Failure Localization for Enterprise Text-to-SQL Agents

Lessons from a **Production-Representative** Financial Case Study

From generating SQL to ***diagnosing its failures***

KI2026

KI2026 AI4SE Workshop

KPMG

Heeyong Eun · Seungyeon Chung

Why Text-to-SQL Matters in Enterprise Data Use

Natural-language access **connects business questions to the data** already stored in enterprise databases.

Enterprise Data

Customer & Account Data
Profiles, accounts, statuses, segments

Payment Information
Payment methods, authorizations, refunds

Transaction Records
Orders, invoices, amounts, line items

Relational Database
Structured, trusted, enterprise data

Why it matters

Faster data access
Get answers quickly from trusted data

Less manual SQL writing
Empower more users with natural language

Better business decisions
Improve insights, actions and outcomes

Business questions
Ask in natural language, like you speak

SQL generation
Text-to-SQL turns intent into accurate queries

Faster data access
Get trusted answers in less time

Our goal is not another SOTA Text-to-SQL model

We focus on an enterprise framework for **controlled**, **inspectable**, and **correctable** generation.

Typical performance question

- How many correct SQL queries are generated?
- Does final SQL match the gold answer?
- What is the benchmark score?

Enterprise operational question

- How do we restrict wrong references and joins?
- Which decision went wrong first?
- How do we reuse administrator corrections?

Enterprise Framework = constrained generation + inspectable decisions + correction loop

- Constrain what can be generated
- Make decisions visible and reviewable
- Capture corrections and reuse them

Prior work improved generation, linking, and robustness

Yet an important challenge remains: pinpointing where the first wrong decision occurs.

Spider

Cross-domain Text-to-SQL benchmark

Generalization to unseen schemas

RAT-SQL

Relation-aware schema encoding

Question–table–column linking

Execution-guided decoding

Use execution signals

Reject invalid SQL candidates

LinkAlign & LLM testing

Large schema-pool selection

LLM apps as testable systems

Remaining gap

When the final SQL is valid yet wrong, existing work does not tell us **which stage** in the generation pipeline first went wrong.

Our research question

Can we **identify the first wrong decision** in a valid-but-wrong SQL and use it for **targeted correction** of the agent?

Inspection scope

Physical column

Table / root selection

Join

Filter

Industrial Case and Experimental Configurations

We evaluate a production-representative financial Text-to-SQL system and compare a baseline with a revised configuration that exposes inspectable intermediate decisions.

Industrial-scale evaluation in a production-representative environment

- **900** tables
- **22K** columns
- **24K** domain terms
- **3K-word** lexicon
- **69** business queries
- Administrator-approved SQL

These issues emerge in a production-representative environment and are difficult to see in small pilot setups.

Baseline

User request → Vector DB retrieval → Single agent → Final SQL

Direct generation

Revised Configuration

Structured request → Ontology & lexicon linking → Catalog check → Table & join planning → SQL planning / generation → Logged artifacts

