Beyond Static Rules: Automated Discovery of Latent Vulnerabilities in Text-to-SQL

Hanqing Wang[1*], **Yongdong Chi**[1*], **Jian Yang**[2],
Lei Yang[3], **Jiehui Zhao**[3], **Yun Chen**[1], **Guanhua Chen**[4†]
[1]Shanghai University of Finance and Economics, [2]Beihang University,
[3]Deepexi Technology Co. Ltd., [4]Southern University of Science and Technology

Abstract

While Large Language Models (LLMs) have achieved remarkable success in Text-to-SQL tasks, their deployment in real-world environments is hindered by latent reliability issues. Identifying these latent weaknesses is critical for building trustworthy database interfaces, yet current diagnostic approaches rely heavily on static, expert-defined rules, which lack the capability for systematic and automated exploration. To bridge this gap, we propose SAGE (Systematic Automated Guided Exploration), a novel framework designed to autonomously uncover latent failure patterns in LLM-based Text-to-SQL generation. Specifically, SAGE generates vulnerability hypotheses for given samples and references a continuously evolving *Vulnerability Codex* to design targeted perturbations, thereby iteratively verifying and documenting potential defects. Extensive experiments on state-of-the-art open-source LLMs demonstrate that SAGE uncovers a substantial number of failure cases, highlighting the significant fragility of current models. Furthermore, our analysis reveals that the Vulnerability Codex exhibits strong cross-model transferability, indicating that the discovered patterns represent generalized structural weaknesses. Finally, we explore SAGE's potential for remediation. Furthermore, a preliminary attempt at lightweight fine-tuning on the generated samples yields promising improvements, suggesting a scalable pathway for closing the reliability loop in future work.

1 Introduction

Text-to-SQL systems (Androutsopoulos et al., 1995; Li and Jagadish, 2014; Li et al., 2024b; Yu et al., 2018) serve as a vital bridge between natural language and structured data, democratizing data access by enabling non-technical users to query complex databases intuitively. With the recent integration of Large Language Models (LLMs) (Pourreza and Rafiei, 2023; Xie et al., 2025; Li et al., 2023a), this field has witnessed a paradigm shift in semantic understanding and execution accuracy.

Systematically identifying latent failure modes is the prerequisite for enhancing this reliability. However, current evaluation paradigms predominantly rely on static benchmarks (e.g., Spider (Yu et al., 2018), BIRD (Li et al., 2023b)) or manual error analysis (Chang et al., 2023a; Gan et al., 2021b,a; Patil et al., 2023; Pi et al., 2022).

These approaches are inherently constrained by their reliance on human priors: they primarily verify model robustness against predefined challenges (Zhang et al., 2024; Deng et al., 2021; Bhaskar et al., 2023), but cannot provide a systematic exploration of potential or previously unidentified vulnerabilities[1]. As a result, certain failure modes (Saparina and Lapata, 2024; Sahitaj et al., 2025; Chang et al., 2023b) may remain insufficiently examined and can have a substantial impact on the reliability of Text-to-SQL generation beyond what is captured by fixed heuristics. Consequently, there is a pressing need for an automated framework capable of proactively exploring the model's failure boundaries.

To bridge this gap, we propose SAGE (**S**ystematic **A**utomated **G**uided **E**xploration), a framework designed to proactively expose latent vulnerabilities in LLM-based Text-to-SQL generation. Unlike passive verification methods (Chang et al., 2023a; Gan et al., 2021b,a; Patil et al., 2023; Pi et al., 2022), SAGE orchestrates an active discovery process. Specifically, it first formulates specific vulnerability hypotheses for a target sample, and then references a continuously evolving

[*] Equal Contribution.
[†] Corresponding Author.

[1]Throughout this paper, "vulnerability" refers to robustness and reliability brittleness under semantic-preserving perturbations, rather than database security exploits such as SQL injection.

Findings of the Association for Computational Linguistics: ACL 2026, pages 17065–17082

Vulnerability Codex, a dynamic repository of generalized error archetypes, to synthesize targeted perturbed samples. This allows SAGE to systematically verify potential weaknesses and iteratively refine its understanding of the model's failure boundaries, transforming the discovery of vulnerabilities from a static manual task into an autonomous, self-improving cycle.

Our main contributions are as follows:

- We propose SAGE, a framework that autonomously evolves a *Vulnerability Codex* to guide exploration. It outperforms static expert-curated baselines, achieving average Vulnerability Exposure Rates (VER) margins of 26.33% and 18.66% on BIRD and Spider across different models, respectively. These results indicate the effectiveness of our dynamic, automated strategy compared to fixed manual heuristics.
- We conduct a systematic vulnerability assessment of LLMs in Text-to-SQL. Our evaluation across diverse models on BIRD and Spider yields average VER of 76.98% and 58.45%, respectively. These figures provide empirical evidence of the prevalence of latent vulnerabilities within current Text-to-SQL systems.
- Beyond detection, we take an initial step towards establishing a reliability loop. Lightweight fine-tuning on 1.5k samples partially alleviates the model's defects. These findings, though preliminary, shed light on a scalable pathway for model improvement and validate the potential of using SAGE for vulnerability remediation.[2]

2 Related Work

2.1 LLM-based Text-to-SQL Generation

With the rapid advancement of LLMs, LLM-based approaches to Text-to-SQL have demonstrated significantly enhanced transferability and reasoning capabilities compared to traditional rule-based systems (Mahmud et al., 2015; Lyu et al., 2020), neural models (Choi et al., 2021; Xu et al., 2017), and pre-trained models (Li et al., 2023a; Yin et al., 2020), propelling the field into a new stage (Hong et al., 2024; Shi et al., 2025). Current methodologies primarily follow two paradigms: prompt engineering and fine-tuning. Prompt engineering methods, typically utilizing closed-source LLMs, enhance generation quality through techniques such as task decomposition (Pourreza and Rafiei, 2023; Xie et al., 2025; Chi et al., 2025), chain-of-thought (CoT) reasoning (Xu et al., 2024; Liu et al., 2025), and multi-agent collaboration (Xia et al., 2024; Deng et al., 2025). Conversely, fine-tuning methods leverage open-source LLMs to reduce inference costs while improving privacy, controllability, and stability (Li et al., 2025; Qu et al., 2025; Guo et al., 2025; Li et al., 2024a; Sheng et al., 2025; Cheng et al., 2025).

Despite their success, the primary objective of these works is to maximize execution accuracy, optimizing the probability of generating correct SQL queries. A specific line of research within this domain—automated self-correction—shares a superficial resemblance to our work by identifying and fixing errors (Shen et al., 2025; Gong et al., 2025; Askari et al., 2025). However, their fundamental goals and mechanisms differ significantly. These methods employ correction solely as an intermediate step to boost performance on specific samples, often addressing errors in an ad-hoc manner without analyzing the underlying causes (Liu and Tan, 2024; Pourreza and Rafiei, 2023). In contrast, our work shifts the focus from *performance enhancement* to *systematic vulnerability discovery*. We aim not merely to correct individual instances but to abstract high-level failure patterns and diagnose latent defects in the model's reasoning process, providing generalized insights that go beyond instance-level repair.

2.2 Robustness and Brittleness in LLM-based Text-to-SQL Generation

Prompt-engineering and fine-tuning strategies have continuously advanced Text-to-SQL performance on foundational benchmarks such as Spider (Yu et al., 2018) and BIRD (Li et al., 2023b).

However, existing approaches to evaluating model reliability are largely limited to confirming the presence of known failure patterns under specific conditions. Researchers have established robustness benchmarks by applying semantic-preserving perturbations (Deng et al., 2021; Gan et al., 2021b,a; Sahitaj et al., 2025) or predefined rule-based alterations to schemas and queries (Chang et al., 2023b; Patil et al., 2023). While valuable, these methods operate within a closed scope defined by human experts—they

[2]Our code is available at https://github.com/sustech-nlp/SAGE.

