

Targeted Token-Level Unlearning for Repairing Text-to-SQL Models

Jinyu Zhang¹, Ruiheng Liu², Yu Zhang^{1*}

¹Harbin Institute of Technology, Harbin, China

²Rocket Force University of Engineering, Xi'an, China

{jyzhang, rhliu, zhangyu}@ir.hit.edu.cn

Abstract

Text-to-SQL models may exhibit recurring error generation patterns that are reproducible under a fixed model and difficult to eliminate through continued training. Existing approaches either perform post-hoc correction at the output level or require access to the target domain’s training data for preference optimization, neither of which directly targets these reproducible local decision tendencies. To address these limitations, we propose **Targeted Token-level Unlearning for SQL (TTU-SQL)**, a machine-unlearning-based framework that uses a probe dataset to detect and correct systematic errors without requiring access to the model’s original training data. The framework comprises three components: (1) AST-based token-level divergence detection that precisely localizes erroneous decision points; (2) a targeted error-type extraction procedure that selects three specific types of forgettable errors based on AST divergence and logit-level behavior; and (3) Part-NPO, which applies negative preference optimization only to localized divergence tokens to suppress erroneous continuations while preserving useful SQL generation knowledge. Across the Spider and BIRD datasets and all evaluated model backbones, TTU-SQL yields consistent execution-accuracy gains over equal-budget training-based repair baselines.

1 Introduction

Large Language Models (LLMs) have achieved remarkable progress on the Text-to-SQL task. Supervised fine-tuning on large-scale Text-to-SQL corpora enables open-source models to acquire strong SQL generation capabilities (Li et al., 2025). However, a widely overlooked fact is that **models may learn not only correct SQL generation patterns during training, but also recurring erroneous**

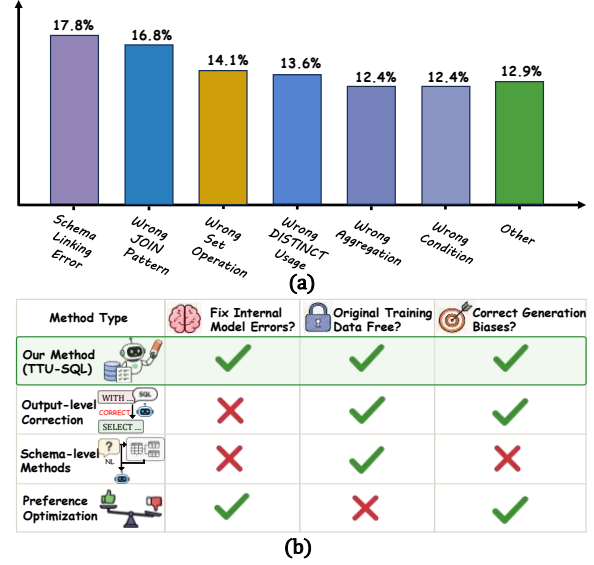


Figure 1: Motivation of TTU-SQL. (a) Distribution of systematic error patterns on Spider Dev using DeepSeek-R1-Distill-Qwen-7B. (b) Comparison between TTU-SQL and existing Text-to-SQL improvement paradigms.

generation tendencies (Chen et al., 2024; Qu et al., 2025). As shown in Figure 1(a), errors are concentrated in a small number of systematic categories rather than being uniformly distributed. This concentration suggests that many Text-to-SQL failures are structured and repeatable rather than isolated decoding noise, making them amenable to targeted unlearning.

Prior work has explored Text-to-SQL model errors from multiple angles. Error correction methods (Li et al., 2024c; Chen et al., 2024) operate at the output level without modifying model parameters. Schema linking methods (Song et al., 2025; Li et al., 2023a) focus on input comprehension rather than generation biases. Preference optimization (Rafailov et al., 2023) updates model behavior, but typically requires distribution-matched paired supervision and does not explicitly target recurring SQL error patterns. As summarized in Figure 1(b),

* Corresponding author

Code: <https://github.com/oakor/TTU-SQL>

existing paradigms either operate externally, require paired and distribution-matched supervision, or do not directly repair internal error generation tendencies. In contrast, **TTU-SQL aims to suppress systematic error patterns at the parameter level without requiring access to the model’s original training corpus.**

Through empirical investigation, we observe that some Text-to-SQL errors cannot be fully explained by missing task knowledge alone. Erroneous predictions often contain substantial correct SQL structure before diverging at a localized decision point. This suggests that useful partial generation behavior may coexist with locally dominant erroneous continuations. Such errors are difficult to repair through uniform likelihood maximization over gold SQL outputs, motivating a post-hoc repair strategy that directly targets recurring erroneous continuations while preserving useful SQL generation knowledge.

Based on this motivation, we propose **Targeted Token-level Unlearning for SQL (TTU-SQL)**, a post-training parameter-level repair framework for eliminating systematic Text-to-SQL error patterns. As illustrated in Figure 2, TTU-SQL follows a diagnose–locate–unlearn–recover procedure organized into three modules. First, the model is adapted to and diagnosed on a labeled probe dataset to expose recurring errors. Second, AST-based divergence detection localizes the first erroneous decision point, and logit-level analysis extracts specific forgettable error types. Third, Part-NPO suppresses the localized erroneous token, followed by recovery SFT to restore fluent and correct SQL generation. Our main contributions are as follows:

- We formulate systematic Text-to-SQL errors as localizable and forgettable generation tendencies, extending machine unlearning from instance-level data removal to post-hoc repair of recurring model errors.
- We propose TTU-SQL, a targeted token-level unlearning framework that combines AST-based divergence detection, extraction of three specific forgettable error types, and Part-NPO selective unlearning without requiring access to the model’s original training corpus.
- We evaluate TTU-SQL on Spider and BIRD across two model families, showing that it consistently improves over equal-budget training-based

repair baselines and remains complementary to inference-time enhancement methods.

2 Related Work

2.1 Text-to-SQL with Large Language Models

Text-to-SQL has long been studied as a structured semantic parsing problem. Early neural approaches generate SQL with sequence-to-sequence models or explicitly encode relations between question tokens and database schema elements (Zhong et al., 2018; Wang et al., 2020). Recent work has increasingly shifted toward large language models, improving Text-to-SQL performance through either inference-time enhancement or task-specific adaptation.

Inference-time methods improve the generation process without modifying model parameters. Schema-centric methods strengthen schema linking or schema grounding by pruning irrelevant schema elements, encoding schema relations, or resolving entity-value mismatches (Li et al., 2023a; Song et al., 2025). Prompting-based methods decompose Text-to-SQL generation into intermediate reasoning steps or select demonstrations to better elicit SQL generation ability from LLMs (Gao et al., 2024; Pourreza and Rafiei, 2023). Other approaches refine candidate SQL queries through self-debugging, post-hoc correction, or multi-candidate selection (Chen et al., 2024; Li et al., 2024a; Qu et al., 2025; Borchmann and Wydmuch, 2025; Tritto et al., 2025). These methods are effective at improving individual predictions, but they operate outside the model parameters and therefore do not directly remove recurring internal generation biases.

Training-stage methods improve the model itself by adapting LLMs to Text-to-SQL data. Large-scale supervised fine-tuning and data synthesis have enabled open-source models to acquire strong SQL generation capabilities (Li et al., 2025). However, additional fine-tuning can also reinforce recurring erroneous patterns together with correct SQL patterns. In contrast, TTU-SQL treats systematic Text-to-SQL failures as localizable error tendencies and repairs them through post-hoc parameter-level unlearning.

2.2 Preference Optimization for Model Adaptation

Preference optimization has become a standard paradigm for aligning language models with de-

sired behaviors. RLHF optimizes models using reward models learned from human preferences (Ouyang et al., 2022), while DPO directly optimizes pairwise preferences without explicit reward modeling (Rafailov et al., 2023). These methods are designed to increase the relative likelihood of preferred outputs over rejected ones.

In Text-to-SQL, a natural preference pair can be constructed from a gold SQL query and an erroneous model prediction. However, directly applying preference optimization couples two effects: increasing the probability of the gold SQL and decreasing the probability of the wrong prediction. This coupling can be undesirable when the probe data differ from the target evaluation distribution, since it may over-emphasize probe-specific SQL realizations. TTU-SQL instead decouples these two roles. It first uses negative preference optimization to suppress localized erroneous continuations, and then applies recovery SFT to restore general SQL generation ability. Thus, our goal is not general preference alignment, but targeted repair of recurring Text-to-SQL error tendencies.

2.3 Machine Unlearning

Machine unlearning aims to remove the influence of undesired data or knowledge from trained models. Exact unlearning methods remove training examples by retraining or partitioning the training process (Bourtoule et al., 2021), but such strategies are impractical for modern LLMs. Approximate unlearning methods reduce the cost through influence estimation, gradient ascent, or post-hoc parameter updates (Koh and Liang, 2017; Jang et al., 2023; Yao et al., 2024). Recent work further studies forgetting specific facts, private information, harmful knowledge, or designated examples (Maini et al., 2024; Li et al., 2024b).

NPO was proposed as an alignment-inspired unlearning objective that suppresses undesirable outputs while mitigating the collapse often caused by naive gradient ascent (Zhang et al., 2024). However, existing unlearning methods typically assume that the forgetting target is a known example, fact, or behavior to be removed. TTU-SQL targets a different object: systematic Text-to-SQL error patterns that emerge as recurring erroneous continuations across structurally similar SQL contexts. Instead of removing a known training instance, TTU-SQL uses a probe dataset to expose such patterns, localizes their divergence points with AST-based comparison, and applies token-level unlearn-

ing only to the erroneous decision positions. This extends machine unlearning from data removal to post-hoc repair of structured generation errors.

3 Preliminary Study

3.1 Problem Formulation

Setting. Let M_θ denote a trained Text-to-SQL model whose original training corpus is unknown or inaccessible. We assume access to a labeled probe dataset $D_P = \{(x_i, y_i^*)\}_{i=1}^N$, where x_i denotes a natural language question with database schema context and y_i^* denotes the corresponding gold SQL query. The probe dataset is used to expose, localize, and repair recurring error patterns. The target evaluation set D_T , such as Spider Dev or BIRD Dev, is used only for evaluation and is never accessed during training, error detection, or forget-set construction.

