RSI-SQL: RECURSIVE SELF-IMPROVEMENT FOR TEXT-TO-SQL

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-to-SQL agents can repair a query while retaining the grounding error that produced it. We introduce RSI-SQL (Recursive Self-Improvement for Text-to-SQL), a training-free framework that uses execution feedback to recursively update its own reasoning state. A Graph Grounding Agent revises intent and schema hypotheses, an Actor generates and tests SQL candidates, and a Critic assigns credit to the interactions supporting each trajectory. A typed Action Graph carries the revised hypotheses, observations, and credit into the next round of reasoning. This recursion connects candidate refinement to updates in the evidence used for subsequent decisions. A local analysis characterizes grounding discrepancy and the propagation of intermediate and terminal errors during lookahead. On all 1,534 BIRD development queries, a shared backbone across the three roles achieves 63.50% execution accuracy with Qwen2.5-14B-Instruct and 64.41% with Qwen2.5-Coder-14B-Instruct. Component comparisons reveal interference between retrieval and credit assignment, making their coordination central to the recursive improvement loop. Code is available at `https://anonymous.4open.science/r/RSI-SQL-425F`.

1 INTRODUCTION

Text-to-SQL makes structured data accessible through natural language, but answering a question requires more than producing executable syntax. A system must connect the user's intent to the database's tables, relations, and stored values. From WikiSQL and Spider to Spider-DK, BIRD, and Spider 2.0, benchmarks have expanded from structured query generation to cross-domain parsing, domain knowledge, value grounding, and enterprise workflows (Zhong et al., 2017; Yu et al., 2018; Gan et al., 2021; Li et al., 2023b; Lei et al., 2025). These settings make database interaction part of the reasoning process: a value probe or candidate execution can reveal information that changes how the question should be interpreted.

Recent systems exploit this interaction through task decomposition and self-correction (Pourreza & Rafiei, 2023), tool-assisted schema selection (Talaei et al., 2024), multi-agent coordination (Wang et al., 2025), and candidate selection (Pourreza et al., 2025a). Yet execution feedback is an incomplete guide to meaning. A query with a missing filter can execute successfully; an empty result can arise from either the data or a mistaken predicate. Turning such observations into better subsequent decisions presents two connected challenges.

Grounding under accumulating context. Each interaction adds schema fragments, candidate programs, and tool outputs. The next decision needs the evidence relevant to its current uncertainty, together with the hypothesis that evidence supports. Long-context studies show that a language model's use of relevant information depends on where it appears in the input (Liu et al., 2024). In Text-to-SQL, the problem is especially concrete: the question may say "accepted" while the database stores `yes`/`no`. Repeating the question and schema leaves this mapping implicit. A useful grounding update must expose the mapping and carry it into the next generation step.

Error propagation and credit assignment. An incorrect schema link can persist through several syntactically valid repairs. Candidate-level success alone does not identify which probe, grounding decision, or edit helped the trajectory. As reasoning deepens, later actions inherit earlier interpretations and their errors. The system therefore needs to preserve dependencies between decisions and observations, and use those dependencies when evaluating further actions.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