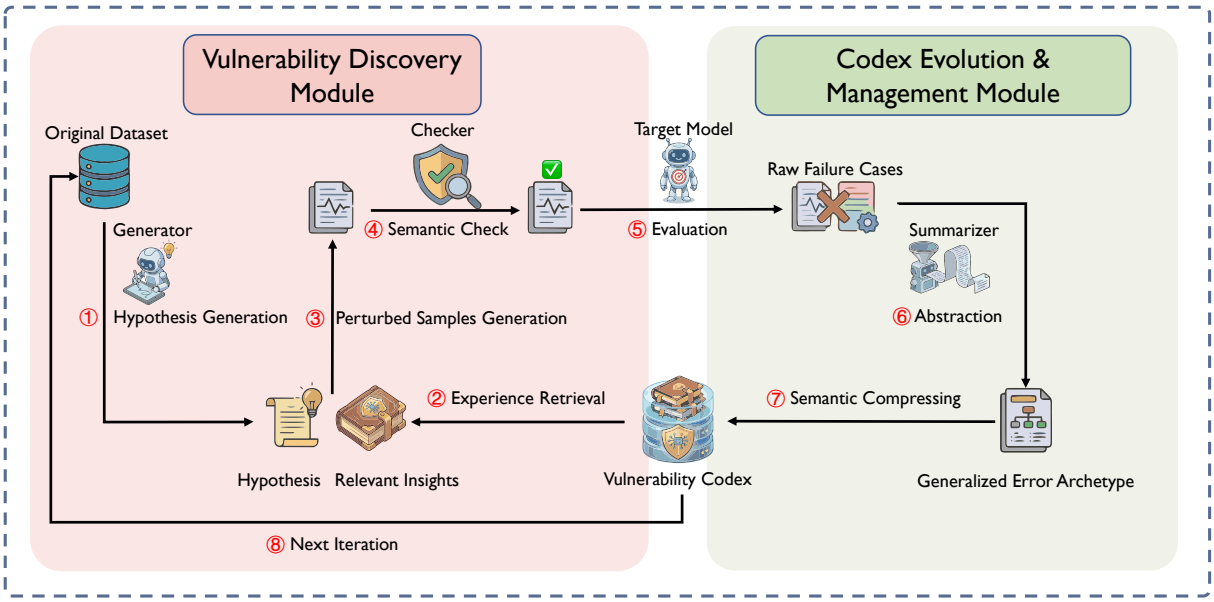

Figure 1: The architecture of the proposed automated vulnerability discovery framework, SAGE. The process iterates between two stages: (1) Discovery, where the system utilizes existing insights from the central Vulnerability Codex to generate and verify hard adversarial samples (Steps 1-5); and (2) Evolution, where identified failures are abstracted into generalized archetypes and merged back into the Codex through semantic compression (Steps 6-7). This closed-loop mechanism allows the system to continuously update its strategy in subsequent iterations (Step 8).

merely verify whether an LLM succumbs to anticipated error types, such as structural ambiguity (Ding et al., 2025; Bhaskar et al., 2023), but fail to explore the model's "blind spots" outside these heuristics (Saparina and Lapata, 2024; Pi et al., 2022; Zhang et al., 2024). Although recent efforts pursue automated bias or failure discovery in other LLM settings (Lai et al., 2026), systematic vulnerability exploration for Text-to-SQL remains largely open.

This highlights the need for an efficient, self-evolving, and automated framework capable of discovering latent failure patterns. Unlike previous works, our work aims to systematically expose unknown defect patterns, providing actionable insights for the fundamental robustness of Text-to-SQL systems.

3 Method

To systematically uncover latent vulnerabilities in Text-to-SQL systems, we propose SAGE, an iterative framework designed to autonomously discover and evolve vulnerability insights. As illustrated in Figure 1, SAGE operates as a closed-loop system structured around two complementary modules: the **Vulnerability Discovery Module** (Section 3.2), which actively probes the target model through hypothesis-driven perturbations, and the **Codex Evolution & Management Module** (Section 3.3), which abstracts validated failures into a self-refining *Vulnerability Codex*. The detailed algorithmic procedure is provided in Algorithm 1.

3.1 Problem Formulation

We formalize the problem of automatic vulnerability discovery within the Text-to-SQL paradigm. Let $D = \{(q_i, s_i, y_i^*)\}_{i=1}^N$ denote a dataset where q_i is the natural language query, s_i is the database schema, and y_i^* is the ground-truth execution result, on which the target model M_t performs correctly under standard conditions, i.e., $\text{Exec}(M_t(q_i, s_i)) = y_i^*$. This selection ensures that any subsequent failure induced by our framework reflects a genuine blind spot or fragility in the model rather than an inherent lack of reasoning ability.

Our objective is to construct and evolve a **Vulnerability Codex**, denoted as $\mathcal{V}$, which systematically captures the latent failure modes of M_t. We define SAGE as an iterative process governed by two mapping functions, $\langle \Phi_{\text{disc}}, \Phi_{\text{evol}} \rangle$, operating over iterations $t = 1, \ldots, T$.

Let $\mathcal{V}_{t-1}$ denote the Codex state from the previous iteration. At each step t:

Discovery (Φ_{disc}): The system synthesizes hypotheses and constructs specific perturbed samples

to probe M_t, yielding a set of raw failure cases $\mathcal{F}_t$ where M_t exposes vulnerability:

$$\mathcal{F}_t = \Phi_{\text{disc}}(D, \mathcal{V}_{t-1}) \quad (1)$$

Evolution (Φ_{evol}): These raw failures are abstracted, compressed, and integrated into the Codex to produce an updated state $\mathcal{V}_t$:

$$\mathcal{V}_t = \Phi_{\text{evol}}(\mathcal{V}_{t-1}, \mathcal{F}_t) \quad (2)$$

The process continues until the Codex stabilizes or the maximum iteration budget is reached.

3.2 Vulnerability Discovery Module

This module executes the active probing phase of our framework. It transforms abstract vulnerability hypotheses into concrete, context-aware adversarial samples to expose latent defects. This process follows a coarse-to-fine pipeline: from *Hypotheses* to *Scopes*, and finally to *Sample Generation*.

3.2.1 Systematic Hypothesis Generation

Diverging from stochastic perturbation methods, we initiate discovery by synthesizing targeted **Vulnerability Hypotheses**. A Generator Agent (M_g) analyzes each data sample $d_i \in D$ to identify potential vulnerabilities (e.g., "Ambiguity in column selection" or "Misinterpretation of complex nested logic"). This yields a set of candidates $\mathcal{C}_{curr} = \bigcup_{d_i \in D} \{(d_i, h_{i,k})\}_{k=1}^{K}$, where each $h_{i,k}$ represents a specific conceptual strategy suspected to induce failure (step 1 in Figure 1).

To ensure structural grounding, these hypotheses are mapped to specific **Perturbation Scopes** $\mathbb{S}$. Specifically, we categorize $\mathbb{S}$ into three distinct semantic dimensions: (1) *natural language queries*, (2) *schema-irrelevant elements*, and (3) *schema-relevant elements*. In the initial cold-start phase ($t = 1$), to balance exploration coverage and efficiency, we deterministically assign each generated hypothesis to a single scope via a round-robin allocation (detailed in Algorithm 1). In subsequent iterations, a comprehensive scan is performed by applying hypotheses across all available scopes.

3.2.2 Experience-Guided Probing & Verification

To rigorously verify whether a candidate hypothesis $C_j = (d_i, h_{i,k})$ exposes actual vulnerabilities, SAGE references the *Vulnerability Codex* to guide the generation process. We employ an Embedding Model (M_e) to retrieve relevant insights E_j from $\mathcal{V}_{t-1}$. This serves as a reference for implementation feasibility rather than content limitation, ensuring that the verification focuses on the hypothesis's validity (step 2 in Figure 1).

$$E_j = \text{RetrieveTop5}(M_e, h_{i,k}, \mathcal{V}_{t-1}) \quad (3)$$

Guided by the candidate C_j, the target scope $s_m \in \mathbb{S}$, and the retrieved experience E_j, the Generator M_g constructs a perturbed sample $x_{j,m}$ (step 3 in Figure 1):

$$x_{j,m} = \text{SampleGenerate}(M_g, C_j, s_m, E_j) \quad (4)$$

To ensure $x_{j,m}$ preserves the original semantic logic (i.e., the ground truth execution result remains invariant), we introduce a Checker Agent (M_c) to validate the qualification of $x_{j,m}$ (step 4 in Figure 1).

$$\text{IsValid} = \text{ValidCheck}(M_c, C_j, x_{j,m}) \quad (5)$$

Given the complexity of semantic verification, we validate the Checker Agent's reliability through both human evaluation and cross-model proxy validation (see Appendix B.5).

