh).

Industry distribution. Metal (38.7%), Chemical (21.3%), Industry_Etc (10.5%), Food (10.4%), Energy (6.3%), Ceramics (6.2%), Paper_Wood (3.8%), Textile (2.8%).

OOT categories. The 10 out-of-template query families are: absolute threshold, complex group aggregation, consecutive pattern, cross-industry comparison, multi-condition filter, percentile, range difference, top-n, variance, and year-over-year. Each family contains 10 queries (n=100 total).

B Hyperparameters

Fine-tuning. LoRA rank r=16, scaling α=32, dropout 0.05. Learning rate 2×10^{-4} with cosine schedule. Batch size 4, gradient accumulation 4, 3 epochs. Training data: 2,400 pairs across 6 template families.

Generation. Max new tokens: 768. Greedy decoding (temperature 0, no sampling). Applied identically across all conditions.

Retry. Maximum K=5 retries per query (up to 6 total generation calls including the initial attempt). The feedback prompt appends the previous SQL and feedback to the original query. No system prompt modification between retries.

Critic. Gemma-3 270M, LoRA r=8, α=16. Binary classification (PASS/FAIL), single-token output constrained via logit masking. Noisy version: 14,400 training samples (46.3% false-FAIL). Clean version: 11,059 training samples (0% label error). Evaluation: 300 held-out samples (0% overlap).

C Retry Curve Details

Table 7 shows the per-attempt FCR progression for each feedback level, complementing Figure 2.

Table 7: Per-attempt FCR progression (OOT, n=100). Baseline (attempt 0) is .69 for all conditions. Level C converges in 1 retry; Level A requires 4.

Level	Att 1	Att 2	Att 3	Att 4	Att 5
A (generic)	.77	.81	.84	.86	.86
B (error)	.81	.84	.87	.88	.88
C (EXPLAIN)	.86	.87	.87	.87	.87