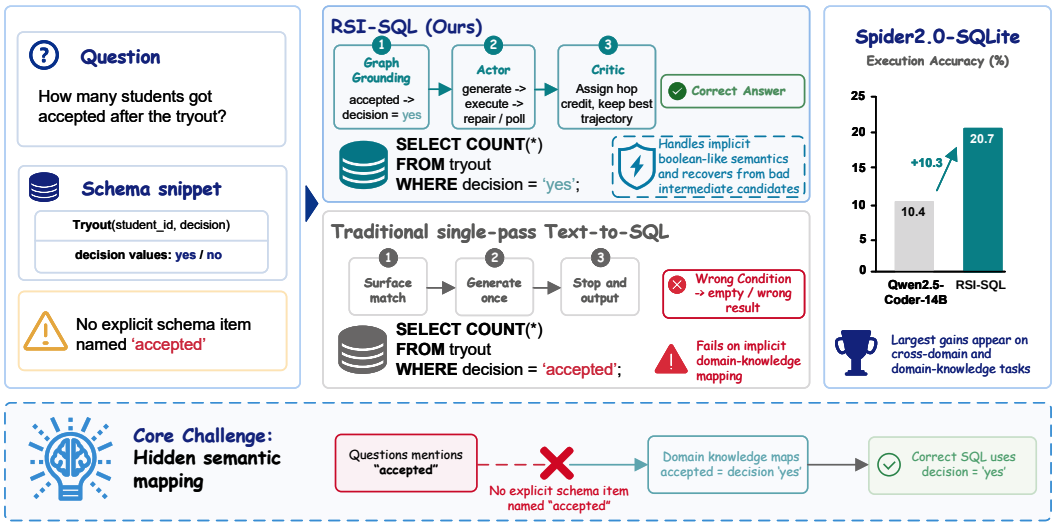


Figure 1: **From direct generation to recursive self-improvement.** The example maps the implicit meaning of "accepted" to `decision = 'yes'`. Grounding, SQL generation and repair, and trajectory credit connect this evidence to candidate selection. The bottom strip isolates the semantic mapping missed by direct surface matching. The right panel shows Spider2.0-SQLite EX for the Coder-14B generator setting in Table 2: 10.4% for the single decoder and 20.7% for RSI-SQL, a 10.3-point difference. The RSI-SQL result is a five-run mean (SD: 1.1 points).

Our central idea is to *improve the reasoning state that produces the next candidate*. We introduce RSI-SQL, short for *Recursive Self-Improvement for Text-to-SQL*. At inference time, a Graph Grounding Agent updates competing intent and schema hypotheses, an Actor generates or repairs SQL and polls the database, and a Critic assigns credit to their interactions. A shared *Action Graph* connects states, actions, observations, and candidate programs through typed dependencies. The resulting hypothesis weights, graph, and trajectory scores become the input to the next round. This recursion operates within each query while the backbone parameters remain frozen.

Figure 1 illustrates the mechanism. Polling the stored values supplies evidence for mapping "accepted" to `decision = 'yes'`. The updated hypothesis guides a repair, and the Action Graph retains the observation supporting that repair. Feedback thus changes both the current SQL and the context for subsequent reasoning. This design connects the iterative refinement principle of Self-Refine and Reflexion (Madaan et al., 2023; Shinn et al., 2023) to explicit database grounding and interaction credit.

Our contributions are:

- **Recursive improvement through typed evidence.** We couple grounding, SQL actions, and credit through an Action Graph whose updated reasoning state guides each subsequent interaction.
- **A local analysis of grounding and lookahead.** We bound grounding discrepancy and characterize how intermediate and terminal state perturbations enter finite-depth rollout error under explicit regularity assumptions.
- **Evaluation of the full system and its components.** On all 1,534 BIRD-Dev queries, RSI-SQL reaches 63.50% and 64.41% EX with aligned Instruct and Coder backbones. Additional benchmarks, retrieval–credit comparisons, and cost and failure analyses examine where recursive interaction helps and where coordination breaks down.

2 Task and Interaction State

Task. Given a question q, schema D, and database executor, the system returns a SQL program y within a finite interaction budget. Execution accuracy compares its answer with the benchmark

reference. During inference, the intended interpretation and reference answer are hidden; the system observes schema information and its own execution feedback.

History, actions, and observations. The observable history x_t contains the question, schema, candidate programs, grounding hypotheses, and interactions through step t. An action $a_t \in \mathcal{A}$ is GENERATE, REPAIR, or POLL: it proposes SQL, edits a candidate using feedback, or inspects the database. The observation o_{t+1} contains an execution result or error, or a schema/value check. An empty result is an observation rather than a correctness label.