Upon validation, we probe the target model M_t with $x_{j,m}$ to obtain the predicted SQL execution result $y_{j,m}$. A raw vulnerability is identified if the perturbation is valid, but the model execution deviates from the ground truth (step 5 in Figure 1):

$$\text{IsCorrect} = (\text{Exec}(M_t(x_{j,m})) == y_i^*) \quad (6)$$

If IsCorrect is false, the sample reveals a genuine vulnerability, triggering the evolution phase.

3.3 Codex Evolution & Management Module

While the Discovery Module exposes individual failures, the Evolution Module functions as the system's cognitive repository, responsible for distilling knowledge. It converts noisy, instance-specific failures into a high-quality, compact **Vulnerability Codex** ($\mathcal{V}$). This involves two key stages: *Abstraction* and *Refinement*.

3.3.1 Abstraction via Summarization

Raw failure cases are often coupled with specific schema entities or query phrasings. To foster generalization, a Summarizer Agent (M_s) analyzes the discovered failure tuple $(C_j, x_{j,m}, y_{j,m})$ (step 6 in

Figure 1). It abstracts the concrete error into a structured **Generalized Error Archetype** v_j, which encapsulates a generalized error summary and causal analysis.

$$v_j = \text{VulnerAbstraction}(M_s, C_j, x_{j,m}, y_{j,m}) \tag{7}$$

These abstracted archetypes are collected into a temporary set $\mathcal{V}_{\text{new}}^{(t)}$:

$$\mathcal{V}_{\text{new}}^{(t)} = \mathcal{V}_{\text{new}}^{(t)} \cup \{v_j\} \tag{8}$$

3.3.2 Refinement via Semantic Compression

As the iterative process proceeds, simply accumulating archetypes in $\mathcal{V}_{\text{new}}^{(t)}$ may lead to a redundant repository. To address this, we introduce a **Semantic Compression** mechanism during the construction of $\mathcal{V}_t$ (Phase 3 in Algorithm 1).

At the conclusion of each iteration, newly discovered archetypes are merged with the existing Codex (step 7 in Figure 1). We perform semantic clustering using M_e: if the Euclidean distance between the embeddings of two vulnerability entries falls below a similarity threshold τ, they are identified as redundant and compressed.

Formally, this refinement updates the Codex by:

$$\mathcal{V}_t = \text{SemanticComp}(M_e, \mathcal{V}_{t-1} \cup \mathcal{V}_{\text{new}}^{(t)}, \tau) \tag{9}$$

This refinement step ensures that the Vulnerability Codex $\mathcal{V}$ remains compact and distinct, improving retrieval efficiency and enabling the system to converge towards a representative set of failure modes on Text-to-SQL generation.

4 Experiment

4.1 Setup

Datasets We evaluate SAGE using the *Spider* (Yu et al., 2018) and *BIRD* (Li et al., 2023b) datasets. To rigorously benchmark vulnerability, we define our evaluation set as the specific subset of samples where the target model yields the correct ground-truth execution result, filtering out instances of initial failure. The exact cardinalities of this initially-correct subset $|D|$ for each model/-dataset pair are reported in Appendix Table 6.

Models For our main experimental evaluations, we select three representative models that span distinct paradigms as our target models: a general-purpose model, Gemma-3 (Team et al., 2025) (*Gemma-3-12B-it*); a domain-adapted model, OmniSQL (Li et al., 2025) (*OmniSQL-32B*); and a reinforcement-learned model, Inf-rl-qwen (Infly and InfTech, 2025) (*inf-rl-qwen-coder-32B-2746*). Additionally, we demonstrate the applicability of our approach to powerful proprietary models by evaluating on *GPT-4o* (Hurst et al., 2024) (see Appendix A.2). For fine-tuning experiments in Section 5.2, we use Qwen2.5-Coder(*Qwen2.5-Coder-7B-Instruct*), considering training cost.

Implementation Details We initialize the Vulnerability Codex using a repository of expert-defined vulnerabilities curated from established studies, including Dr.Spider, SpiderSyn, and ADVETA (Chang et al., 2023b; Gan et al., 2021a; Pi et al., 2022). We employ **Qwen3-32B** (Yang et al., 2025) as the backbone for the Generator, Checker, and Summarizer components, while **Qwen3-Embedding-4B** (Zhang et al., 2025) serves as the embedding model. Unless otherwise specified, we use $T = 3$ iterations, $K = 3$ initial hypotheses per sample, Top-5 Codex retrieval, and a semantic compression threshold $\tau = 0.1$; the search over a hypothesis stops once a valid failure is exposed or the maximum iteration budget is reached. We follow each model's default chat sampling settings during generation (for Qwen3-32B, $temperature = 0.6$, $top_k = 20$, and $top_p = 0.95$). A consolidated summary of these settings, together with the exact $|D|$ values, is provided in Appendix Table 6. Full details regarding the prompts used for sample perturbation are provided in Appendix B.4. Given the critical role of semantic verification, we provide a dedicated validation of the Checker module in Appendix B.5.

For fine-tuning experiments involving *Qwen2.5-Coder-7B-Instruct*, we utilize LoRA for parameter-efficient adaptation. Our training pipeline is built upon the `llama-factory`[3] framework. Comprehensive hyperparameter configurations are listed in Appendix Table 7.

Baselines To demonstrate the effectiveness of SAGE in discovering latent vulnerabilities on text-to-SQL generation, we introduce such baselines:

- **Original** Represents the standard performance of the target model on the unperturbed benchmarks (Spider and BIRD), serving as the reference for initial capability.
- **Expert** Due to the absence of direct baselines

[3] https://github.com/hiyouga/LlamaFactory

Model	Setting	BIRD				Spider			
		EX (↓)	**VES** (↓)	**VER** (↑)	**ApD** (↓)	**EX** (↓)	**VES** (↓)	**VER** (↑)	**ApD** (↓)
Gemma-3	Original	53.65	63.75	–	–	81.53	109.35	–	–
	Expert	24.64	29.38	54.07	7.73	48.36	61.83	52.16	10.95
	SAGE	**8.34**	**9.86**	**84.45**	**5.08**	**12.38**	**15.72**	**66.32**	**7.85**
Inf-rl-qwen	Original	70.53	87.03	–	–	87.72	116.38	–	–
	Expert	38.20	44.87	45.84	9.81	63.93	82.11	27.12	19.97
	SAGE	**21.90**	**26.03**	**68.95**	**7.27**	**56.19**	**71.05**	**48.07**	**12.93**
OmniSQL	Original	63.23	74.77	–	–	81.83	110.85	–	–
	Expert	30.31	36.63	52.06	7.09	48.84	63.57	40.10	10.60
	SAGE	**14.21**	**17.06**	**77.53**	**5.47**	**31.82**	**40.35**	**60.97**	**8.15**

Table 1: Main results on **BIRD** and **Spider** datasets. We report **EX** (Execution Accuracy), **VES** (Valid Efficiency Score), **VER** (Vulnerability Exposure Rate), and **ApD** (Attempts per Discovery). Arrows indicate whether higher (↑) or lower (↓) values are preferred. *Original* denotes the standard baseline performance. *Expert* and SAGE represent vulnerability probing methods via manual rules and our automated framework, respectively. Note that VER and ApD quantify the discovery capability and are not applicable (–) to the static Original baseline.

for automated vulnerability discovery in this specific context, we construct a strong baseline derived from prior work. This method executes vulnerability discovery using a fixed repository of expert-defined patterns (from Dr.Spider, SpiderSyn and ADVETA) without the dynamic update mechanism of our framework. This comparison is designed to highlight the advantage of evolving the Vulnerability Codex versus relying solely on static expert knowledge.

Metrics We evaluate model performance using three complementary metrics:

• **Execution Accuracy (EX):** EX measures the ratio of correctly predicted SQL programs by comparing their execution results with those of the ground-truth SQL programs on the same database instance.

• **Valid Efficiency Score (VES):** VES evaluates the efficiency of correctly predicted SQL programs by considering their execution time. Formally, VES is given by:

$$\text{VES} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}(n) \cdot R(n)$$

, Where $R(n)$ denotes the ratio between the execution time of the predicted SQL and the gold SQL for the n-th text query. The indicator function $\mathbb{1}(n)$ equals 1 if the predicted SQL is correct, and 0 otherwise.