Structured decisions · validation · traceability

Industrial case: production-representative financial environment

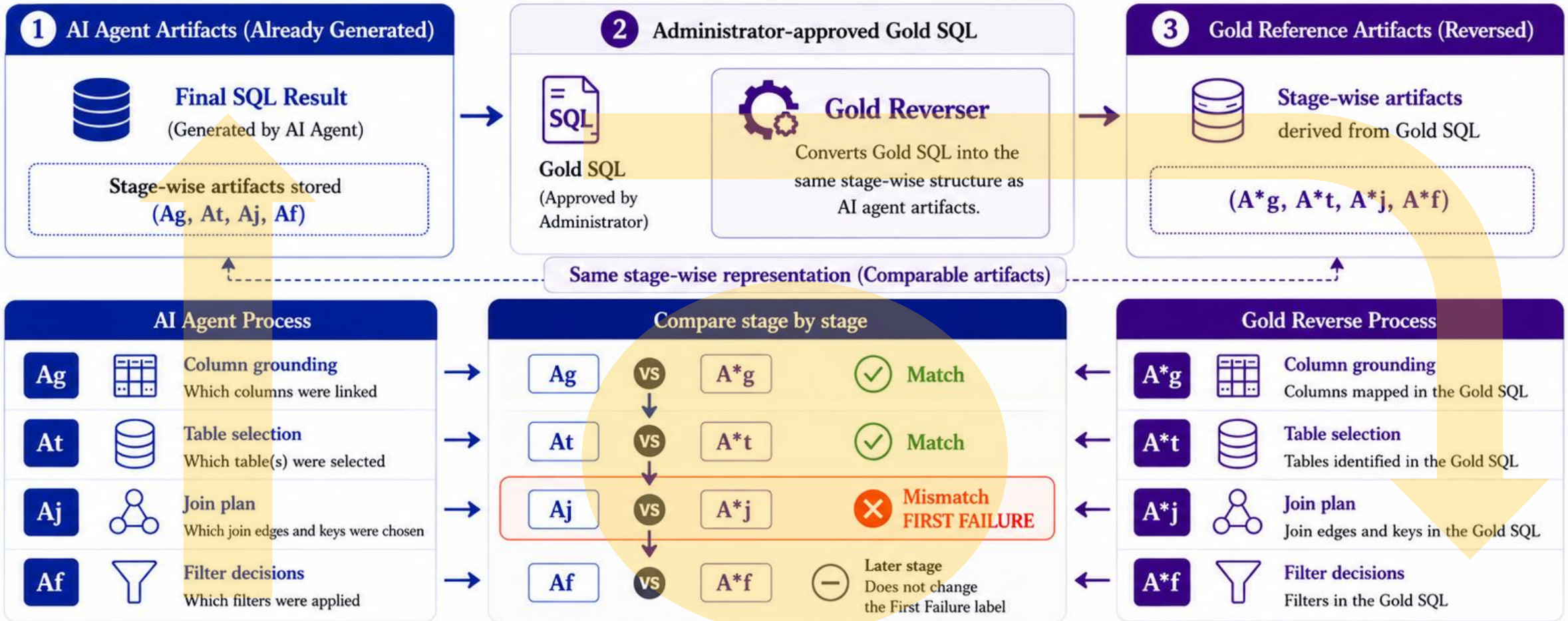
Real-world scale, real-world complexity, real-world issues.

The revised configuration exposes intermediate decisions for later inspection.

Enabling validation, debugging, and auditability.

Methodology: Gold Reverser and First-Failure Localization

Administrator-approved Gold SQL is reversed into the same stage-wise format as the agent artifacts.
Then we compare them stage by stage to locate the first failure.



Case 1 — First Failure at Grounding (Ag)

Example S01: the revised agent selects the correct table but misses two required columns.

User Query

"Retrieve authorized users, control department code, control button type code, control button authorization flag, and task-related information for the screen with ID {scrId}."

Revised SQL (fragment)

```
SELECT
    SCR_ID,
    CTRL_DEPT_COD,
    CTRL_BTN_TCD,
    CTRL_BTN_AUTH_YN,
    ...
FROM BS0001
WHERE SCR_ID = '{scrId}'
```

Missing: AUTH_USER_COD, AUTH_USER_NM

Gold SQL (fragment)

```
SELECT
    SCR_ID,
    AUTH_USER_COD,
    AUTH_USER_NM,
    CTRL_DEPT_COD,
    CTRL_BTN_TCD,
    CTRL_BTN_AUTH_YN,
    ...
FROM BS0001
WHERE SCR_ID = :scrId
```

Gold Reverser + stage-wise comparison

Compare Revised artifacts with Gold-derived reference artifacts.

Artifact-level comparison

Stage	Revised result vs Gold reference	Outcome
Ag	Revised columns omit AUTH_USER_COD and AUTH_USER_NM	Mismatch — FIRST FAILURE
At	BS0001 vs BS0001	Match
Aj	No join needed	Match
Af	SCR_ID filter present	Match

CLA = 0.889

TLA = 1.000

First Failure = Grounding (Ag)

The table is correct, but the grounding step misses two required columns.

Case 2 — First Failure at Table Selection (At)

Example S05: the revised agent grounds the columns correctly but selects the wrong table set.

User Query

"Retrieve the department code and department name of the employee whose employee number is {epNo}."

Revised SQL (fragment)

```
SELECT
    BC2016.DEPT_COD,
    FA2115.DEPT_NM

FROM BC2016
JOIN FA2115 ON
    BC2016.DEPT_COD =
    FA2115.DEPT_COD

WHERE BC2016.EP_NO = '{epNo}'
```

Gold SQL (fragment)

```
SELECT
    A.DEPT_COD,
    B.DEPT_NM

FROM BC2016 A,
     BC2013 B
WHERE A.EP_NO = :epNo
  AND A.DEPT_COD =
      B.DEPT_COD
```

Gold Reverser + stage-wise comparison

Compare Revised artifacts with Gold-derived reference artifacts.