Partial observability. The latent interpretation and task configuration z_t define a POMDP (Kaelbling et al., 1998) with transition kernel T, observation likelihood O, reward R, and discount γ. Its conceptual posterior obeys

$$b_{t+1}(z') \propto O(o_{t+1} \mid z', a_t) \int T(z' \mid z, a_t) b_t(z)\, dz. \tag{1}$$

Grounding particles represent uncertainty over intent, entity links, and join paths. The history x_t and its embedding are observable representations, distinct from z_t. The implemented selector ranks execution status and accumulated credit (Section 3.4); γ belongs to the lookahead analysis. Appendix A.1 lists the notation.

3 RECURSIVE SELF-IMPROVEMENT FOR TEXT-TO-SQL

RSI-SQL implements recursive self-improvement over the reasoning state of one query (Figure 2). Let $\mathcal{M}_t = (\mathcal{G}_t, \mathcal{P}_t, \mathcal{T}_t)$ contain the Action Graph, weighted grounding particles, and scored candidate trajectories. Each interaction applies

$$a_t = \mathrm{Act}_\theta(q, D, \mathcal{M}_t), \qquad \mathcal{M}_{t+1} = \mathrm{Update}(\mathcal{M}_t, a_t, o_{t+1}, c_t). \tag{2}$$

Here Act_θ grounds the current context and selects a proposed action. After the tool observation o_{t+1} and Critic credit c_t arrive, Update revises hypothesis weights, appends graph dependencies, and updates trajectory scores. The next interaction uses $\mathcal{M}_{t+1}$ as its reasoning state. This feedback changes the evidence and assessments available to subsequent decisions. The backbone parameters θ remain frozen, while the three state components evolve within the interaction budget.

3.1 TYPED MEMORY FOR RECURSIVE IMPROVEMENT

The Action Graph is $\mathcal{G}_t = (V_t, E_t)$. Its vertices have four types: state, action, observation, and candidate SQL. A state vertex stores the current grounded context. An action vertex stores its

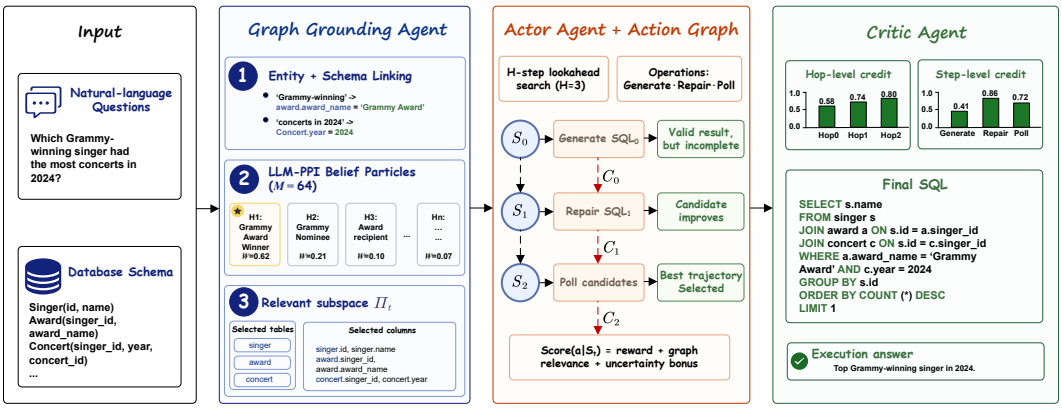


Figure 2: **The RSI-SQL framework.** The Graph Grounding Agent links the question to schema items and maintains weighted hypotheses. The Actor uses the selected schema to generate, repair, and poll SQL candidates along the Action Graph. The Critic assigns hop- and step-level credit to support trajectory selection. Successive states incorporate observations and credit, guiding later actions in the recursive update of Eq. (2). Particle weights and credit bars illustrate the example.

operation and prompt. An observation vertex records an executor result or a schema check. A candidate vertex stores SQL text and execution status. Each vertex has a creation index $\kappa(v)$; every directed edge (u, v) satisfies $\kappa(u) < \kappa(v)$. Creation indices order events within a round as well as across rounds.