• **Vulnerability Exposure Rate (VER):** This metric quantifies the method's *capability* to uncover latent defects in scenarios where the model typically performs well. We focus specifically on the set of instances D that the target model solves correctly under standard baseline conditions. Let $D_{exposed} \subseteq D$ denote the subset of these instances for which our method successfully induces a valid failure. The VER is defined as:

$$\text{VER} = \frac{|D_{exposed}|}{|D|}.$$

A higher VER indicates a stronger capability of the evaluation framework to penetrate the model's superficial robustness and reveal deep-seated vulnerabilities that remain hidden during standard testing.

• **Attempts per Discovery (ApD):** We introduce this metric to quantify the *efficiency* of capturing valid failure patterns. Let m denote the total number of interaction attempts, and n denote the number of *distinct, validated discoveries* (deduplicated by root issue). We define ApD as:

$$\text{ApD} = \frac{m}{\max(n,\, 1)}.$$

This ratio reflects the **discovery efficiency**: a lower ApD indicates a highly efficient process where valid issues are surfaced with fewer interactions.

4.2 Main Results

Table 1 presents the comparative evaluation results on the BIRD and Spider benchmarks. We assess model performance under the standard setting (*Original*) against two vulnerability discovery

settings: the rule-based *Expert* baseline and our proposed automated framework, SAGE. The results indicate a substantial performance gap between the standard evaluation and the vulnerability discovery settings across both datasets. On the **BIRD** dataset, while the evaluated models achieve an average Execution Accuracy (EX) of 62.47% in the *Original* setting, this metric decreases to an average of 14.82% under the SAGE setting. A similar trend is observed on **Spider**, where the average EX drops from 83.69% to 33.46% when subjected to our framework. These declines in EX and the corresponding Valid Efficiency Score (VES) suggest that widely used benchmarks may not fully reflect the reliability of models when facing semantically equivalent but adversarial contexts constructed for vulnerability discovery.

Comparing the two discovery methods, SAGE demonstrates consistent improvements over the *Expert* baseline in terms of both discovery capability and efficiency. Regarding capability, SAGE exposes a higher proportion of vulnerabilities across both benchmarks. On BIRD, our method achieves an average Vulnerability Exposure Rate (VER) of 76.98%, surpassing the *Expert* baseline's average of 50.66%. Similarly, on Spider, SAGE outperforms the baseline with an average VER of 58.45% compared to 39.79%. We perform an analysis of discovered vulnerability patterns in Appendix A.1 for a better understanding of the effectiveness of SAGE. In terms of efficiency, SAGE requires fewer interactions to identify valid issues, as evidenced by the Attempts per Discovery (ApD) metric. The average ApD on BIRD decreases from 8.21 with the *Expert* method to 5.94 with SAGE, and on Spider, it decreases from 13.84 to 9.64, indicating a more targeted discovery process.

The quantitative advantage of SAGE can be attributed to its dynamic knowledge refinement mechanism. While the *Expert* approach relies on a fixed set of pre-defined heuristic rules, which limits its discovery scope to anticipated patterns, SAGE utilizes information from identified vulnerabilities to guide subsequent exploration during the runtime. This feedback loop allows the framework to adaptively target specific weaknesses in the LLMs, resulting in higher exposure rates and improved search efficiency compared to static strategies. This demonstrates the necessity of an evolving, automated framework for comprehensively auditing model robustness.

Perturbation Scope	**VER** (↑)		**ApD** (↓)	
	Score	**Δ**	**Score**	**Δ**
Query Only	73.75	-10.70	5.54	+0.46
Relevant Schema Only	51.15	-33.30	8.90	+3.82
Irrelevant Schema Only	48.12	-36.33	10.84	+5.76
SAGE	**84.45**	–	**5.08**	–

Table 2: Ablation study on **BIRD** with Gemma-3, analyzing the impact of distinct perturbation scopes. We report **VER** (Vulnerability Exposure Rate, higher is better) and **ApD** (Attempts per Discovery, lower is better). SAGE targets all scopes simultaneously. **Δ** values indicate the performance degradation (lower VER or higher ApD) when restricting the perturbation to a single scope.

4.3 Ablation Study: Impact of Perturbation Scopes

Table 2 investigates the contributions of different input components—Query & Evidence, Relevant Schema, and Irrelevant Schema—to the vulnerability discovery process. We compare the full SAGE framework against restricted baselines where perturbations are confined to a single scope.

Effectiveness of Individual Scopes. The results indicate that perturbations applied to any individual scope are effective in revealing model defects. Confining the exploration to the *Query & Evidence* alone yields a high Vulnerability Exposure Rate (VER) of 73.75%, confirming that semantic variations in the natural language question are a primary source of model confusion. Similarly, modifying only the *Relevant Schema* or *Irrelevant Schema* also uncovers a significant portion of failures, with VERs of 51.15% and 48.12% respectively. This demonstrates that vulnerabilities are distributed across all dimensions of the input context, and each component serves as a critical entry point for probing model's potential vulnerability.

Superiority of the Comprehensive Scope. While single-scope strategies are valid, SAGE achieves superior performance by covering the comprehensive input space. First, regarding **discovery capability**, SAGE attains the highest VER of 84.45%, outperforming the best single-scope variant by over 10 percentage points. By not limiting the perturbation to a single dimension, our method can identify a wider range of failure patterns that exist across the query and schema components.

Second, regarding **discovery efficiency**, SAGE achieves the lowest Attempts per Discovery (ApD) score of 5.08, indicating a more efficient search

Iteration T	EX (↓)	VER (↑)	ApD (↓)
1	24.64	54.07	7.73
2	14.73	72.54	5.68
3	**8.34**	**84.45**	**5.08**

Table 3: Sensitivity of SAGE to the iteration budget T, evaluated on Gemma-3 over the BIRD dev set. As T increases, VER improves consistently while ApD decreases, demonstrating that a larger search budget uncovers more vulnerabilities with greater efficiency.

process than any restricted setting. We attribute this efficiency gain to the data-driven nature of our *Knowledge Refinement* mechanism. Since the full-scope approach exposes a larger volume and variety of failures (higher VER), it provides richer feedback data for the system to learn from. This allows SAGE to accumulate more diverse failure patterns and guide subsequent exploration more effectively, thereby accelerating the discovery process compared to searching within a constrained scope.

4.4 Ablation Study: Sensitivity to Iteration Count

To study the effect of the iteration budget, we evaluate SAGE on Gemma-3 over the BIRD dev set while varying the maximum number of iterations T.

As shown in Table 3, the VER rises consistently from 54.07% at $T{=}1$ to 84.45% at $T{=}3$, demonstrating that SAGE systematically uncovers deeper layers of latent vulnerabilities as the search budget expands. Notably, even a single iteration already rivals the static *Expert* baseline, confirming that experience-guided probing with just one pass is competitive with hand-crafted heuristics, and that subsequent iterations compound this advantage through evolving Codex feedback.

More critically, the ApD decreases consistently from 7.73 at $T{=}1$ to 5.08 at $T{=}3$. This directly validates the effectiveness of the Vulnerability Codex as a *memory mechanism*: as SAGE accumulates and refines failure knowledge across iterations, the search becomes progressively more targeted and efficient, rather than degrading into blind guessing. Together, these trends validate our choice of $T{=}3$ as a favorable balance between discovery depth and computational cost.

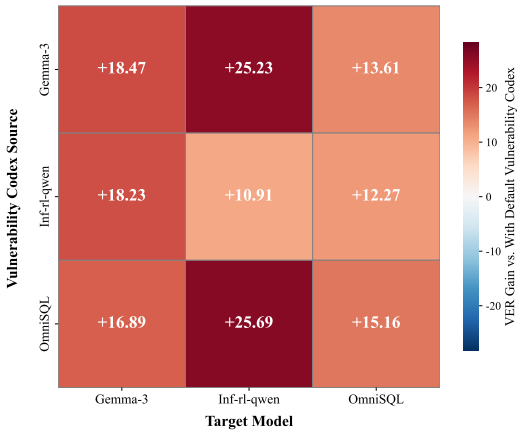

Figure 2: Cross-model strategy transfer performance. The heatmap illustrates the absolute VER gain of Target Models (x-axis) when SAGE uses a Vulnerability Codex constructed from different Source Models (y-axis), compared with the *Expert* baseline that uses a static codex. The values are absolute gains over the Expert baseline under the same evaluation framework.