Objective. The goal is to obtain an updated model $M_{\theta'}$ that improves execution accuracy on D_T by selectively suppressing systematic error patterns detected on D_P . Unlike standard SFT, which uniformly increases the likelihood of gold outputs, our objective is to identify local erroneous decision points and reduce the model’s preference for the corresponding wrong continuations while preserving useful SQL generation knowledge.

3.2 Key Observations

Our preliminary analysis on the probe dataset reveals three observations that motivate targeted token-level unlearning. These observations are phrased as empirical regularities in our setting (Spider/BIRD with DeepSeek-R1-Distill backbones), not universal properties of Text-to-SQL models; the supporting statistics are reported in Appendix D.8.

Probe inference exposes recurring error tendencies. Model errors are not uniformly distributed across SQL structures. Instead, many failures recur around similar decision points, such as unnecessary aggregation clauses, incorrect column choices, or wrong structural operators. This concentration suggests that probe-set errors can serve as observable evidence of systematic generation tendencies rather than isolated prediction noise.

Some errors reflect suppressed correct behavior. After applying a simple NPO objective to detected erroneous outputs, 45 out of 495 previously incorrect predictions become correct. This does not imply that the model fully encodes the

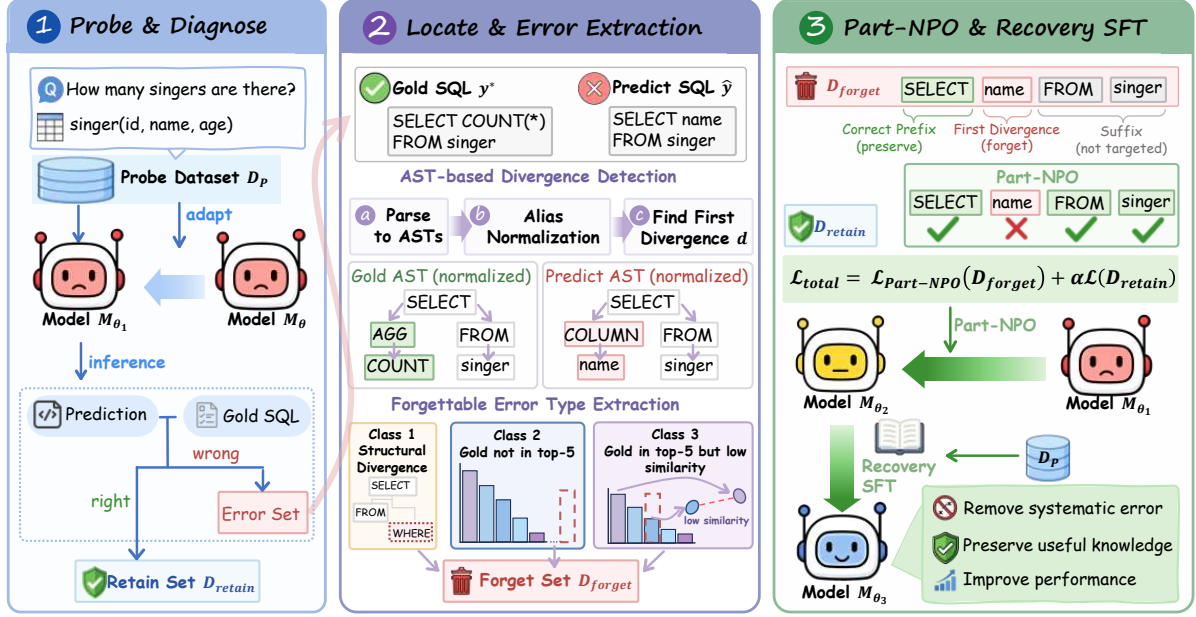


Figure 2: Overview of the TTU-SQL framework. The pipeline is organized into three modules: probe-based diagnosis, AST-based localization with forgettable error extraction, and Part-NPO followed by recovery SFT.

gold SQL in all such cases. Rather, it suggests that some correct behaviors are partially accessible under the model distribution and can emerge once dominant erroneous continuations are suppressed.

Coarse-grained unlearning is overly destructive. Applying unlearning to the entire erroneous sequence can damage useful SQL knowledge because many incorrect predictions contain long correct prefixes. This observation motivates selective unlearning at localized divergence positions rather than sequence-level suppression.

3.3 Background: Negative Preference Optimization

Negative Preference Optimization (Zhang et al., 2024) was originally proposed for knowledge unlearning in LLMs. Its core idea is to reduce the generation probability of undesirable outputs without requiring correct answers.

Given forget data $(x, \hat{y})$ and a reference model π_{ref} , the NPO loss is:

$$\mathcal{L}_{NPO} = -\frac{2}{\beta} \mathbb{E}_{(x, \hat{y})} \left[\log \sigma \left(-\beta \cdot \log \frac{\pi_{\theta}(\hat{y}|x)}{\pi_{ref}(\hat{y}|x)} \right) \right]$$

where σ is the sigmoid function and $\beta > 0$ is the inverse-temperature coefficient. Intuitively, NPO reduces $\pi_{\theta}(\hat{y}|x)$ relative to $\pi_{ref}(\hat{y}|x)$, thereby suppressing the undesirable output while avoiding the unstable dynamics of naive gradient ascent.

4 Targeted Token-level Unlearning for SQL

Given a Text-to-SQL model M_{θ} whose original training data is unknown, TTU-SQL repairs recurring Text-to-SQL errors through the probe-based framework in Figure 2. The framework first uses a labeled probe dataset D_P to expose recurring errors, then localizes the first erroneous decision point through AST-based divergence detection. Logit-level criteria select forgettable errors, Part-NPO suppresses localized erroneous tokens, and recovery SFT restores correct SQL generation after unlearning.

4.1 Error Pattern Detection with AST-Based Divergence Detection

Given a model prediction $\hat{y}$ and its corresponding gold standard y^* , the goal is to localize the earliest position at which the predicted SQL departs from the gold program structure. Due to the structured nature of SQL, simple string comparison can produce false positives because semantically equivalent queries may have different surface forms. We therefore adopt an AST-based pipeline that first parses both queries into abstract syntax trees using `sqlglot` (Mao, 2023), then normalizes table aliases to reduce mismatches caused by aliasing differences. On the normalized ASTs, we perform node-by-node comparison to identify the first structural divergence position, which corresponds to the

earliest erroneous generation decision detected in the AST space. Finally, we map this AST divergence position back to the original token sequence position, obtaining the divergence token d . Additional implementation details and examples are provided in Appendix B.1.

4.2 Forgettable Error Type Extraction

Effective token-level unlearning requires localizable errors whose divergence is associated with an over-prioritized wrong continuation. We therefore use the divergence position d as an anchor to analyze the model’s local preference behavior and select errors suitable for targeted suppression. The extraction procedure focuses on three such error types, summarized in the error extraction module of Figure 2.

Class 1: Structural Divergence. The token at the divergence position introduces a structural change in the SQL AST, such as an unnecessary GROUP BY, WHERE, or JOIN clause. These errors correspond to local structural decision biases that alter the semantics of the generated query.

Class 2: Gold Not in Top-5. The correct token y_d^* is not among the model’s top-5 logit-ranked candidates at the divergence position. This indicates that the correct token receives low local preference under the current model distribution. In such cases, limited-step SFT may need to overcome a large initial logit gap before the correct token becomes competitive.

Class 3: Gold in Top-5 but Low Similarity. The correct token y_d^* appears among the top-5 candidates, but the cosine similarity between its embedding and the actually predicted token is below a threshold $\tau = 0.8$. This indicates that the model assigns non-negligible probability to the correct token but still prefers a semantically distant alternative. Such cases capture local wrong-preference errors rather than complete absence of the correct candidate.

4.3 Part-NPO: Token-Level Selective Unlearning

Standard NPO applies the unlearning objective to the entire erroneous sequence $\hat{y}$. However, Text-to-SQL errors are often locally concentrated: an incorrect query may contain a correct prefix before diverging at a specific decision point. Suppressing the entire sequence therefore risks penalizing

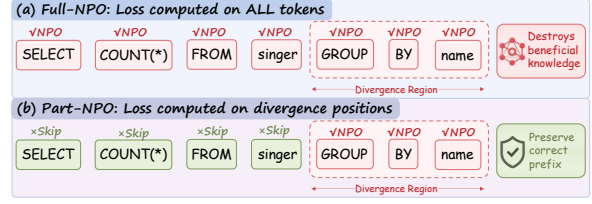


Figure 3: Comparison of Full-NPO and Part-NPO. Part-NPO preserves the correct prefix by applying the unlearning loss only to localized divergence tokens.

useful SQL generation behavior together with the actual erroneous continuation.

Part-NPO is designed to avoid this over-suppression by applying the NPO objective only to localized divergence tokens. As illustrated in Figure 3, Full-NPO suppresses all tokens in the erroneous sequence, whereas Part-NPO preserves the correct prefix and applies the unlearning signal only at the localized divergent decision points.

While Full-NPO optimizes over all tokens in the predicted sequence,

$$\mathcal{L}_{\text{Full-NPO}} = -\frac{2}{\beta} \sum_{t=1}^T \log \sigma \left(-\beta \log \frac{\pi_{\theta}(\hat{y}_t | \hat{y}_{<t}, x)}{\pi_{\text{ref}}(\hat{y}_t | \hat{y}_{<t}, x)} \right)$$

Part-NPO restricts the unlearning signal to the divergence-token set:

$$\mathcal{L}_{\text{Part-NPO}} = -\frac{2}{\beta} \sum_{t \in D} \log \sigma \left(-\beta \log \frac{\pi_{\theta}(\hat{y}_t | \hat{y}_{<t}, x)}{\pi_{\text{ref}}(\hat{y}_t | \hat{y}_{<t}, x)} \right)$$

Here, $t \in D$ means that the NPO loss is applied only to tokens selected by divergence detection. In our final implementation, $D = \{d\}$, where d is the first divergence position identified for each forget sample. Thus, Part-NPO suppresses only the first divergent token, rather than the full sequence or the post-divergence suffix.