Edges connect a prompting state to an action, an action to its candidate, a candidate execution to its observation, and that observation to a later state. Appending new states preserves acyclicity. Credit annotations weight the dependencies used by the Critic to assess a trajectory. Each decision is thus traceable to its supporting observations and hypotheses (Appendix A.3).

3.2 Updating grounding hypotheses

The Graph Grounding Agent maintains M particles $\{(\pi_t^{(i)}, w_t^{(i)})\}_{i=1}^{M}$, where $\pi_t^{(i)}$ is a hypothesis about intent, schema links, and join paths. This weighted hypothesis representation follows the particle-filtering principle of reweighting and resampling from sequential observations (Arulampalam et al., 2002). The language model proposes hypotheses; consistency with execution observations determines their relative weights. With normalized weights, the effective sample size is $\mathrm{ESS} = 1/\sum_i (w_t^{(i)})^2$. Resampling is triggered below $M/2$. Weak evidence can also prompt new candidate programs or join paths. The hypothesis set therefore changes as observations arrive, allowing an initially plausible link to lose support.

Grounding yields a relevant symbol set R_t containing tables, columns, relations, and SQL fragments. For representation analysis, we encode the complete configuration and the grounded abstraction in a shared space:

$$s_t = P\,\mathrm{Pool}(h_\theta(x_t)) \in \mathbb{R}^d, \qquad \widehat{s}_t = E(\phi(x_t)) \in \mathbb{R}^d. \tag{3}$$

Here h_θ is the frozen backbone, Pool is a fixed reduction over token states, $P \in \mathbb{R}^{d \times d_{\mathrm{bb}}}$ is a fixed linear map, and ϕ extracts grounded symbols. The encoder E uses the same backbone, reduction, and projection. These objects have different roles: P changes representation dimension, ϕ selects symbolic content, and E embeds that content. The diagnostic implementation uses $d = 1536$ after projection from $d_{\mathrm{bb}} = 5120$ for the two Qwen2.5-14B variants.

Writing $E_R = [E(r_1), \ldots, E(r_m)]$, we form the regularized grounding operator

$$A_t = E_R(E_R^\top E_R + \lambda I)^{-1} E_R^\top, \qquad \lambda = 10^{-4}. \tag{4}$$

This symmetric shrinkage operator attenuates components outside the selected schema span. Its eigenvalues and regularization dependence are given in Appendix A. Generation consumes the selected symbols and their evidence; the operator supplies a geometric description of this grounding.

3.3 Generation, repair, and evidence acquisition

The Actor expands candidates from the current reasoning state with three operations. Generate conditions SQL synthesis on the question and grounded schema. Repair conditions an edit on a candidate and its observed error or inconsistency. Poll gathers evidence, such as sample values, a join-path check, or a comparison of candidate predicates. A SQL executor supplies tool observations for the latter two operations. Actor proposals are prioritized by

$$S(a \mid x_t) = 0.5\, r_{\mathrm{exec}}(a) + 0.3\, \mathrm{rel}(a, \mathcal{G}_t) + 0.2\, u(a), \tag{5}$$

where the terms are heuristic scores for predicted execution progress, relevance to grounded evidence, and expected uncertainty reduction. The weights were selected once on a validation split and reused across benchmarks. A candidate-action budget and rollout depth bound the expansion. Invalid executions return observations that can trigger repair; an empty result may instead call for another predicate or value check. These observations enter the state update in Eq. (2).