5 Analysis

5.1 Cross-Model Discovered Vulnerability Generalization Analysis

Figure 2 illustrates the cross-model generalization of SAGE by visualizing the VER gains of a Target Model (x-axis) when guided by the vulnerability codex derived from a Source Model (y-axis), measured as absolute gains relative to the *Expert* baseline. Along the main diagonal, where each target model uses its own evolved codex, the self-codex gains range from $+10.91\%$ to $+18.47\%$, showing that dynamically refined strategies consistently outperform manually curated expert rules. We also observe robust cross-model transferability: strategies derived from one model (e.g., OmniSQL) still improve the VER of distinct architectures (e.g., Gemma-3 by $+13.74\%$). This indicates that the discovered vulnerabilities capture structural weaknesses that generalize across models rather than merely reflecting model-specific artifacts.

5.2 Mitigating Vulnerability with Fine-tuning

The adversarial samples produced by SAGE's discovery process constitute a targeted form of data synthesis for LLM training (Zhu et al., 2025a,b). To investigate whether these samples can be transformed into actionable defense signals, we conducted a lightweight mitigation experiment. Specifically, we selected 1,000 samples from the BIRD training set and employed SAGE to discover vulnerabilities on *Gemma-3*, generating 1,580 adver-

sarial samples. These samples—representing high-confidence failure modes or "hard" cases—were then used to perform Supervised Fine-Tuning (SFT) on *Qwen2.5-Coder-7B-Instruct*.

As presented in Table 4, this targeted fine-tuning yields consistent improvements. Under the adversarial evaluation of SAGE, the Execution Accuracy (EX) of the model rises from 4.37% to 9.58%, representing an absolute gain of $+5.21\%$ and a relative improvement of over 119% compared to the base model. On the standard BIRD dev set, EX also improves from 42.63% to 52.87%, indicating no observable robustness-general accuracy trade-off in this setting. Simultaneously, Attempts per Discovery (ApD) increases from 3.61 to 4.59.

These results are particularly notable given the data efficiency of the approach—using only 1,580 samples to achieve such gains. The concurrent rise in both EX and ApD indicates that the model has not only corrected specific errors but has also become inherently more resilient; it now forces the attacker to search deeper to find a successful perturbation. This suggests that the specific structural weaknesses targeted by our adversarial data have been effectively mitigated, validating the quality of the generated samples.

Crucially, this experiment provides a practical validation of the transferability phenomenon discussed in Section 5.1. By successfully repairing vulnerabilities in *Qwen2.5-Coder* using adversarial patterns discovered in *Gemma-3*, we reconfirm that the flaws exposed by SAGE are not model-specific artifacts but generalized linguistic or logic loopholes shared across different LLMs.

Ultimately, this experiment highlights the dual value of SAGE. It functions not merely as a diagnostic tool for identifying defects but as a constructive framework capable of "closing the loop"—enabling researchers to not only pinpoint weaknesses but also leverage them to synthesize robust training data, thereby actively enhancing the reliability of large language models.

6 Conclusion

In this paper, we propose SAGE, a self-evolving framework for automated vulnerability discovery in Text-to-SQL. Extensive experiments confirm that SAGE significantly outperforms static expert baselines in identifying weaknesses. We further demonstrate that the discovered vulnerabilities are highly transferable across different LLMs. Finally, our mitigation experiments show that SAGE can effectively close the loop: the generated adversarial samples serve as valuable training data to repair model defects and improve overall robustness via lightweight fine-tuning.

Model Setting	EX		ApD
	Adv.	Std.	
Qwen2.5-Coder	4.37	42.63	3.61
+ SFT	**9.58** (+5.21)	**52.87** (+10.24)	**4.59** (+0.98)

Table 4: Mitigation results via adversarial fine-tuning. We fine-tune *Qwen2.5-Coder-7B-Instruct* using 1,580 adversarial samples generated by SAGE (targeting *Gemma-3*). Adv. and Std. denote EX on the SAGE-generated adversarial evaluation set and the standard BIRD dev set, respectively.

Limitations

We acknowledge two main limitations in our current work. First, our ablation studies are conducted primarily at the framework level; a more fine-grained analysis is required to isolate the specific contributions of internal components like the Judge, Embedding, and Summarizer modules. Second, the update mechanism of our Vulnerability Codex operates at coarse intervals; adopting a high-frequency, per-batch update strategy could further enhance the efficiency of the self-evolving process.

Acknowledgements

This project was supported by National Natural Science Foundation of China (No. 62306132), Guangdong Basic and Applied Basic Research Foundation (No. 2025A1515011564), Natural Science Foundation of Shanghai (No. 25ZR1402136). We thank the anonymous reviewers for their insightful feedback on this work. This work was done during Yongdong's internship at SUSTech.

References

Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(1):29–81.

Arian Askari, Christian Poelitz, and Xinye Tang. 2025. Magic: Generating self-correction guideline for in-context text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23433–23441.

Adithya Bhaskar, Tushar Tomar, Ashutosh Sathe, and Sunita Sarawagi. 2023. Benchmarking and improving text-to-sql generation under ambiguity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7053–7074.

Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, William Yang Wang, Zhiguo Wang, Vittorio Castelli, Patrick Ng, and Bing Xiang. 2023a. Dr.spider: A diagnostic evaluation benchmark towards text-to-SQL robustness. In *The Eleventh International Conference on Learning Representations*.

Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, and 1 others. 2023b. Dr. spider: A diagnostic evaluation benchmark towards text-to-sql robustness. *arXiv preprint arXiv:2301.08881*.

Song Cheng, Qiannan Cheng, Linbo Jin, Lei Yi, and Guannan Zhang. 2025. Sqlord: A robust enterprise text-to-sql solution via reverse data generation and workflow decomposition. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 919–923.

Yongdong Chi, Hanqing Wang, Yun Chen, Yan Yang, Jian Yang, Zonghan Yang, Xiao Yan, and Guanhua Chen. 2025. Pi-SQL: Enhancing text-to-SQL with fine-grained guidance from pivot programming languages. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 25120–25144, Suzhou, China. Association for Computational Linguistics.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2021. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. *Computational Linguistics*, 47(2):309–332.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. 2025. Reforce: A text-to-sql agent with self-refinement, format restriction, and column exploration. In *ICLR 2025 Workshop: VerifAI: AI Verification in the Wild*.

Xiang Deng, Ahmed Hassan, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-sql. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350.

Zhongjun Ding, Yin Lin, and Tianjing Zeng. 2025. Ambisql: Interactive ambiguity detection and resolution for text-to-sql. *arXiv preprint arXiv:2508.15276*.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-sql models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2505–2515.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-sql generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8926–8931.

Yue Gong, Chuan Lei, Xiao Qin, Kapil Vaidya, Balakrishnan Murali Narayanaswamy, and Tim Kraska. 2025. SQLens: An end-to-end framework for error detection and correction in text-to-SQL. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yu Guo, Dong Jin, Shenghao Ye, Shuangwu Chen, Jianyang Jianyang, and Xiaobin Tan. 2025. SQL-Forge: Synthesizing reliable and diverse data to enhance text-to-SQL reasoning in LLMs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8441–8452, Vienna, Austria. Association for Computational Linguistics.

Zijin Hong, Zheng Yuan, Qinggang Zhang, and 1 others. 2024. Next-generation database interfaces: A survey of llm-based text-to-sql. *arXiv preprint arXiv:2406.08426*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Infly and InfTech. 2025. inf-rl-qwen-coder-32b-2746. `https://huggingface.co/infly/inf-rl-qwen-coder-32b-2746`. Accessed: 2026-01-05.

Peng Lai, Zhihao Ou, Yong Wang, Longyue Wang, Jian Yang, Yun Chen, and Guanhua Chen. 2026. Biasscope: Towards automated detection of bias in LLM-as-a-judge evaluation. In *The Fourteenth International Conference on Learning Representations*.

Fei Li and Hosagrahar V Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, and 1 others. 2025. Omnisql: Synthesizing high-quality text-to-sql data at scale. *arXiv preprint arXiv:2503.02240*.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of*

the AAAI Conference on Artificial Intelligence, volume 37, pages 13067–13075.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024a. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2024b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Jinyang Li, Binyuan Hui, Ge Qu, and 1 others. 2023b. Bird: A big bench for large-scale database grounded text-to-sql. *NeurIPS*.