4.4 Optimization Procedure

Given a probe dataset $D_P = \{(x_i, y_i^*)\}_{i=1}^N$, TTU-SQL optimizes the model through the diagnose–locate–unlearn–recover procedure introduced above. We first perform a brief probe adaptation step on D_P , obtaining M_{θ_1} . This step is not intended as the final optimization objective; rather, it places the model under the probe distribution so

that recurring error tendencies become observable across diverse SQL structures.

We then run M_{θ_1} on D_P and collect erroneous predictions. For each error $(x, \hat{y}, y^*)$, AST-based divergence detection identifies the first divergence token d . The error-type extraction procedure in Section 4.2 selects forgettable errors into D_{forget} , while correctly predicted probe examples are sampled as D_{retain} to stabilize unlearning.

The unlearning stage uses M_{θ_1} as the reference model and optimizes Part-NPO on D_{forget} , together with a standard retain loss:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{Part-NPO}}(D_{\text{forget}}) + \alpha \mathcal{L}(D_{\text{retain}}).$$

This produces an intermediate model M_{θ_2} in which localized erroneous continuations are suppressed. Finally, recovery SFT continues training on D_P , yielding M_{θ_3} . The resulting model combines localized suppression of recurrent wrong continuations with renewed likelihood training on correct SQL behavior.

5 Experiments

5.1 Dataset

Probe Dataset. We construct a probe dataset by sampling approximately 50,000 single-turn examples from OmniSQL (Li et al., 2025), a large-scale synthetic Text-to-SQL corpus with diverse SQL structures. The probe dataset is used to expose recurring errors and to support unlearning and recovery training. For compatibility with target benchmarks, we retain examples matching the target structural format.

Evaluation Dataset. We evaluate on Spider Dev (Yu et al., 2018), a cross-domain Text-to-SQL benchmark, and BIRD Dev (Li et al., 2023b), a more challenging real-world database-grounded benchmark.

5.2 Evaluation Metrics

We report execution accuracy (EX), which measures whether the predicted SQL returns the same result as the gold SQL when executed on the corresponding database. EX is insensitive to semantically equivalent surface-form variations and is the standard metric for Text-to-SQL evaluation.

5.3 Implementation Details

Models. We evaluate TTU-SQL on DeepSeek-R1-Distill-Llama-8B and DeepSeek-R1-Distill-Qwen-7B. For each benchmark, the corresponding

base model has been supervised-fine-tuned on the benchmark’s training split before applying TTU-SQL, so the starting point already has strong Text-to-SQL generation accuracy. This setting makes the comparison conservative by testing whether TTU-SQL can further repair residual systematic errors in an already task-adapted model.

Training Configuration. All training stages use LoRA (Hu et al., 2022) with rank $r = 32$ and scaling factor $\alpha = 64$, targeting query and value projection layers. Probe adaptation trains for 1 epoch with learning rate 2×10^{-5} . Part-NPO trains for 1 epoch with learning rate 1×10^{-5} , $\beta = 0.1$, and retain loss weight 1.0. Recovery SFT trains for 7 epochs with learning rate 2×10^{-5} . All stages use cosine scheduling, gradient checkpointing, BF16 training, and a single A100 80GB GPU.

5.4 Baselines

Table 1 is organized into two panels. The **direct-comparison panel** lists parameter-update methods run under identical localized data, retain set, LoRA configuration, and total optimizer steps: (1) **Base Model**, the task-adapted model before applying probe-set adaptation and unlearning; (2) **Direct SFT**, the base model further trained on the probe dataset for the same total number of epochs as TTU-SQL, which controls for the effect of additional probe-set training; (3) **Localized DPO**, which optimizes a DPO objective on gold-vs.-predicted pairs at the divergence token; (4) **SFT on selected error subset + retain**, which trains only on the localized forget subset plus a retain set; (5) **Weighted SFT at divergence tokens**, which up-weights the divergence-token loss; and (6) TTU-SQL. Complete objective definitions and implementation details for (3)–(5) are in Appendix C.1. The **composition panel** applies five training-free inference-time enhancement methods to the Base, Direct SFT, and TTU-SQL models under the identical inference configuration, so that the composition gain attributable specifically to TTU-SQL is visible. These methods are not parameter-only baselines: they use additional inference-time resources such as schema retrieval, database probing, output correction modules, execution feedback, or multi-candidate decoding. We group them into three categories: (3) **Input preprocessing**: Rethinking Schema Linking (Nahid et al., 2025) prunes irrelevant schema elements via bidirectional retrieval; DIVER (Nan et al., 2026) resolves entity-value

Method	DeepSeek-R1-Distill-Llama-8B		DeepSeek-R1-Distill-Qwen-7B	
	Spider	BIRD	Spider	BIRD
<i>Direct comparison (parameter-update methods, equal localized data and compute)</i>				
Base Model	61.41	32.79	58.51	31.94
Direct SFT	66.15	33.57	62.48	33.24
Localized DPO (gold vs. predicted)	67.70	34.49	64.31	34.09
SFT on selected error subset + retain	66.44	33.38	62.57	33.12
Weighted SFT at divergence tokens	64.80	33.38	61.41	32.79
TTU-SQL	68.57	35.53	65.57	34.87
<i>Panel 2: Composition with inference-time enhancement (macro-avg over the four settings)</i>				
Inference-time method	Base + method	Direct SFT + method	TTU-SQL + method	
Rethinking Schema Linking	48.62	50.74	52.91	
DIVER	48.61	50.36	52.01	
SEA-SQL	54.62	55.21	55.93	
SHARE	52.75	53.48	54.60	
MBR-SQL (N=10)	46.50	49.32	51.37	

Table 1: Main results on Spider and BIRD Dev (EX, %). **Panel 1 (top):** direct comparison of parameter-update methods under equal localized data, retain set, LoRA configuration, and total optimizer steps. **Panel 2 (bottom):** composition with training-free inference-time enhancement methods; each method is applied to Base, Direct SFT, and TTU-SQL under the identical inference configuration. Panel 2 values are macro-averages over the four model–dataset pairs; per-setting values are reported in Appendix 5.

mismatches through dynamic database probing. **(4) Output post-processing:** SEA-SQL (Li et al., 2024a) eliminates output biases via a specialized correction model; SHARE (Qu et al., 2025) performs hierarchical SQL correction through three lightweight SLMs. **(5) Multi-candidate selection:** MBR-SQL (Borchmann and Wydmuch, 2025) selects the optimal SQL via execution-guided Minimum Bayes Risk decoding with 10 candidates. Detailed descriptions are in Appendix C.

6 Results and Analysis

6.1 Overall Results

Table 1 shows that TTU-SQL consistently improves over Direct SFT across both evaluated models and datasets. On Spider Dev, TTU-SQL improves DeepSeek-R1-Distill-Llama-8B from 66.15% to 68.57% and DeepSeek-R1-Distill-Qwen-7B from 62.48% to 65.57%. On BIRD Dev, TTU-SQL improves DeepSeek-R1-Distill-Llama-8B from 33.57% to 35.53% and DeepSeek-R1-Distill-Qwen-7B from 33.24% to 34.87%. These results suggest that suppressing systematic erroneous continuations before recovery training is more effective than allocating the same training budget to Direct SFT alone.

The gains on BIRD are smaller than those on Spider, which is expected because BIRD contains more real-world grounding challenges, including value linking and schema-value alignment. Such

errors are less likely to be fully addressed by suppressing local SQL generation biases alone. This indicates that TTU-SQL is most effective when the dominant failure mode lies in generation preference rather than external grounding. TTU-SQL does not repair all Text-to-SQL failure types. A category-level before/after analysis (Appendix D.9) shows that the largest error reductions occur in the local generation categories it targets, namely column selection, structural operators, aggregation, and grouping/ordering. In contrast, condition/value grounding and non-local errors change little.

TTU-SQL is also complementary to inference-time enhancement methods. All five inference-time baselines achieve higher accuracy when combined with TTU-SQL than when applied to the base model alone. Several of these methods use additional inference-time resources, so their standalone scores should not be interpreted as direct parameter-only comparisons with TTU-SQL. Importantly, ordinary continued SFT is also complementary to these inference-time methods: applying each method to the Direct SFT model already improves over applying it to the base model. The composition gain is therefore not unique in kind to TTU-SQL. However, under the identical inference configuration, TTU-SQL retains an additional 0.72–2.17 point advantage over Direct SFT across the five methods, indicating that localized parameter repair provides benefits beyond continued SFT even

	EX (%)
TTU-SQL	65.57
NPO on all errors	63.82
w/o recovery SFT	44.20
Class 1 only	64.12
Class 2 only	64.50
Class 3 only	63.44
Class 1+2	64.22
Class 1+3	63.64
Class 2+3	64.89
Class 1+2+3	65.57
Full-NPO	64.22
Part-NPO (divergence sequence)	64.70
Part-NPO (divergence token)	65.57

Table 2: Ablation study on Spider with DeepSeek-R1-Distill-Qwen-7B.

when both are composed with the same inference-time enhancement.

6.2 Ablation Study

Table 2 examines the main design choices of TTU-SQL, including forget-set selection, recovery training, and unlearning granularity. All variants are evaluated on Spider with DeepSeek-R1-Distill-Qwen-7B under the same training configuration.

Effect of forget-set selection. Targeted forget-set construction is more effective than unlearning all erroneous predictions. NPO on all errors reaches 63.82% EX, while the full TTU-SQL reaches 65.57%. This gap suggests that the three specific error types defined in Section 4.2 provide cleaner unlearning targets than the full error set, which may contain noisy, non-local, or structurally under-specified failures. Among the individual classes, Class 2 performs best at 64.50%, and combining all three classes further improves the result to 65.57%, indicating that the extracted classes capture complementary error patterns.

Effect of recovery SFT. Removing recovery SFT leads to a substantial drop to 44.20% EX. This result shows that NPO alone is not sufficient as a final model update. Although NPO can suppress erroneous continuations, it may also perturb general SQL generation behavior and reduce output fluency. Recovery SFT is therefore necessary to restore fluent SQL generation and convert the weakened erroneous preferences into improved task performance.