3.4 Credit assignment and trajectory selection

The Critic scores an observed interaction by

$$c_t = 0.5\, r_t^{\mathrm{valid}} + 0.3\, r_t^{\mathrm{progress}} + 0.2\, r_t^{\mathrm{explore}}. \tag{6}$$

Algorithm 1 RSI-SQL: recursive self-improvement for one question

Require: question q, schema D, frozen backbone, particle count M, depth H, action budget B
Initialize reasoning state $\mathcal{M}_0$: grounding particles, an Action Graph, and candidate trajectories
for each permitted expansion round t up to depth H **do**
  Ground the next expansion using hypotheses and observations in $\mathcal{M}_t$
  Propose GENERATE, REPAIR, and POLL actions within the remaining budget
  Rank proposals by Eq. (5); run selected actions and collect tool observations
  Append typed vertices and dependencies to the Action Graph
  Update particle weights; resample if $\mathrm{ESS} < M/2$
  Assign hop and edge credit; update candidate trajectory scores
  Carry the updated graph, particles, and trajectories into $\mathcal{M}_{t+1}$ for the next round
end for
Rank terminal candidates by Eq. (8) and return the selected SQL with its execution status

The first term indicates successful execution. The progress term measures particle concentration. If $v_t = M^{-1} \sum_i \|s_t^{(i)} - \bar{s}_t\|_2^2$ is the particle variance, then

$$r_t^{\text{progress}} = \max\left(0, \frac{v_{t-1} - v_t}{v_{t-1} + \epsilon}\right). \tag{7}$$

The exploration term rewards useful hypothesis expansion. Credit is distributed to contributing graph edges in proportion to their nonnegative relevance. A concentrated belief can still be wrong, so progress supplies a confidence signal alongside execution and evidence consistency.

For a trajectory τ_i, let $C_i = \sum_{t \in \tau_i} c_t$, $L_i = |\tau_i|$, and y_i be its terminal SQL. Selection uses the lexicographic key

$$i^* = \arg\max_i \big(\mathbf{1}[y_i \text{ executes}],\ C_i,\ -L_i\big), \qquad y^* = y_{i^*}, \tag{8}$$

with candidate order breaking any remaining tie. Length is a tie-breaker after credit, so this rule can favor a longer trajectory with a larger accumulated score. This behavior is relevant to the selection errors analyzed in Section 5.5.

4 LOCAL ANALYSIS OF GROUNDING AND LOOKAHEAD

Two choices shape recursive improvement: how much context a grounding update retains, and how far the Actor looks ahead. We analyze them in the representation space of Eq. (3). The quantities below describe local surrogate dynamics under explicit smoothness and perturbation assumptions.

4.1 WHAT CHANGES WHEN CONTEXT IS GROUNDED?

Let V be L_V-Lipschitz on the states under consideration. Define a residual-to-retained energy ratio and an abstraction discrepancy:

$$\rho_t = \frac{\|(I - A_t)s_t\|_2^2}{\|A_t s_t\|_2^2 + \epsilon}, \qquad \delta_t = \|A_t s_t - \widehat{s}_t\|_2, \quad \epsilon > 0. \tag{9}$$

Proposition 1 (Grounding discrepancy)**.** *For s_t, $A_t s_t$, and $\widehat{s}_t$ in this neighborhood,*

$$|V(s_t) - V(\widehat{s}_t)| \le L_V \left[\sqrt{\rho_t(\|A_t s_t\|_2^2 + \epsilon)} + \delta_t\right]. \tag{10}$$

The first term measures how much representation grounding removes; the second measures the mismatch between the retained representation and the encoded symbols. Interpreting the removed component as irrelevant context requires the selected schema to be appropriate. A wrong schema hypothesis or discarded value information can shift the retained representation away from the intended query. The bound characterizes the surrogate value discrepancy introduced by this grounding update.

4.2 How do rollout perturbations accumulate?

Consider a fixed action sequence $a_{0:H-1}$ and coupled clean and perturbed surrogate trajectories s_t and $\widetilde{s}_t$. Holding actions fixed separates transition sensitivity from changes in the decision rule. Let $e_t = \widetilde{s}_t - s_t$ and suppose $\mathbb{E}\|e_t\|_2 \le d_t$. Lemma 1 in Appendix B derives admissible sequences d_t for independent and dependent transition perturbations.