Hanbing Liu, Haoyang Li, Xiaokang Zhang, Ruotong Chen, Haiyong Xu, Tian Tian, Qi Qi, and Jing Zhang. 2025. Uncovering the impact of chain-of-thought reasoning for direct preference optimization: Lessons from text-to-sql. *arXiv preprint arXiv:2502.11656*.

Xiping Liu and Zhao Tan. 2024. Epi-sql: Enhancing text-to-sql translation with error-prevention instructions. *arXiv preprint arXiv:2404.14453*.

Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. 2020. Hybrid ranking network for text-to-sql. *arXiv preprint arXiv:2008.04759*.

Tanzim Mahmud, KM Azharul Hasan, Mahtab Ahmed, and Thwoi Hla Ching Chak. 2015. A rule based approach for nlp based query processing. In *2015 2nd international conference on electrical information and communication technologies (EICT)*, pages 78–82. IEEE.

Rajaswa Patil, Manasi Patwardhan, Shirish Karande, Lovekesh Vig, and Gautam Shroff. 2023. Exploring dimensions of generalizability and few-shot transfer for text-to-sql semantic parsing. In *Transfer Learning for Natural Language Processing Workshop*, pages 103–114. PMLR.

Xinyu Pi, Bing Wang, Yan Gao, Jiaqi Guo, Zhoujun Li, and Jian-Guang Lou. 2022. Towards robustness of text-to-sql models against natural and realistic adversarial table perturbation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2007–2022.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36:36339–36348.

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, Vienna, Austria. Association for Computational Linguistics.

Ariana Sahitaj, Markus Nilles, Ralf Schenkel, and Vera Schmitt. 2025. Utilising large language models for adversarial attacks in text-to-sql: A perpetrator and victim approach. In *Datenbanksysteme für Business, Technologie und Web (BTW 2025)*, pages 919–931. Gesellschaft für Informatik, Bonn.

Irina Saparina and Mirella Lapata. 2024. Ambrosia: A benchmark for parsing ambiguous questions into database queries. *Advances in Neural Information Processing Systems*, 37:90600–90628.

Jiawei Shen, Chengcheng Wan, Ruoyi Qiao, Jiazhen Zou, Hang Xu, Yuchen Shao, Yueling Zhang, Weikai Miao, and Geguang Pu. 2025. A study of in-context-learning-based text-to-sql errors. *arXiv preprint arXiv:2501.09310*.

Lei Sheng, Shuai-Shuai Xu, and Wei Xie. 2025. Basesql: A powerful open source text-to-sql baseline approach. *arXiv preprint arXiv:2502.10739*.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. 2025. A survey on employing large language models for text-to-sql tasks. *ACM Comput. Surv.* Just Accepted.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Hanchen Xia, Feng Jiang, Naihao Deng, Cunxiang Wang, Guojiang Zhao, Rada Mihalcea, and Yue Zhang. 2024. $r\hat{}3$:" this is my sql, are you with me?" a consensus-based multi-agent system for text-to-sql tasks. *arXiv preprint arXiv:2402.14851*.

Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. Opensearch-sql: Enhancing text-to-sql with dynamic few-shot and consistency alignment. *Preprint*, arXiv:2502.14913.

Bo Xu, Shufei Li, Yifei Wu, Shouang Wei, Ming Du, Hongya Wang, and Hui Song. 2024. Chain-of-program prompting with open-source large language models for text-to-sql. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others.

2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, and 1 others. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, pages 3911–3921. Association for Computational Linguistics.

Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024. Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. *arXiv preprint arXiv:2403.02951*.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, and 1 others. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

He Zhu, Yifan Ding, Yicheng Tao, Zhiwen Ruan, Yixia Li, Wenjia Zhang, Yun Chen, and Guanhua Chen. 2025a. FANNO: Augmenting high-quality instruction data with open-sourced LLMs only. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 17633–17653, Vienna, Austria. Association for Computational Linguistics.

He Zhu, Zhiwen Ruan, Junyou Su, Xingwei He, Yun Chen, Wenjia Zhang, and Guanhua Chen. 2025b. Tag-instruct: Controlled instruction complexity enhancement through structure-based augmentation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 17708–17729, Vienna, Austria. Association for Computational Linguistics.

A Complementary Experiments

A.1 Analysis of Discovered Vulnerability Patterns

As illustrated in Figure 3, our framework uncovers a distinct "Novel Patterns" category that, while constituting only approximately 10% to 11% of the total error distribution, represents a critical set of previously overlooked vulnerabilities. Far from being a monolithic outlier, this category encapsulates a rich diversity of complex structural failures—most notably Schema Misinterpretation and Syntactic Overfitting—that distinctively differ from traditional error taxonomies. The significance of this segment extends far beyond its statistical proportion; as validated by our mitigation experiments, these specific failure modes serve as high-value training signals. The ability to leverage these patterns to construct effective data for model repair confirms that these newly identified vulnerabilities are not merely rare edge cases, but cornerstone defects whose resolution is fundamental to enhancing the overall robustness of LLMs.

Setting	EX	VER	ApD
Original	65.38	-	-
SAGE	**22.36** (-43.02)	65.80	5.35

Table 5: **Resource-efficient evaluation on GPT-4o.** Constrained by budget, we perform only a **single iteration** of attack using strategies transferred from *Gemma-3*. Even under this restricted setting, the framework successfully identifies widespread vulnerabilities.

A.2 Cost-Effective Scalability on State-of-the-Art LLMs

To assess the scalability of our approach while addressing the practical challenges of evaluating proprietary models, we targeted *GPT-4o*. Motivated by the prohibitive inference costs associated with extensive adversarial optimization on such large-scale models, we adopted a budget-constrained evaluation protocol.

Instead of performing a full-scale search from scratch, we directly leveraged the vulnerability codex pre-distilled from the smaller *Gemma-3* model. Furthermore, to strictly minimize computational overhead, we restricted the attack process on GPT-4o to a **single iteration**.

As presented in Table 5, despite this minimal investment of resources, our framework proved strikingly effective. The single-iteration attack successfully exposed significant vulnerabilities in GPT-4o, driving the Execution Accuracy down to 22.36% and achieving a Vulnerability Exposure Rate (VER) of 65.80%. This result highlights a critical advantage of SAGE: it enables researchers to uncover latent flaws in costly, state-of-the-art models by transferring "knowledge" from cheaper, open-source models, offering a highly resource-efficient pathway for robustness evaluation.

B More implementation details

B.1 Reproducibility Details

Table 6 consolidates the main search budget, inference settings, and the size of the initially-correct subset $|D|$ used to compute VER.

B.2 Detailed Algorithm of SAGE

The complete Algorithm 1 summarizes the implementation of SAGE.

B.3 Training configurations of Fine-tuning Experiments

This section lists the key hyperparameters for the parameter-efficient fine-tuning experiments on *Qwen2.5-Coder-7B-Instruct* in Table 7.

Beyond the established categories, Figure 3 further presents a detailed analysis of errors typically grouped under the *OTHER* category. By combining expert manual inspection with systematic reclassification, we uncover recurring and nuanced cognitive failure patterns. These include: schema misinterpretation, syntactic overfitting, insufficient prioritization of explicit evidence, overconfidence, non-robust text matching, erratic self-correction, and overreliance on temporal inference. This trigger-based partitioning moves beyond surface-level heuristics, substantially enriching standard error categories and, more importantly, providing fine-grained, actionable insights for targeted model improvement.

B.4 Prompt Templates for Sample Perturbation

We provide the prompt templates used to generate sample perturbations in our framework. Figures 4, 5, and 6 illustrate the prompt templates for perturbations applied to natural language queries, schema-irrelevant elements, and schema-relevant elements, respectively.

B.5 Checker validation

To verify the reliability of the automated Checker (implemented with Qwen3-32B), we conducted both a targeted human evaluation and a larger-scale cross-model proxy validation. We recruited three graduate-level annotators with expertise in Text-to-SQL tasks to ensure the quality of the assessment. They were compensated at a competitive market rate for such tasks. All participants were informed about the purpose of the evaluation task and how the data would be used prior to the annotation. They provided informed consent to participate in the study. Since the study involves standard data annotation tasks with minimal risk to participants and does not collect personally identifiable information (PII), it was determined to be exempt from formal IRB review according to our institution's guidelines.