Effect of unlearning granularity. We compare three granularities for applying the unlearning objective. Full-NPO applies NPO to the entire erro-

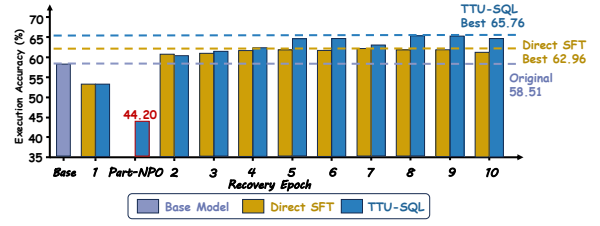


Figure 4: Optimization trajectories of Direct SFT and TTU-SQL on Spider Dev with DeepSeek-R1-Distill-Qwen-7B.

neous sequence and obtains 64.22% EX. Part-NPO with divergence sequence applies NPO to every token from the divergence position to the end of the sequence, improving the result to 64.70%. However, this suffix-level objective can still suppress tokens beyond the actual error trigger and may penalize useful continuations after the first divergence point. The final Part-NPO with divergence token applies NPO only to the token at the localized divergence position and achieves the best result of 65.57%, suggesting that the main repairable signal lies in the localized erroneous decision rather than the entire generated sequence.

6.3 Optimization Trajectory Analysis

Figure 4 compares the training trajectories of Direct SFT and TTU-SQL on Spider Dev with DeepSeek-R1-Distill-Qwen-7B. Direct SFT steadily improves the base model, but its gains gradually saturate. In contrast, TTU-SQL applies Part-NPO after the first epoch, which causes a sharp temporary drop in execution accuracy. This drop is expected: Part-NPO is an unlearning step that suppresses localized erroneous continuations, and is not intended to serve as a standalone final model.

After the erroneous patterns are weakened, recovery SFT quickly restores SQL generation ability and further improves performance beyond the Direct SFT trajectory. This optimization trajectory suggests that the benefit of TTU-SQL does not come from additional SFT epochs alone. Rather, it comes from first weakening recurring error patterns and then relearning correct SQL generation behavior on the repaired model state.

7 Discussion

Unlearning as post-hoc repair. The results suggest that continued fine-tuning and post-hoc repair address different aspects of Text-to-SQL improvement. Direct SFT reinforces gold SQL outputs,

whereas TTU-SQL explicitly weakens localized wrong continuations before recovery training. The trajectory in Figure 4 illustrates this distinction: Part-NPO first perturbs learned generation behavior, and recovery SFT then rebuilds SQL generation from a repaired model state. This supports viewing unlearning not only as data or knowledge removal, but also as a tool for repairing recurring generation biases in structured prediction.

Scope of token-level repair. The effectiveness of TTU-SQL depends on whether errors are localizable. Many Text-to-SQL failures contain useful partial structures before diverging at a local decision point, explaining why token-level Part-NPO outperforms sequence-level suppression. However, errors dominated by global query planning, compositional reasoning, or schema-value grounding may require structure-level repair or additional grounding mechanisms. This boundary is consistent with the smaller gains on BIRD. The parameter-level interpretation is therefore evidence-supported for localizable errors, not a causal claim for all SQL failure types: recurring errors persist under semantics-preserving perturbations such as question paraphrases and schema-identifier renamings, indicating that their roots lie in internal model preferences rather than external prompt artifacts, while residual errors and the smaller BIRD gains reflect that schema linking, value grounding, and other global capabilities exceed the current repair scope.

8 Conclusion

We proposed TTU-SQL, a targeted token-level unlearning framework for repairing systematic error patterns in Text-to-SQL models. Rather than treating incorrect predictions as entirely wrong sequences, TTU-SQL uses a probe dataset to expose recurring failures, localizes the first divergent decision through AST-based comparison, and applies Part-NPO only to the localized erroneous token before recovery SFT. Experiments on Spider and BIRD show that this selective suppress-then-recover strategy improves over equal-budget training-based repair baselines and can be combined with inference-time enhancement methods. These findings indicate that machine unlearning can be used not only for removing data influence, but also for parameter-level repair of recurrent generation preferences in structured semantic parsing.

9 Limitations

This work has several limitations. First, TTU-SQL requires a labeled probe dataset to expose errors and identify divergence positions; the unavailability of labeled probe data is an explicit limitation of the current framework. When gold SQL annotations are unavailable, the same framework would require reliable alternatives such as execution feedback, verified candidate sets, or other weak supervision signals.

Second, TTU-SQL is designed for recurring errors that can be localized to token-level divergence points. It is less suitable for failures dominated by global query planning, multi-step compositional reasoning, or schema-value grounding, where the necessary correction may span multiple clauses or depend on external database evidence.

Third, the current error extraction rules include implementation choices, such as the top-5 logit criterion and the Class 3 similarity threshold, that may require recalibration for different tokenizers, model families, or SQL dialects. Finally, our experiments cover two SQL styles (Spider and BIRD, both SQLite-based); extending TTU-SQL to additional SQL dialects (e.g., DuckDB, PostgreSQL, MySQL) and to other structured generation tasks remains an important direction for future work.

References

- Łukasz Borchmann and Marek Wydmuch. 2025. [Query and conquer: Execution-guided SQL generation](#). *arXiv preprint arXiv:2503.24364*.
- Lucas Bourtole, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. [Machine unlearning](#). In *2021 IEEE Symposium on Security and Privacy*, pages 141–159. IEEE.
- Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. [Teaching large language models to self-debug](#). In *International Conference on Learning Representations*.
- 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):1132–1145.
- 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 *International Conference on Learning Representations*.

- Joel Jang, Dongkeun Yoon, Sohee Yang, Sungmin Cha, Moontae Lee, Lajanugen Logeswaran, and Minjoon Seo. 2023. [Knowledge unlearning for mitigating privacy risks in language models](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pages 14389–14408. Association for Computational Linguistics.
- Pang Wei Koh and Percy Liang. 2017. [Understanding black-box predictions via influence functions](#). In *Proceedings of the 34th International Conference on Machine Learning*, pages 1885–1894. PMLR.
- Chaofan Li, Yingxia Shao, Yawen Li, and Zheng Liu. 2024a. [SEA-SQL: Semantic-enhanced text-to-SQL with adaptive refinement](#). *arXiv preprint arXiv:2408.04919*.
- Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tiejing Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. [OmniSQL: Synthesizing high-quality text-to-SQL data at scale](#). *Proceedings of the VLDB Endowment*, 18(11):4695–4709.
- Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. [Resdsq: Decoupling schema linking and skeleton parsing for text-to-sql](#). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. [Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs](#). In *Advances in Neural Information Processing Systems*.
- Nathaniel Li, Alexander Pan, Anjali Gopal, Summer Yue, Daniel Berrios, Alice Gatti, Justin D Li, Ann-Kathrin Dombrowski, Shashwat Goel, Long Phan, and 1 others. 2024b. [The wmdp benchmark: Measuring and reducing malicious use with unlearning](#). *arXiv preprint arXiv:2403.03218*.
- Zhaodonghui Li, Haitao Yuan, Huiming Wang, Gao Cong, and Lidong Bing. 2024c. [LLM-R2: A large language model enhanced rule-based rewrite system for boosting query efficiency](#). *Proceedings of the VLDB Endowment*, 18(1):53–65.
- Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary C. Lipton, and J. Zico Kolter. 2024. [TOFU: A task of fictitious unlearning for LLMs](#). In *Proceedings of the First Conference on Language Modeling*.
- Toby Mao. 2023. [SQLGlot: A SQL transpiler and optimizer](#). <https://github.com/tobymao/sqlglot>.
- Md Mahadi Hasan Nahid, Davood Rafiei, Weiwei Zhang, and Yong Zhang. 2025. [Rethinking schema linking: A context-aware bidirectional retrieval approach for text-to-SQL](#). *arXiv preprint arXiv:2510.14296*.
- Yafeng Nan, Haifeng Sun, Zirui Zhuang, Qi Qi, Guojun Chu, Jianxin Liao, Dan Pei, and Jingyu Wang. 2026. [DIVER: A robust text-to-SQL system with dynamic interactive value linking and evidence reasoning](#). *arXiv preprint arXiv:2602.12064*.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. [Training language models to follow instructions with human feedback](#). *Advances in neural information processing systems*, 35:27730–27744.
- Mohammadreza Pourreza and Davood Rafiei. 2023. [DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction](#). In *Advances in Neural Information Processing Systems*.
- 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 the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11268–11292.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. [Direct preference optimization: Your language model is secretly a reward model](#). In *Advances in Neural Information Processing Systems*.
- Jinwang Song, Hongying Zan, Kunli Zhang, Lingling Mu, Yingjie Han, Haobo Hua, and Min Peng. 2025. [JOLT-SQL: Joint loss tuning of text-to-SQL with confusion-aware noisy schema sampling](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 6040–6053. Association for Computational Linguistics.
- Mattia Trito, Giuseppe Farano, Dario Di Palma, Gaetano Rossiello, Fedelucio Narducci, Dharmashankar Subramanian, and Tommaso Di Noia. 2025. [GradeSQL: Outcome reward models for ranking SQL queries from large language models](#). *arXiv preprint arXiv:2509.01308*.
- Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. [RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578. Association for Computational Linguistics.
- Yuanshun Yao, Xiaojun Xu, and Yang Liu. 2024. [Large language model unlearning](#). In *Advances in Neural Information Processing Systems*.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. [Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921. Association for Computational Linguistics.

Ruiqi Zhang, Licong Lin, Yu Bai, and Song Mei. 2024. [Negative preference optimization: From catastrophic collapse to effective unlearning](#). In *Proceedings of the First Conference on Language Modeling*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2018. [Seq2SQL: Generating structured queries from natural language using reinforcement learning](#). In *International Conference on Learning Representations*.

A Reproducibility Details

A.1 Datasets and Splits

Probe dataset. We use OmniSQL (Li et al., 2025) as the source of the probe dataset D_P . OmniSQL is a large-scale synthetic Text-to-SQL corpus designed to cover diverse database schemas, natural language question forms, and SQL program structures. In this work, OmniSQL is not used as a target benchmark. Instead, it serves as a diagnostic and repair source: we use it to expose recurring error tendencies, construct forget and retain sets, perform Part-NPO unlearning, and conduct recovery SFT.