For $0 \le \gamma < 1$, define a surrogate rollout with terminal estimate V_k by

$$Q_H^{\text{noisy},V_k} = \mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t r(\widetilde{s}_t, a_t) + \gamma^H V_k(\widetilde{s}_H)\right]. \tag{11}$$

Let $Q_H^{\text{clean},V}$ use s_t and V with the same actions.

Proposition 2 (Rollout error). *If $r(\cdot, a)$ and V are respectively L_R- and L_V-Lipschitz on these states, and $\mathbb{E}\|e_t\|_2 \le d_t$, then*

$$\left|Q_H^{\text{noisy},V_k} - Q_H^{\text{clean},V}\right| \le \gamma^H \|V_k - V\|_\infty + L_R \sum_{t=0}^{H-1} \gamma^t d_t + \gamma^H L_V d_H. \tag{12}$$

Increasing depth discounts terminal estimation error while exposing the rollout to additional intermediate and terminal state errors. The depth profile depends on the sensitivity sequence and discount. This trade-off motivates measuring the quality and cost of recursive expansion. Extending the fixed-action coupling to an adaptive Actor requires control of action changes. Appendix B gives the proofs and the relation to the recorded diagnostic measurements.

5 Experiments

5.1 Evaluation settings

We evaluate execution accuracy (EX, %) on Spider-DK, Spider-Dev, BIRD-Dev, and Spider2.0-SQLite (Gan et al., 2021; Yu et al., 2018; Li et al., 2023b; Lei et al., 2025). These settings cover domain knowledge, cross-domain schema generalization, realistic databases, and enterprise-derived SQL tasks. The SQLite evaluation is a subset of the Spider 2.0 setting; results on it do not measure performance across all of that benchmark's database engines.

The principal BIRD comparison evaluates all 1,534 development questions in one run per configuration. It uses Qwen2.5-14B-Instruct and Qwen2.5-Coder-14B-Instruct (Qwen et al., 2024; Hui et al., 2024). Within each full-pipeline run, grounding, generation, and credit assignment use the same backbone. DAIL-SQL (Gao et al., 2024) is run with the corresponding backbone, decoding temperature, and execution environment. A base configuration retains Actor and Critic calls while removing graph memory and explicit credit assignment. The particle specification uses $M = 64$, resampling below $M/2$, and up to five new hypotheses per expansion. The documented inference setting uses depth $H = 3$ and a candidate-action budget of 50.

The BIRD experiment aligns the grounding and credit modules with the SQL generator. The additional multi-benchmark and cost evaluations have separate configuration records. Their five-run error bars describe those evaluations; the aligned BIRD study reports one run per setting.

5.2 Full-set BIRD results

Table 1 reports 63.50% EX for the Instruct backbone and 64.41% for Coder. Relative to the DAIL-SQL runs in this experiment, the differences are 29.19 and 22.32 percentage points. Relative to the Actor/Critic base, the recursive improvement pipeline gains 9.91 and 2.61 points. These gains compare the recorded inference budgets; accuracy at equal computation remains unmeasured.

MAC-SQL (Wang et al., 2025) was evaluated with these backbones on five samples of 59 queries. Its reported means are 23.05 ± 2.30 and 35.93 ± 1.66. Appendix C records the sampled protocol, whose query selection differs from the full-set comparison.

Table 1: BIRD-Dev execution accuracy on all 1,534 questions. Each entry is one run. The full pipeline uses the named backbone in all three roles; the two columns are separate backbone settings.

System	Qwen2.5-14B-Instruct	Qwen2.5-Coder-14B-Instruct
DAIL-SQL (same-backbone runs)	34.31	42.09
Actor/Critic base	53.59	61.80
RSI-SQL (full pipeline)	**63.50**	**64.41**



Figure 3: BIRD-Dev component study on all 1,534 questions, one run per setting. The first four settings compare Top-K retrieval and credit around the same Actor/Critic base. The full pipeline includes the complete grounding mechanism and changes several components relative to these simplified settings. Dashed lines mark the full-system EX.