From the Gemma-3 experiments, we randomly sampled 100 perturbed cases for evaluation. Each annotator independently judged whether each perturbed query was valid (i.e., the perturbation preserves semantic equivalence and the correct answer should remain unchanged) or invalid (i.e., the perturbation alters the meaning or introduces inconsistencies).

Each annotator was provided with (i) the original natural-language query and its gold answer, (ii) the perturbed query, and (iii) the corresponding M-schema context. They followed a concise guideline: (1) assess semantic equivalence between the original and perturbed queries, (2) verify whether the perturbation is realistic and contextually appropriate, and (3) label the sample as either Valid or Invalid. The final ground-truth label for each case was determined by majority vote across the three annotators to reduce subjectivity and ensure consistency.

We then compared the Checker's predictions with these human consensus labels. The Checker achieved an average accuracy of 89.3%, indicating a high level of agreement with expert human judgments in identifying valid perturbations.

To complement this targeted human assessment, we further sampled 2,000 perturbations that were originally validated by the Qwen3-32B Checker and re-evaluated them using GPT-5.1 as an independent proxy judge. Table 8 reports the agreement rates across different perturbation scopes.

The Relevant Schema setting is the most challenging because modifying metadata for columns used in the gold SQL more easily introduces subtle semantic ambiguity. We therefore view this proxy study as complementary evidence rather than a replacement for human judgment. At the same time, the consistently high agreement and the mitigation gains in Table 4 jointly suggest that the discovered perturbations are not dominated by invalid rewrites.

C Ethical Considerations

This work aims to improve the robustness of Text-to-SQL systems by automating the discovery of

Item	Details
Max iterations (T)	3
Hypotheses per sample (K)	3 initial hypotheses per sample
Perturbation scopes	Query, Relevant Schema, and Irrelevant Schema
Experience retrieval	Top-5 entries retrieved from the current Vulnerability Codex
Similarity threshold (τ)	0.1 for semantic compression
Sampling settings	Default chat sampling settings of each backbone; for Qwen3-32B, we use $temperature = 0.6$, $top\_k = 20$, and $top\_p = 0.95$
Stopping rule	Stop searching a hypothesis once a valid failure is exposed or when the maximum iteration budget is reached
LLM modules	Generator / Checker / Summarizer: Qwen3-32B; Embedding model: Qwen3-Embedding-4B
Initially-correct subset $\|D\|$ on BIRD	Gemma-3: 823; Inf-rl-qwen: 1082; OmniSQL: 970
Initially-correct subset $\|D\|$ on Spider	Gemma-3: 843; Inf-rl-qwen: 907; OmniSQL: 846

Table 6: Consolidated reproducibility details and the size of the initially-correct evaluation subset $|D|$ used to compute VER.

Hyperparameters	Details
batch_size	64
Effective learning rate	1.0×10^{-4}
Training epochs	3
LR scheduler	cosine
warmup	0.1
Precision	bf16
Max seq length	10240

Table 7: Training hyperparameters of fine-tuning experiments.

Scope	# Samples	Agreement
Query Only	661	96.52
Irrelevant Schema	662	95.62
Relevant Schema	677	87.44
Overall	2000	93.15

Table 8: Cross-model proxy validation of the Qwen3-32B Checker using GPT-5.1 as an independent judge. Agreement is reported in percentage.

corner cases and vulnerabilities. Here, "vulnerability" refers to robustness and reliability brittleness under semantic-preserving perturbations, not SQL-injection-style exploits. While our method is designed to facilitate debugging and model improvement, we recognize that it carries a potential risk of misuse for adversarial attacks. We emphasize that identifying these weaknesses proactively is essential for mitigating risks in real-world deployments, and we urge users to apply this framework responsibly to enhance system security.

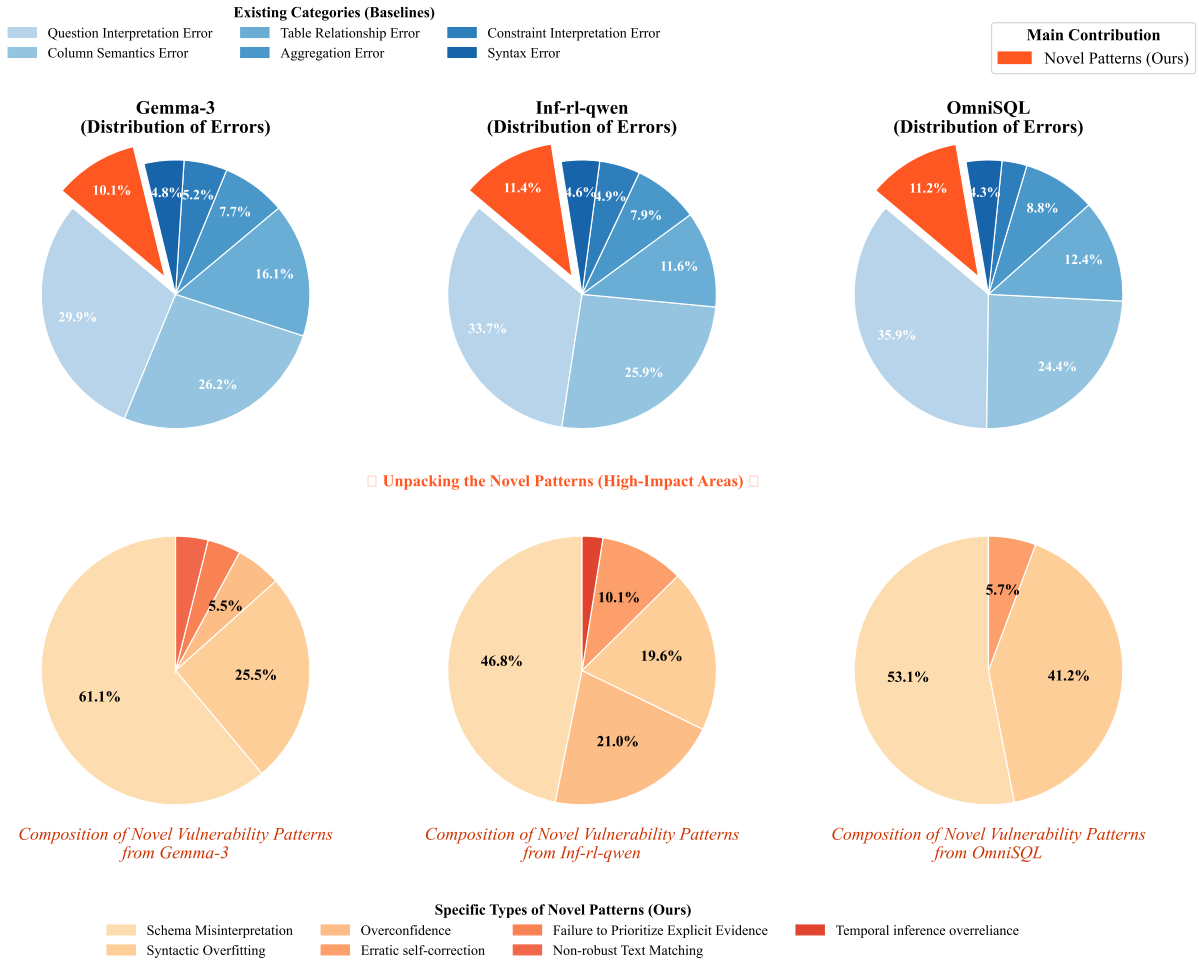

Figure 3: Distribution and hierarchical analysis of failure patterns across Gemma-3, Inf-rl-qwen, and OmniSQL. The top row illustrates the overall error distribution, distinguishing between established error categories (cool tones) and the Novel Patterns identified by our approach (orange, exploded slice). The bottom row provides a fine-grained breakdown of these novel patterns, revealing that specific failures—such as Schema Misinterpretation and Syntactic Overfitting—constitute a significant portion of errors previously categorized broadly as "Other".

Algorithm 1 Systematic Automated Guided Exploration (SAGE)

Require: Target Model M_t.
Require: Dataset $D = \{(q_i, s_i, y_i^*)\}_{i=1}^{N}$ where M_t yields correct execution results for all (q_i, s_i).
Require: LLM Modules: Generator M_g, Checker M_c, Summarizer M_s, Embedding Model M_e.
Require: Perturbation Scopes $\mathbb{S} = \{s_{\text{query}}, s_{\text{schema}}, \dots\}$.
Ensure: Refined Vulnerability Codex $\mathcal{V}_{final}$.