We construct D_P by sampling approximately 50,000 examples from OmniSQL. The sampling procedure is designed to provide broad SQL-structure coverage while avoiding any use of target evaluation examples. We first retain single-turn Text-to-SQL examples, so that each instance contains one natural language question, one database schema context, and one gold SQL query. Then we apply target-style filtering to keep examples structurally compatible with the intended evaluation setting. For Spider-targeted experiments, we additionally remove examples requiring external knowledge outside the database schema, since Spider-style evaluation focuses on database-grounded semantic parsing rather than open-domain factual reasoning.

The resulting probe dataset is used in four stages. First, it is used for probe adaptation to place the model under the probe distribution. Second, model predictions on this dataset are used to expose recurring errors. Third, the erroneous and correct predictions are used to construct D_{forget} and D_{retain} . Finally, the same probe dataset is used for recovery SFT after unlearning. Importantly, the probe dataset is separate from the target evaluation sets and does not include Spider Dev or BIRD Dev examples.

Evaluation datasets. We evaluate on Spider Dev (Yu et al., 2018) and BIRD Dev (Li et al.,

2023b). Spider is a cross-domain Text-to-SQL benchmark that evaluates generalization to unseen databases and SQL structures. The Spider development set contains 1,034 questions over 20 databases, and is widely used for evaluating semantic parsing and Text-to-SQL generalization.

BIRD is a real-world database-grounded Text-to-SQL benchmark with more challenging schema-value grounding requirements. Compared with Spider, BIRD contains more realistic database contents and places greater emphasis on value linking, schema-value alignment, and database-specific grounding. This makes BIRD a useful testbed for examining whether TTU-SQL remains effective when the dominant failure mode is not only SQL generation preference but also grounding.

Both Spider Dev and BIRD Dev are used only for evaluation. They are never used during probe adaptation, error detection, forget-set construction, Part-NPO training, recovery SFT, or hyperparameter selection. This separation ensures that the recurring error patterns repaired by TTU-SQL are detected from the probe dataset rather than from the target evaluation data.

A.2 Error Category Annotation for Motivation Analysis

To support the motivation analysis in Figure 1(a), we conduct a lightweight error-category annotation on Spider Dev using DeepSeek-R1-Distill-Qwen-7B. We first run the model on Spider Dev and collect predictions that fail execution matching against the gold SQL. For each erroneous prediction, we parse both the predicted SQL and the gold SQL into ASTs using sqlglot, normalize aliases and surface-form variants, and identify the first structural divergence point.

Each error is then assigned to a coarse SQL error category according to the clause or operator family associated with the first divergence. The categories include column selection, aggregation, grouping or ordering, condition/value prediction, join/table selection, nested or set-operator structure, and other query-structure errors. Predictions that cannot be parsed or cannot be assigned to a clear category are assigned to the “Other” category.

The percentages reported in Figure 1(a) are computed over all erroneous Spider Dev predictions in this analysis. This annotation is used only for motivating the presence of recurring SQL error categories. It is separate from the forgettable error types used by TTU-SQL, which are defined by AST

divergence and logit-level criteria in Section 4.2.

A.3 Evaluation Protocol

We report execution accuracy (EX) for all main experiments. For each example, the predicted SQL and the gold SQL are executed against the corresponding database. A prediction is counted as correct if its execution result matches the execution result of the gold SQL. Invalid SQL queries, execution errors, and timeouts are counted as incorrect. This protocol avoids penalizing semantically equivalent SQL queries that differ only in surface form.

All methods are evaluated using the same database files, schema serialization format, and execution evaluator. Unless otherwise specified, each method produces one final SQL prediction per question. For multi-candidate methods such as MBR-SQL, the selected candidate after the method’s internal selection procedure is treated as the final prediction. The target evaluation sets are never used for selecting forget samples, tuning unlearning targets, or choosing intermediate checkpoints for the main results.

The main result tables report performance under the fixed training configuration in Appendix A.4. Figure 4 reports an optimization trajectory and marks the best checkpoint along the plotted curve. These trajectory checkpoints are used for analysis and should be distinguished from the fixed-protocol results reported in the main table.

A.4 Training Hyperparameters

All training stages use LoRA (Hu et al., 2022). Table 3 summarizes the complete training configuration used for both model families. The same configuration is used for the corresponding Direct SFT baseline, except that Direct SFT removes the Part-NPO stage and allocates the same overall training budget to supervised fine-tuning.

A.5 Compute and Implementation

All experiments use mixed-precision BF16 training, cosine learning-rate scheduling, and gradient checkpointing. The main training runs are conducted on a single A100 80GB GPU. AST parsing, alias normalization, divergence localization, and forget-set construction are implemented as offline preprocessing steps. These preprocessing steps are CPU-based and do not require gradient computation.

During training, probe adaptation and recovery SFT optimize the standard next-token likelihood on gold SQL outputs. The Part-NPO stage optimizes the token-level NPO loss on D_{forget} , together with a retain loss on D_{retain} . All trainable stages use the same schema serialization and prompt format unless otherwise specified in Appendix C.3.

A.6 Randomness and Variance

To assess the stability of TTU-SQL, we report three independent runs under the same training configuration. Run 1 corresponds to the values reported in Table 1. As shown in Table 4, TTU-SQL consistently improves over Direct SFT across all runs and all evaluated model–dataset pairs.

The mean results show the same trend as the main table. TTU-SQL improves over Direct SFT by 2.51 points on Llama-8B Spider, 2.02 points on Llama-8B BIRD, 3.13 points on Qwen-7B Spider, and 1.63 points on Qwen-7B BIRD. The standard deviations of TTU-SQL are comparable to those of Direct SFT, indicating that the gains are stable across repeated runs rather than being caused by a single favorable seed.

B Additional Method Details

B.1 AST-Based Divergence Detection

The goal of divergence detection is to identify the earliest token-level decision where the predicted SQL begins to deviate from the gold SQL. A direct string comparison is insufficient because SQL queries can be semantically equivalent while differing in aliases, table qualifiers, or other surface-form choices. We therefore perform divergence detection in a normalized AST space and then map the detected structural divergence back to the original token sequence.

The procedure consists of five steps. First, both the predicted SQL $\hat{y}$ and the gold SQL y^* are parsed into ASTs using `sqlglot` (Mao, 2023). If either query cannot be parsed, the example is treated as non-localizable and is excluded from token-level forget-set construction. Second, we normalize table aliases. For instance, two queries using `T1.name` and `driver.name` may express the same reference after alias resolution. Alias normalization maps such surface forms to a canonical representation to avoid false positive divergences. Third, we compare the normalized ASTs node by node. The comparison prioritizes structural nodes such as projection, aggregation, filtering, grouping, joining, and

Stage	Description	DeepSeek-R1-Distill-Llama-8B	DeepSeek-R1-Distill-Qwen-7B
Stage 1	Probe dataset adaptation	LR 2×10^{-5} , 1 epoch	LR 2×10^{-5} , 1 epoch
Stage 3	Part-NPO training	LR 1×10^{-5} , 1 epoch, $\beta = 0.1$, retain weight = 1.0	LR 1×10^{-5} , 1 epoch, $\beta = 0.1$, retain weight = 1.0
LoRA	Low-rank adaptation configuration	$r = 32$, $\alpha = 64$, query and value projections	$r = 32$, $\alpha = 64$, query and value projections
Stage 4	Recovery SFT	LR 2×10^{-5} , 7 epochs	LR 2×10^{-5} , 7 epochs

Table 3: Complete training hyperparameters for all model configurations.

Method	Run	Llama-8B Spider	Llama-8B BIRD	Qwen-7B Spider	Qwen-7B BIRD
Direct SFT	Run 1	66.15	33.57	62.48	33.24
Direct SFT	Run 2	65.38	32.99	61.70	32.72
Direct SFT	Run 3	66.73	34.03	63.15	33.77
Direct SFT	Mean $\pm$ Std.	66.09 ± 0.68	33.53 ± 0.52	62.44 ± 0.73	33.25 ± 0.53
TTU-SQL	Run 1	68.57	35.53	65.57	34.87
TTU-SQL	Run 2	67.89	34.94	64.80	34.29
TTU-SQL	Run 3	69.34	36.18	66.34	35.46
TTU-SQL	Mean $\pm$ Std.	68.60 ± 0.73	35.55 ± 0.62	65.57 ± 0.77	34.87 ± 0.59

Table 4: Variance over three runs. Run 1 corresponds to the main results in Table 1.

ordering operations. Fourth, the first mismatched AST node is selected as the structural divergence point. This node represents the earliest place where the prediction begins to follow a different SQL construction from the gold query. Finally, we map the divergent AST node back to the original predicted SQL tokens and obtain the divergence position d . If a divergent AST node spans multiple tokens, we use the first token in the corresponding span as the localized decision point.

Localization validity audit. To validate the accuracy of the identified divergence positions, two annotators independently audited 300 randomly sampled erroneous predictions on Spider/DeepSeek-R1-Distill-Qwen-7B and judged whether the detected node was the earliest genuine semantic divergence. The inter-annotator agreement was $\kappa = 0.86$. Of the 300 audited cases, 274 (91.3%) were exact localizations, 17 (5.7%) were semantically ambiguous but still within the erroneous branch, and 9 (3.0%) were false localizations. Localization is applied only to predictions that fail execution matching, so a fully execution-equivalent alternative is never entered into D_{forget} ; the implementation canonicalizes aliases and surface-form variants before comparing semantic AST nodes, which handles representative equivalence cases such as alias differences, commutative predicate order, and equivalent subquery-vs.-join reformulations. To assess the impact of localization errors on the main results, we compared the EX score of the automatic

pipeline (65.57) against the score obtained when the 26 ambiguous and false-localization cases are excluded (65.49); the negligible difference (0.08 EX) indicates that the main result is not driven by localization errors.

B.2 Forgettable Error Type Extraction

After obtaining the divergence position d , we inspect the model’s local behavior at that position. The extraction procedure is designed to select errors that are suitable for targeted unlearning. It is not intended to form an exhaustive taxonomy of all Text-to-SQL errors.

For each erroneous prediction, we collect the model’s logit distribution at position d , the predicted token $\hat{y}_d$, the gold token y_d^* , and the local AST divergence information. We then apply three extraction criteria.

Class 1: Structural divergence. An error belongs to Class 1 if the predicted token introduces a structural change in the SQL AST. Typical cases include unnecessary or incorrect structural operators such as GROUP BY, WHERE, JOIN, ORDER BY, or set-operation keywords. These tokens often change the semantics of the entire query even though the prefix before them may be correct. Such errors are suitable for token-level unlearning because the harmful behavior is concentrated at a structural decision point.