5.3 Evidence and credit in the improvement loop

The component study examines the retrieval and evaluation signals available to an improvement step. It varies simple embedding-similarity Top-K retrieval and explicit credit assignment around the Actor/Critic base. The retrieval setting selects schema fragments without the full particle-based grounding process. The credit-only setting scores the interactions without the full Action Graph. A fourth setting combines Top-K retrieval and credit. The complete pipeline additionally supplies the full grounding and graph mechanism.

Figure 3 shows that Top-K retrieval changes the base accuracy by $+0.19$ points for Instruct and -0.07 for Coder. Credit assignment changes it by $+3.39$ and $+0.73$ points. Combining retrieval and credit gives 57.00% and 54.37%; on Coder this is 8.16 points below credit alone. The interaction contrast, $\mathrm{EX}_{\mathrm{both}} - \mathrm{EX}_{\mathrm{retrieval}} - \mathrm{EX}_{\mathrm{credit}} + \mathrm{EX}_{\mathrm{base}}$, is -0.17 for Instruct and -8.09 for Coder. Thus the two simplified components are not additive, especially with Coder.

The full pipeline reaches 63.50% and 64.41%, improving over the Top-K plus credit setting by 6.50 and 10.04 points. A possible explanation is that fragment retrieval supplies locally relevant schema items without their dependencies, while the full system retains competing hypotheses and execution evidence. The observed interaction supports evaluating these mechanisms together; identifying the source of the interaction requires controlled changes to the full grounding pipeline.

5.4 Additional benchmarks and resource use

Table 2 reports the additional benchmark evaluation with the SQL generator identified explicitly. The Coder-generator configuration reaches 80.2% on Spider-DK, 81.7% on Spider-Dev, and 20.7% on Spider2.0-SQLite. The corresponding single-decoder entries are 77.8%, 80.6%, and 10.4%. Its BIRD entry is 58.5%, while 63.5% belongs to the Instruct-generator configuration. Table 1 supplies the BIRD comparison with a shared backbone across all roles.

Table 2: Additional multi-benchmark settings. Backbone labels identify the SQL generator; Table 1 specifies the shared-backbone BIRD configuration. Values are EX (%); ± denotes standard deviation over five runs.

System / generator	Spider-DK	Spider-Dev	BIRD-Dev	Spider2-SQLite
Single decoder / Coder-14B	77.8	80.6	56.9	10.4
RSI-SQL / Coder-14B	80.2 ± 0.5	81.7 ± 0.8	58.5 ± 0.6	20.7 ± 1.1
Single decoder / 14B-Instruct	74.0	69.7	62.1	4.4
RSI-SQL / 14B-Instruct	76.1 ± 0.4	72.9 ± 0.8	63.5 ± 0.6	18.5 ± 1.1

Table 3: Recorded inference cost averaged over 100 BIRD-Dev questions in a separate cost sample. Token counts are totals per query in thousands (K).

Setting	LM calls	Input (K)	Output (K)	Time (s)
ReAct (Coder-14B)	9.2	26.9	17.7	40.3
Majority vote (Coder-14B, $N = 3$)	24.4	77.1	49.8	107.9
RSI-SQL, $H = 2$	34.2	21.8	16.3	45.6
RSI-SQL, $H = 3$	78.4	42.6	31.5	136.8

Table 3 gives the recorded 100-query BIRD cost sample. The $H = 3$ setting uses 78.4 model calls, 74.1 thousand total tokens, and 136.8 seconds per query. The $H = 2$ setting uses 34.2 calls, 38.1 thousand tokens, and 45.6 seconds. Model calls alone obscure the length of their contexts: majority voting uses fewer calls than the $H = 3$ configuration but more total tokens in this sample. These measurements motivate reporting calls, tokens, and wall-clock time together when assessing recursive expansion. Accuracy and cost were recorded on different query sets.

5.5 Failure modes of recursive improvement

Inspection of 50 BIRD failures assigns 38% to schema mismatch, 28% to intent ambiguity, 20% to execution semantics, and 14% to credit misassignment. For 50 Spider2.0-SQLite failures, the corresponding categories are unsupported SQL or dialect handling (34%), irrelevant-schema anchoring (26%), trajectory selection (24%), and unnecessary polling or repair (16%). These proportions are conditional on the inspected failures.