Artifact-level comparison

Stage	Revised result vs Gold reference	Outcome
Ag	Both include DEPT_COD, DEPT_NM, EP_NO	Match
At	BC2016, FA2115 vs BC2016, BC2013	Mismatch — FIRST FAILURE
Aj	JOIN: FA2115 on DEPT_COD vs JOIN: BC2013	Later mismatch
Af	WHERE EP_NO = ? vs WHERE EP_NO = ? AND DEPT_COD = DEPT_COD	Later mismatch

CLA = 1.000

TLA = 0.500

First Failure = Table Selection (At)

The columns are correct, but the revised agent selects FA2115 instead of BC2013.

Paired Replay Evaluation

We evaluate the Baseline and Revised configurations by **paired replay** on the same frozen, production-representative metadata snapshot.

Study setting

- Financial production-representative environment
- 900 tables, 22,000 columns
- 24,000 domain terms, 3,000-word lexicon
- Metadata-focused scope (no customer or transaction row data)
- 69 anonymized business queries

This evaluation focuses on metadata, schema grounding, and SQL structure. End-to-end result accuracy is out of scope.

Evaluation protocol

1. 69 anonymized business queries
2. Same frozen metadata snapshot
3. Replay both systems: Baseline and Revised
4. Paired comparison under the same evidence resources

This design makes the comparison controlled and reproducible.

- One-month practice operation is analyzed separately for operational insights and is not included in CLA, TLA, or SER.

Metrics

CLA — Macro-F1
Over physical column names (exact match).

TLA — Macro-F1
Over physical table names (exact match).

SER — Structural Error Rate
Rate of parsing error, catalog violation, or invalid join structure

Higher CLA and TLA are better.
Lower SER is better.

Results across 69 queries

CLA — Macro-F1	TLA — Macro-F1	SER — Structural Error Rate
0.543 → 0.782	0.327 → 0.547	0.725 → 0.536

The revised configuration improves schema selection and reduces structural errors under the same controlled replay setting.

Discussion
What the Evaluation Reveals

The evaluation shows clear improvement, but also why human review and reusable references remain essential.

1 Human Review Is Still Needed for New Requests

New / unseen requests

- No pre-existing Gold SQL is available
- Human review is required before use

Practice operation (observed)

- Inefficient SQL (same result but costly)
- Missing implicit business rules e.g., `DEL_YN = 'N'` (logical delete filter)

Human review remains necessary for new and operationally sensitive requests to ensure correctness.

2 Gold Reverser Makes Corrections Reusable

Human review
DA/DBA reviews the generated SQL and corrects it.

Approved SQL (Gold)
The corrected SQL becomes the approved version.

Gold Reverser
Convert approved SQL to reference artifacts: **A*g / A*t / A*j / A*f**

Reusable reference & regression case
Add to the reference artifact store and regression corpus.

Human corrections become **reusable references** for future diagnosis and regression testing.

We do not eliminate human review — we make it focused and reusable.

Conclusion and limitations

Enterprise Text-to-SQL must **reduce errors** and make the remaining errors **correctable**.

What we showed

- Baseline → Revised Configuration improved schema selection and reduced structural errors
- Step artifacts make decisions inspectable
- Approved-SQL comparison localizes the first wrong decision

Why it matters

- Supports constrained, inspectable, correctable enterprise Text-to-SQL
- Turns administrator corrections into reusable evidence
- Enables targeted fixes and regression testing

Limitations

- One financial domain, 69 anonymized queries
- Metadata-centric evaluation, not a row-data execution study
- Whole-system Baseline vs. Revised Configuration comparison; component effects not isolated

Enterprise Text-to-SQL must reduce errors and make the remaining errors correctable.

Thank You

Questions and Discussion

Key Message

Localize the first wrong decision → **Review** with confidence → **Reuse** corrections for the future → **Improve** continuously

We turn human review into reusable knowledge
to build reliable and trustworthy Enterprise Text-to-SQL agents.

Contact
seungyeonchung@kr.kpmg.com
heeyonge@gmail.com

Affiliation
KPMG Korea

Source Code