Class 2: Gold not in top-5. An error belongs to Class 2 if the gold token y_d^* is not among the

top-5 candidates under the model’s logit distribution at the divergence position. This indicates that the correct continuation is locally under-prioritized. In this case, Direct SFT must both increase the probability of the gold token and overcome the dominance of several competing tokens. Suppressing the predicted wrong token can reduce this local competition before recovery training.

Class 3: Gold in top-5 but low similarity. An error belongs to Class 3 if the gold token appears among the top-5 candidates but is semantically distant from the predicted token. We compute cosine similarity using the model token embeddings. If the similarity between y_d^* and $\hat{y}_d$ is lower than $\tau = 0.8$, the sample is selected. This captures cases where the model has not completely lost access to the correct token, but still prefers a substantially different continuation.

Automatic extraction procedure. The three error types above are generated automatically from AST structure and model logits; no human annotation is involved. Algorithm 1 summarizes the end-to-end procedure.

B.3 Forget and Retain Set Construction

Forget and retain sets are constructed from probe-set predictions after probe adaptation. We first run M_{θ_1} on D_P and compare each prediction with the gold SQL using execution accuracy. Correctly executed predictions are placed into the candidate retain pool. Incorrect predictions are passed to the divergence detection and error-type extraction pipeline.

For each incorrect prediction $(x, \hat{y}, y^*)$, we identify the first divergence position d . If the example satisfies at least one of the three forgettable error criteria in Appendix B.2, it is added to D_{forget} . Each forget example stores the input x , the predicted SQL $\hat{y}$, the gold SQL y^* , the divergence position d , the predicted divergence token $\hat{y}_d$, and the assigned error type. When multiple criteria are satisfied, the example is included once in the union forget set.

The retain set D_{retain} is sampled from correctly predicted probe examples. Its role is to stabilize unlearning and reduce degradation of useful SQL generation behavior. During Part-NPO, forget samples contribute token-level NPO loss at $D = \{d\}$, while retain samples contribute a standard likelihood-preserving loss. During recovery SFT, the full

probe dataset D_P is used to rebuild fluent SQL generation after unlearning.

B.4 End-to-End Procedure

The full TTU-SQL procedure is organized around a single probe dataset, but different stages use it for different purposes. Probe adaptation exposes recurring errors, divergence detection localizes erroneous decisions, Part-NPO suppresses localized wrong continuations, and recovery SFT restores general SQL generation behavior.

The pipeline begins with the base Text-to-SQL model M_θ . We first fine-tune the model on the probe dataset for one epoch, obtaining M_{θ_1} . This stage is not intended as the final optimization objective. Instead, it places the model under the probe distribution so that recurring generation tendencies can be observed through inference.

We then run M_{θ_1} on D_P and split the predictions according to execution correctness. Correct predictions form the retain candidate pool. Incorrect predictions are passed to AST-based divergence detection, which identifies the first divergence position d . The error-type extraction procedure then selects forgettable errors from these localized error records. The resulting forget set is denoted as D_{forget} , and the sampled retain set is denoted as D_{retain} .

In the unlearning stage, M_{θ_1} serves as the reference model. For each forget example, Part-NPO applies NPO only to the localized divergence token d , rather than to the entire predicted SQL sequence. At the same time, retain examples are used to preserve general SQL generation behavior. This stage produces an intermediate model M_{θ_2} . Finally, recovery SFT continues training M_{θ_2} on the probe dataset D_P , yielding the final model M_{θ_3} . The final model is used for all downstream evaluations and for combination with inference-time enhancement methods.

C Baseline Details and Combination Protocol

C.1 Internal Baselines

Base Model. The base model is the task-adapted Text-to-SQL model before applying probe-set adaptation and unlearning. It provides the reference point for measuring the effect of additional training and post-hoc repair.

Direct SFT. Direct SFT continues training the base model on the probe dataset for the same total

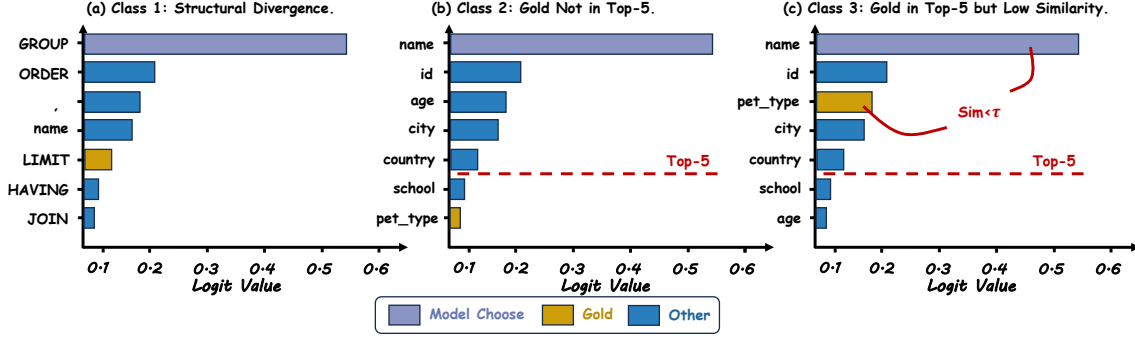


Figure 5: Logit distributions at divergence positions for the three extracted error types. (a) Class 1: SQL keyword divergence. (b) Class 2: gold token not in top-5. (c) Class 3: gold token appears in top-5 but has cosine similarity below $\tau = 0.8$.

number of training epochs as TTU-SQL, but without the unlearning stage. This baseline controls for whether the improvement of TTU-SQL comes merely from additional supervised fine-tuning. A comparison against Direct SFT is therefore central to our evaluation: if TTU-SQL improves over Direct SFT under the same training budget, the gain cannot be attributed only to more exposure to the probe dataset.

Localized DPO. Localized DPO optimizes a DPO objective on gold-vs.-predicted token pairs at the divergence position. With forget data $(x, \hat{y})$, gold y^* , reference model π_{ref} , and divergence token d , the loss is

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E} \log \sigma \left(\beta \log \frac{\pi_{\theta}(y_d^* | x, \hat{y}_{<d})}{\pi_{\text{ref}}(y_d^* | x, \hat{y}_{<d})} - \beta \log \frac{\pi_{\theta}(\hat{y}_d | x, \hat{y}_{<d})}{\pi_{\text{ref}}(\hat{y}_d | x, \hat{y}_{<d})} \right)$$

followed by the same recovery SFT stage as TTU-SQL. It uses the same localized forget set, retain set, LoRA configuration, and total optimizer steps as TTU-SQL.

SFT on selected error subset + retain. This baseline replaces Part-NPO with standard next-token SFT on D_{forget} mixed with D_{retain} , keeping all other stages (probe adaptation and recovery SFT budget) identical to TTU-SQL. It tests whether simply training on the localized error subset suffices without an explicit suppression objective.

Weighted SFT at divergence tokens. This baseline performs standard SFT on gold SQL outputs but scales the loss at the divergence token d by a constant factor $w > 1$, again with the same recovery budget and configuration as TTU-SQL. It tests

whether up-weighting the divergence token without decoupled suppression reproduces the gain of TTU-SQL.

Joint SFT (original + probe). Joint SFT is a stronger control that additionally exposes the original training corpus: it trains on a 1:9 mixture of the original benchmark training set and the probe set, while keeping the same total optimizer steps, LoRA configuration, and number of consumed examples as TTU-SQL. **Data provenance and sampling protocol.** The original-training-set portion is drawn from the official Spider/BIRD training split on which the base model was already supervised-fine-tuned; the probe portion is sampled independently from OmniSQL and is disjoint from Spider Dev and BIRD Dev, so no target-test examples are used. The 1:9 ratio fixes the probe-set exposure of TTU-SQL while additionally exposing the original corpus, isolating whether the gain of localized unlearning exceeds the gain of simply adding the original training data. Joint SFT obtains 66.44 / 34.26 / 62.86 / 33.83 EX (Llama-8B Spider/BIRD, Qwen-7B Spider/BIRD), which benefits from access to the original training data but remains 1.04–2.71 points below TTU-SQL (68.57 / 35.53 / 65.57 / 34.87), supporting the narrower conclusion that additional supervision is useful but does not account for the full gain of localized unlearning.

C.2 Inference-Time Enhancement Baselines

The inference-time baselines are training-free with respect to TTU-SQL. They modify either the input context, the generated output, or the candidate selection process without changing the model parameters. We combine them with TTU-SQL to test whether parameter-level repair and inference-time

Algorithm 1 Automatic forgettable-error-type extraction

Require: probe set D_P ; model M_θ ; top- k ($k=5$); similarity threshold $\tau=0.8$

```
1:  $D_{\text{forget}} \leftarrow \emptyset$ 
2: for each  $(x, y^*) \in D_P$  do
3:    $\hat{y} \leftarrow M_\theta(x)$ 
4:   if  $\hat{y}$  fails execution matching against  $y^*$  then
5:     parse  $\hat{y}, y^*$  to ASTs; canonicalize aliases and surface forms
6:      $d \leftarrow$  first AST divergence node, mapped to the token position
7:     if the divergence node is a structural operator change then ▷ Class 1
8:        $D_{\text{forget}} \leftarrow D_{\text{forget}} \cup \{(x, \hat{y}, d)\}$ 
9:     else if  $y_d^* \notin \text{top-}k(\text{logits}_\theta(\cdot \mid x, \hat{y}_{<d}))$  then ▷ Class 2
10:       $D_{\text{forget}} \leftarrow D_{\text{forget}} \cup \{(x, \hat{y}, d)\}$ 
11:     else ▷ Class 3
12:        $\hat{w}, w^* \leftarrow$  predicted / gold first word at  $d$ 
13:       if  $\cos(\text{emb}(\hat{w}), \text{emb}(w^*)) < \tau$  then
14:          $D_{\text{forget}} \leftarrow D_{\text{forget}} \cup \{(x, \hat{y}, d)\}$ 
15:       end if
16:     end if
17:   end if
18: end for
19: return  $D_{\text{forget}}$ 
```

enhancement address different failure modes.