Initialize: Iteration counter $t \leftarrow 1$, Vulnerability Codex $\mathcal{V}_0$.
Phase 1: Systematic Hypothesis Generation
Synthesize initial set of vulnerability hypotheses from dataset D:
$\mathcal{H}_{curr} \leftarrow \bigcup_{d_i \in D} \{(d_i, h_{i,k}) \mid h_{i,k} \in \text{HypothesisGen}(M_g, d_i, K = 3)\}$ ▷ Each sample d_i spawns K distinct hypotheses
while $t \leq T$ **and** $\mathcal{H}_{curr} \neq \emptyset$ **do**
  $\mathcal{V}_{\text{new}}^{(t)} \leftarrow \emptyset$ ▷ Storage for new archetypes found in this iter
  $\mathcal{H}_{next} \leftarrow \emptyset$ ▷ Hypotheses that fail to expose bugs are retained
  Phase 2: Experience-Guided Probing & Verification
  for each candidate tuple $C_j = (d_i, h_{i,k})$ in $\mathcal{H}_{curr}$ **do**
    Retrieve relevant insights: $E_j \leftarrow \text{RetrieveTopN}(M_e, \mathcal{V}_{t-1}, h_{i,k})$
    ▷ Dynamic Scope Selection
    if $t = 1$ **then**
      Select single scope: $\mathbb{S}_{active} \leftarrow \{\mathbb{S}[(k-1) \pmod{|\mathbb{S}|} + 1]\}$ ▷ Cold Start: Efficiently map 1 hypothesis to 1 scope
    else
      Select all scopes: $\mathbb{S}_{active} \leftarrow \mathbb{S}$
    end if
    $IsExposed \leftarrow$ **false**
    for each scope $s_m \in \mathbb{S}_{active}$ **do**
      ▷ Apply hypothesis $h_{i,k}$ onto scope s_m guided by experience E_j
      $x_{j,m} \leftarrow \text{SampleGenerate}(M_g, C_j, s_m, E_j)$
      if $\neg\text{QualityCheck}(x_{j,m}, M_c)$ **then continue**
      end if
      $y_{j,m} \leftarrow \text{TargetModelQuery}(M_t, x_{j,m})$
      $IsCorrect \leftarrow \text{VerifyAnswer}(d_i, x_{j,m}, y_{j,m})$
      if $\neg IsCorrect$ **then** ▷ Vulnerability Discovered
        ▷ Abstract the concrete failure case into a generalized archetype
        $v_j \leftarrow \text{AbstractVulnerability}(M_s, C_j, x_{j,m}, y_{j,m})$
        if v_j is non-trivial **then**
          $\mathcal{V}_{\text{new}}^{(t)} \leftarrow \mathcal{V}_{\text{new}}^{(t)} \cup \{v_{\text{arch}}\}$
        end if
        $IsExposed \leftarrow$ **true**
        break ▷ Pruning: Hypothesis validated, skip remaining scopes
      end if
    end for
    if $\neg IsExposed$ **then**
      $\mathcal{H}_{next} \leftarrow \mathcal{H}_{next} \cup \{C_j\}$
    end if
  end for
  Phase 3: Codex Evolution & Management
  $\mathcal{V}_t \leftarrow \mathcal{V}_{t-1} \cup \mathcal{V}_{\text{new}}^{(t)}$
  ▷ Refine the Codex by merging similar archetypes
  $\mathcal{V}_t \leftarrow \text{SemanticCompress}(M_e, \mathcal{V}_t, \tau)$
  $\mathcal{H}_{curr} \leftarrow \mathcal{H}_{next}$
  $t \leftarrow t + 1$
end while
$\mathcal{V}_{final} \leftarrow V_{t-1}$
return $\mathcal{V}_{final}$

The prompt templates for perturbations applied to natural language queries

Task:
Please **rewrite the question and the evidence** in a way that **preserves the original intent and does not change the final correct answer** (i.e., the gold SQL query must still be valid and applicable).
Your rewritten version should:
- Better reflect **real-world use cases or natural user phrasing**.
- Increase **linguistic and structural diversity** ...
- Introduce **more contextual framing, ambiguity, or reasoning cues** ...
- Remain **faithful to the original semantics and answer**.

Guidelines:
- Do **not** change the final answer (i.e., gold SQL must still be valid).
- Focus on **rewriting**, not expanding or omitting information.
- You must **describe your modification strategy** clearly in the ‘improvement‘ field.

Output Format:
Please return your output strictly in the following JSON format:
“‘json
{{
“improvement": “Describe what changes you made and why ...",
“question": “your rewritten question",
“evidence": “your rewritten evidence"
}}
Example 1-3: ...

Now it is your turn:
Below is the database schema:
{schema}
Question: {question}
Evidence: {evidence}
The gold SQL query is: {gold_sql}

Figure 4: The prompt templates for perturbations applied to natural language queries.

The prompt templates for perturbations applied to schema-irrelevant elements

Task:
Your task is to **modify the provided database-related information** while ensuring that the **core semantics and the correct answer remain unchanged**.
Your goal is to introduce **maximum lexical, structural, and semantic diversity** in order to **evaluate the model’s robustness** against variations in metadata.
You are allowed to revise the following fields for each column:
- **column_description**: Modify the column description using paraphrasing, synonyms, different sentence structures ...
- **value_description**: Change the way value types, ranges, or categories are described ...
- **example_value**: Add more diverse, realistic, or representative sample values ...

Important Guidelines:
- **Do not** change the table names or column names.
- **Do not** alter the underlying meaning or change the final answer.
- You must output your result in the **strict JSON format** shown below.
- You must change the metadata for **at least 3 columns** and **at most 5 columns**.

Output Format:
“table_name": [
{{
“column": “column_name",
“improvement": “Describe your strategy for modifying this column’s metadata.",
“column_description": “Modified description of the column.",
“value_description": “Modified description of the values.",
“example_value": [“example1", “example2"]
}}

Example 1-3: ...

Below is a partial view of the database schema. It includes only the tables and columns that are directly relevant to answering the given question; all unrelated schema elements have been omitted.
{json_str}
Question: {question}
Evidence: {evidence}
The gold SQL query is: {gold_sql}
Here are some successful attack strategies you can refer to: {error_strategies}
The column_description and value_description fields should each be no longer than 100 tokens.
Now it is your turn. Please return your answer strictly in the above JSON format.

Figure 5: The prompt templates for perturbations applied to schema-irrelevant elements.

The prompt templates for perturbations applied to schema-relevant elements

You are an expert in prompt-based robustness evaluation for database-related tasks.
Task:
Please **modify the database-related information** while ensuring that the **core semantics and the correct answer remain unchanged**. Your goal is to introduce **greater lexical, structural, and semantic diversity** to challenge the robustness of the model.
You may revise the following elements for each column:
- **column_description**: Modify the description using paraphrasing, synonyms, different sentence structures, or multilingual expressions.
- **value_description**: Alter the value type description or re-express the value mappings, ranges, or classifications.
- **example_value**: Enrich the example values with varied, realistic, and semantically appropriate entries.

Important:
- Do **not** change the table name or column name.
- Your output must be in **strict JSON format** as shown below.
- You must change the metadata for **at least 3 columns** and **at most 5 columns**.

Here are some output examples:
Example 1-3: ...
Now it is your turn.
Below is a partial view of the database schema. It includes only the tables and columns that are relevant to answering the given question; unrelated schema elements have been omitted.
{json_str}
Question: {question}
Evidence: {evidence}
The gold SQL query is: {gold_sql}

Return your modifications using the following structure:
{{
“table_name": [
{{
“column": “column_name",
“improvement": “Describe your strategy for modifying this column’s metadata.",
“column_description": “Modified description of the column.",
“value_description": “Modified description of the values.",
“example_value": [“example1", “example2"]
}}
]
}}

Here are some successful attack strategies you can refer to: {error_strategies}
The column_description and value_description fields should each be no longer than 100 tokens.
Please return your answer strictly in the above JSON format.

Figure 6: The prompt templates for perturbations applied to schema-relevant elements.