Rethinking Schema Linking. Rethinking Schema Linking (Nahid et al., 2025) treats schema linking as a retrieval problem. It uses a context-aware bidirectional retrieval framework that combines table-first and column-first schema retrieval. In our experiments, this method is used as an input preprocessing baseline. It prunes irrelevant schema elements before SQL generation, reducing schema noise in the model input.

DIVER. DIVER (Nan et al., 2026) is an input-side enhancement method designed for robust Text-to-SQL generation with dynamic interactive value linking and evidence reasoning. It probes the database to construct evidence for resolving schema-value ambiguity. In our experiments, DIVER is used to improve the grounding information provided to the SQL generator before decoding.

SEA-SQL. SEA-SQL (Li et al., 2024a) performs semantic-enhanced SQL refinement with adaptive bias elimination and dynamic execution adjustment. It is used as an output post-processing baseline. Given an initially generated SQL query, SEA-SQL refines the query to reduce output-level errors.

SHARE. SHARE (Qu et al., 2025) is a hierarchical SQL correction framework based on special-

ized small language models. It decomposes SQL correction into stepwise actions and performs fine-grained refinement over the generated query. In our experiments, SHARE is treated as an output correction method applied after the SQL generator produces its initial prediction.

MBR-SQL. MBR-SQL (Borchmann and Wydmuch, 2025) represents multi-candidate selection. It generates multiple SQL candidates and selects the final prediction using execution-guided Minimum Bayes Risk decoding. Following the main text, we use $N = 10$ candidates. This baseline improves output selection rather than repairing the model parameters.

C.3 Combination Protocol

For each inference-time enhancement method, we evaluate two settings. The first setting applies the inference-time method to the base model. The second setting applies the same inference-time method to the TTU-SQL-updated model. This protocol tests whether parameter-level repair is complementary to inference-time improvement.

The combination procedure depends on the method type. For input preprocessing methods, including Rethinking Schema Linking and DIVER, the preprocessing module is applied before SQL generation. The pruned schema, linked values, or

database evidence are inserted into the model input, and the model then generates SQL using the same decoding setup as in the main experiments. For output post-processing methods, including SEA-SQL and SHARE, the model first generates an initial SQL query, and the correction module revises the output. The corrected SQL is used as the final prediction. For MBR-SQL, the model generates $N = 10$ SQL candidates, and the final prediction is selected using the method’s execution-guided selection criterion.

All final predictions are evaluated using the same execution-accuracy protocol described in Appendix A.3. The target evaluation sets are used only for evaluation. They are not used for constructing forget sets, tuning unlearning targets, or updating model parameters. Therefore, when an inference-time method is combined with TTU-SQL, the main difference from the base-model setting is that the underlying SQL generator has been repaired by TTU-SQL before inference.

Per-setting composition results. Table 5 reports the per-setting execution accuracy when each inference-time enhancement method is applied to the base model and to the TTU-SQL-repaired model, complementing the macro-average comparison in Table 1 (Panel 2).

D Additional Results and Analyses

D.1 Forget-Class Combination

Table 6 expands the forget-class ablation in the main text by reporting the number of forget samples for each class combination. The sample counts are not additive because the three error-type criteria can overlap.

The results show that the three extracted error types provide complementary unlearning targets. Among single-class settings, Class 2 achieves the best result at 64.50% EX, suggesting that errors where the gold token is not locally competitive are particularly amenable to suppress-then-recover optimization. Class 1 and Class 3 also improve over weaker variants, but neither alone reaches the full method. Among pairwise combinations, Class 2+3 performs best at 64.89%, indicating that low-preference gold tokens and wrong-preference errors capture related but non-identical local failure modes. Combining all three classes yields the best result, 65.57%, even though the total sample count increases only slightly over Class 2+3. This suggests that the structural divergence cases in Class 1

still contribute useful repair signals when combined with the other two classes.

D.2 Effect of Retain and Forget Set Size

Table 9 reports the sensitivity of TTU-SQL to retain-set and forget-set size. For the retain set, using only 2,000 examples yields 63.44% EX, which is substantially below the full setting. This indicates that a small retain set is insufficient to stabilize the unlearning stage. Increasing the retain set generally improves performance, with the largest retain setting reaching 65.57%. The non-monotonic result at 10,000 examples suggests that retain sampling quality also matters, but the overall trend supports using enough correct examples to preserve general SQL generation behavior during Part-NPO.

For the forget set, the best result is obtained with 10,000 forget samples. Smaller forget sets under-cover recurring error patterns, while larger forget sets do not further improve performance. In particular, increasing the forget set to 15,000 or 20,000 slightly reduces EX. This suggests that adding more forget samples can introduce less localizable or noisier errors, which is consistent with the main ablation result that indiscriminately applying NPO to all errors is less effective than targeted forget-set construction.

D.3 Probe-Set Size Sensitivity

Table 7 reports TTU-SQL under equal-budget probe sets of different sizes on Spider with DeepSeek-R1-Distill-Qwen-7B. The benefit remains under substantially smaller probe sets, although absolute accuracy increases with additional supervision.

D.4 Effect of Similarity Threshold τ

The Class 3 extraction rule uses a cosine-similarity threshold τ to select cases where the gold token appears among the top-5 candidates but is still semantically distant from the predicted token. Table 8 shows that increasing τ monotonically increases the number of Class 3 samples and the total forget-set size. However, performance does not improve monotonically.

A low threshold such as $\tau = 0.6$ selects only 8,842 Class 3 samples and reaches 64.61% EX, suggesting that the forget set under-covers useful wrong-preference errors. Increasing the threshold to 0.7 improves EX to 65.18%. The best result is achieved at $\tau = 0.8$, where the model obtains

Inference-time method	Applied to	Llama-Spider	Llama-BIRD	Qwen-Spider	Qwen-BIRD
Rethinking Schema Linking	Base + method	63.34	34.29	61.70	35.14
	TTU-SQL + method	69.73	36.57	67.99	37.35
DIVER	Base + method	62.19	35.85	59.96	36.44
	TTU-SQL + method	67.70	37.09	65.37	37.87
SEA-SQL	Base + method	70.99	39.11	69.73	38.66
	TTU-SQL + method	72.53	39.83	71.66	39.70
SHARE	Base + method	68.96	37.29	67.60	37.16
	TTU-SQL + method	71.47	38.85	68.86	38.20
MBR-SQL (N=10)	Base + method	61.80	33.18	58.99	32.01
	TTU-SQL + method	69.05	35.52	65.76	35.14

Table 5: Per-setting execution accuracy of composition with inference-time enhancement methods on Spider and BIRD Dev (EX, %). Base and TTU-SQL rows are reported per setting; Direct SFT + method is reported as a macro-average in Table 1 (Panel 2).

Forget set	Samples	EX (%)
Class 1 only	3,811	64.12
Class 2 only	6,551	64.50
Class 3 only	12,027	63.44
Class 1+2	9,545	64.22
Class 1+3	12,955	63.64
Class 2+3	18,578	64.89
Class 1+2+3	18,689	65.57

Table 6: Effect of forget-class combinations on Spider with DeepSeek-R1-Distill-Qwen-7B.

Probe size	Direct SFT	TTU-SQL	Gain
5K	59.10	60.30	+1.20
10K	61.17	63.04	+1.87
25K	61.96	64.52	+2.56
50K	62.48	65.57	+3.09

Table 7: Probe-set size sensitivity on Spider with DeepSeek-R1-Distill-Qwen-7B under equal budgets.

65.57% EX. Further increasing the threshold to 0.9 expands the forget set to 21,846 samples but reduces EX to 65.09%. This indicates that a threshold that is too high may include less clean or weakly related samples, introducing noise into the unlearning target. We therefore use $\tau = 0.8$ in the main experiments.

D.5 Effect of the Top- k Criterion

The Class 2 extraction rule checks whether the gold token appears among the model’s top- k logit-ranked candidates at the divergence position. Table 9 reports the sensitivity of TTU-SQL to k on Spider with DeepSeek-R1-Distill-Qwen-7B. Performance is stable across $k \in \{3, 5, 10\}$, with $k = 5$ giving the best result; we therefore use $k = 5$ in the main experiments.

τ	Class 3	Forget	EX (%)
0.6	8,842	16,376	64.61
0.7	10,485	17,927	65.18
0.8	12,027	18,689	65.57
0.9	15,914	21,846	65.09

Table 8: Effect of the similarity threshold τ for Class 3 extraction.

D.6 Optimization Trajectory Raw Results

Figure 4 in the main text compares the optimization trajectories of Direct SFT and TTU-SQL. Table 10 reports the raw epoch-wise execution accuracy values used to generate the figure.

The trajectory provides a more detailed view of the suppress-then-recover process. After the first epoch, both Direct SFT and the TTU-SQL trajectory are at 53.48% EX, indicating that the initial probe adaptation alone does not directly improve target-set performance. After Part-NPO, performance drops to 44.20%, which is expected because the unlearning stage suppresses localized erroneous continuations and is not intended to be a final model update. During recovery SFT, TTU-SQL rapidly recovers and surpasses Direct SFT from recovery epoch 4 onward. Direct SFT reaches its best checkpoint at 62.96%, whereas TTU-SQL reaches 65.76%, showing that the advantage of TTU-SQL comes from the optimization trajectory rather than additional SFT epochs alone.

D.7 Error Pattern Distribution Analysis

Table 11 reports the distribution of the extracted error types across Spider SQL complexity levels. All three error types are concentrated in harder SQL examples. For Class 1, 70.0% of the samples come

Setting	EX (%)
TTU-SQL	65.57
Retain set size	
2,000	63.44
5,000	64.51
10,000	63.73
15,000	64.70
18,578	65.57
Forget set size	
2,000	64.89
5,000	64.99
10,000	65.57
15,000	64.80
20,000	64.70
Top-k criterion	
$k = 3$	65.10
$k = 5$	65.57
$k = 10$	65.31

Table 9: Sensitivity analysis for retain-set and forget-set size and the top- k criterion on Spider with DeepSeek-R1-Distill-Qwen-7B.

Checkpoint	Direct SFT	TTU-SQL
Base model	58.51	58.51
Epoch 1 / before Part-NPO	53.48	53.48
After Part-NPO	—	44.20
Recovery epoch 2	60.64	60.15
Recovery epoch 3	60.93	61.61
Recovery epoch 4	61.99	62.86
Recovery epoch 5	62.38	64.41
Recovery epoch 6	62.28	64.60
Recovery epoch 7	62.96	63.25
Recovery epoch 8	62.48	65.57
Recovery epoch 9	62.48	65.76
Recovery epoch 10	62.38	64.60

Table 10: Raw epoch-wise execution accuracy used for the optimization trajectory figure.

from extra-hard queries and 90.0% come from hard or extra-hard queries. For Class 2, 69.7% are extra-hard and 88.3% are hard or extra-hard. For Class 3, 69.2% are extra-hard and 86.7% are hard or extra-hard. This pattern suggests that localizable erroneous continuations occur most frequently in structurally complex SQL programs, where the model must make more clause-level and operator-level decisions.

Figure 6 further shows that divergence positions are heavily skewed toward the early part of the generated SQL sequence. The first 0–10% bin dominates the distribution, followed by a long tail over later positions. This supports the design choice of localizing the first divergence position: many repairable errors emerge early and can affect the remaining generation trajectory if not corrected.

Error Type	Easy	Medium	Hard	Extra
Class 1	32	350	761	2668
Class 2	96	672	1220	4563
Class 3	156	1440	2105	8326

Table 11: Distribution of extracted error types across Spider SQL complexity levels.

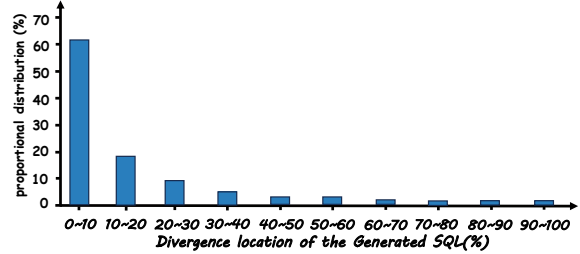


Figure 6: Distribution of normalized divergence positions within generated SQL sequences.

D.8 Quantitative Support for the Key Observations

This section provides quantitative support for the three observations in Section 3.

O1: Errors recur around shared decision signatures. Among 22,156 erroneous probe predictions in the Spider/DeepSeek-R1-Distill-Qwen-7B setting, 73.4% belong to an AST-divergence signature occurring at least five times, and the 20 most frequent signatures account for 41.8% of errors. Comparable rates across the other three model-dataset settings are 69.1–75.6%.

O2: Suppressing a dominant wrong continuation can expose a correct behavior. In this subset, 45 of 495 previously incorrect predictions become execution-correct after a simple NPO step; the wrong-token logit decreases substantially and the median rank of the gold token improves, indicating a local preference shift rather than random output variation.

O3: Coarse suppression is more destructive. Under identical data and optimization, Full-NPO obtains 64.22 EX, suffix-level Part-NPO obtains 64.70, and first-token Part-NPO obtains 65.57. The corresponding post-recovery invalid-SQL rates are 4.9%, 4.2%, and 3.5%.

D.9 Category-Level Repair Analysis

Table 12 reports a before/after analysis on Spider Dev with DeepSeek-R1-Distill-Qwen-7B, using the same AST-based first-divergence categorizer as

First-div. category	Base	Direct SFT	TTU-SQL	Δ vs. Direct SFT
Column selection	76	69	61	-11.6%
Structural operator	72	64	56	-12.5%
Aggregation	60	53	45	-15.1%
Grouping/ordering	58	51	44	-13.7%
Condition/value	53	49	48	-2.0%
Join/table selection	53	49	47	-4.1%
Other/non-local	55	53	55	+3.8%

Table 12: Before/after error counts by first-divergence category on Spider Dev (DeepSeek-R1-Distill-Qwen-7B), over 427 categorizable base-model errors. Δ is the relative reduction of TTU-SQL vs. Direct SFT.

Figure 1(a). Counts are over the 427 base-model errors that can be assigned to a category. The largest reductions occur in the local generation categories explicitly targeted by TTU-SQL (column selection, structural operator, aggregation, grouping/ordering), whereas condition/value grounding and non-local errors change little, delineating the scope of token-level repair.

E Theoretical Illustration

E.1 A Toy View of Why SFT Can Be Slow for Low-Probability Tokens

This section provides a simplified logit-space illustration. It is not intended as a complete optimization theory, but as an explanatory model for why low-probability gold tokens can be difficult to recover through limited-step SFT.

Let g denote the gold token and w denote the dominant wrong token at a divergence position. For cross-entropy training, the gradients with respect to the logits are:

$$\frac{\partial \mathcal{L}}{\partial z_g} = p_g - 1, \quad \frac{\partial \mathcal{L}}{\partial z_w} = p_w.$$

Suppose the gold token has probability $p_g = 0.01$, while the wrong token has probability $p_w = 0.80$. The initial logit gap is:

$$z_g - z_w = \log \frac{p_g}{p_w} = \log \frac{0.01}{0.80} \approx -4.38.$$

Under a simplified first-order approximation, the per-step change in the logit difference is:

$$\Delta(z_g - z_w) \approx (1 - p_g + p_w)\eta \approx 1.79\eta.$$

The approximate number of SFT steps required to flip the decision boundary is:

$$N = \frac{4.38}{1.79\eta}.$$

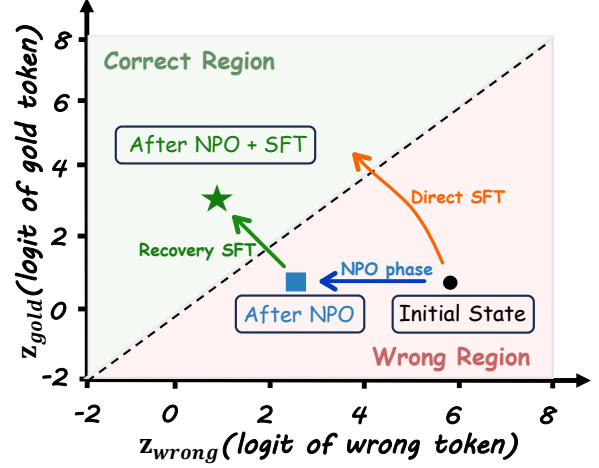


Figure 7: Toy logit-space trajectory. Direct SFT slowly moves toward the decision boundary, while NPO first suppresses the wrong token and recovery SFT crosses the boundary from a smaller local gap.

This toy calculation illustrates why Class 2 samples can be difficult for limited-step SFT: the gold token starts outside the top-ranked candidates, and the model must overcome a large local logit gap before the correct token becomes competitive.

E.2 How NPO Can Shorten the Subsequent SFT Path

NPO directly suppresses the wrong continuation. If NPO reduces p_w from 0.80 to 0.20, the local logit gap becomes:

$$z_g - z_w : -4.38 \xrightarrow{\text{NPO}} -0.69.$$

The approximate number of subsequent SFT steps becomes:

$$N_{\text{new}} = \frac{0.69}{1.79\eta}.$$

The key intuition is that NPO does not itself learn the correct SQL. Instead, it reduces the local dominance of an erroneous continuation, making subsequent SFT operate from a more favorable model state.

E.3 Conceptual Comparison with DPO

DPO (Rafailov et al., 2023) jointly increases the relative likelihood of a chosen output and decreases that of a rejected output. In the Text-to-SQL setting, this means that a gold SQL query and an erroneous prediction can be treated as a chosen-rejected pair. However, this objective couples two effects: increasing the probability of the gold query

Method	Recovery SFT	EX (%)
DPO-only	No	35.49
DPO + recovery SFT	Yes	62.96
NPO-only	No	44.20
TTU-SQL	Yes	65.57

Table 13: Objective-level comparison on Spider with DeepSeek-R1-Distill-Qwen-7B.

Field	Content
Question	What is the total number of singers?
Database	concert_singer
Before NPO	SELECT count(*) FROM singer GROUP BY Name
After NPO	SELECT count(*) FROM singer
Error type	Structural divergence
Divergence token	GROUP
Outcome	Correct after suppressing the local erroneous continuation

Table 14: A successful local repair case.

and suppressing the erroneous query. When the probe distribution differs from the target evaluation distribution, this coupled update may over-emphasize probe-specific SQL realizations.

In contrast, the TTU-SQL training procedure decouples suppression and recovery: $\mathcal{L}_{\text{NPO}}$ suppresses $\pi_\theta(\hat{y}|x)$, while $\mathcal{L}_{\text{SFT}}$ recovers SQL generation on D_P . This separation allows the unlearning stage to focus on weakening localized wrong continuations, while the recovery stage restores general SQL generation ability.

Empirical comparison. Table 13 compares DPO and NPO-based objectives under the same model setting. DPO-only obtains 35.49% EX, indicating that directly applying pairwise preference optimization is unstable in this setting. Adding recovery SFT improves DPO to 62.96%, but it remains below TTU-SQL at 65.57%. NPO-only obtains 44.20%, confirming that unlearning alone is not sufficient as a final update. The best result is achieved by TTU-SQL, which combines localized unlearning with recovery SFT. These results support the suppress-then-recover design rather than a coupled preference-optimization objective.

F Case Studies

F.1 A Successful Local Repair

Table 14 shows a case where Part-NPO removes an unnecessary structural continuation while preserving the correct prefix.

Field	Content
Question	Find names of customers with orders for both product A and B.
Before NPO	SELECT ... WHERE o.product = 'A' AND o.product = 'B'
After NPO	Still wrong.
Gold behavior	Requires an INTERSECT structure or a double-JOIN formulation.
Failure type	Global SQL planning.
Outcome	Not resolved by token-level unlearning alone.

Table 15: A failure case requiring global query planning rather than local token-level repair.

In this case, the model incorrectly adds a GROUP BY clause after a counting query. The divergent token corresponds to a local structural decision. Suppressing this continuation removes the unnecessary grouping behavior while preserving the correct prefix SELECT count(*) FROM singer.

F.2 A Failure Case Requiring Global Planning

Table 15 shows a failure case where the required correction is not a local token-level preference change. Instead, the example requires global SQL planning.

This error is not primarily a single-token local preference issue. It requires global SQL planning and compositional reasoning over two simultaneous conditions. Such cases exceed the intended scope of token-level unlearning and motivate future work on structure-level repair.