The cases show how a recursive update can reinforce an error. Successful execution can support a semantically incorrect predicate, and particle concentration can increase around the wrong hypothesis. Accumulated credit can also favor an unnecessarily long path. An earlier Spider-Test evaluation recorded 82.1% for the coordinated system versus 88.0% for its Coder single-decoder comparison (Appendix C.2). The value of additional interaction therefore depends on the query and configuration.

The current evidence covers two related 14B backbones, a single run per setting in the aligned BIRD study, and the separately recorded multi-benchmark evaluations. Compute-matched candidate sampling, isolated particle-filter replacement, and coefficient-sensitivity measurements remain necessary to attribute the gains more precisely. The mathematical analysis provides conditional surrogate bounds, while the empirical measurements characterize the observed runs.

6 Related work

Schema grounding and structured generation. Seq2SQL combines structured generation with execution rewards (Zhong et al., 2017). IRNet's intermediate representation and RAT-SQL's relation-aware encoding connect questions to schemas (Guo et al., 2019; Wang et al., 2020). PICARD constrains decoding through incremental parsing (Scholak et al., 2021); RESDSQL separates schema linking from SQL skeleton parsing (Li et al., 2023a). RSI-SQL revises grounding hypotheses and their supporting evidence after database interaction.

Interactive Text-to-SQL and candidate selection. DAIL-SQL studies prompt representation and example selection (Gao et al., 2024); DIN-SQL combines decomposition with self-correction (Pour-

reza & Rafiei, 2023). CHESS and Interactive-T2S retrieve database information through tools (Talaei et al., 2024; Xiong et al., 2024), and MAC-SQL coordinates schema selection, decomposition, and refinement (Wang et al., 2025). OpenSearch-SQL aligns agent inputs and outputs with dynamic few-shot construction (Xie et al., 2025). CHASE-SQL and XiYan-SQL combine diverse generation with learned selection (Pourreza et al., 2025a; Liu et al., 2025). RSI-SQL records typed dependencies linking grounding, observations, and credit to subsequent decisions.

Learning SQL reasoning and data synthesis. OmniSQL scales synthetic training data (Li et al., 2025), while SemanticAgent uses semantic analysis and diagnostic refinement to improve synthesis (Gao et al., 2026). Reasoning-SQL, SQL-R1, and Arctic-Text2SQL-R1 train reasoning models with SQL-oriented rewards (Pourreza et al., 2025b; Ma et al., 2025; Yao et al., 2025). MARS-SQL combines multi-agent reinforcement learning with trajectory ranking (Yang et al., 2025). TRUST-SQL studies tool use over unknown schemas with a POMDP formulation and separate exploration and execution credit (Jian et al., 2026). RSI-SQL updates query-specific grounding weights and heuristic credit with a frozen backbone.

Reasoning, search, and recursive feedback. Chain-of-thought prompting exposes intermediate reasoning (Wei et al., 2022), and self-consistency aggregates sampled paths (Wang et al., 2022). Tree of Thoughts searches over intermediate thoughts; Language Agent Tree Search combines search, actions, and environmental feedback (Yao et al., 2023a; Zhou et al., 2024). Self-Refine iterates between feedback and revision, while Reflexion carries verbal feedback into later trials (Madaan et al., 2023; Shinn et al., 2023). STaR repeatedly trains on successful generated rationales (Zelikman et al., 2022). RSI-SQL carries feedback through explicit grounding hypotheses and interaction dependencies.

Agents, credit, and temporal abstraction. ReAct interleaves reasoning and actions, and Toolformer learns when and how to call external tools (Yao et al., 2023b; Schick et al., 2023). AutoGen supports multi-agent conversations, while multiagent debate refines responses through repeated exchange (Wu et al., 2024; Du et al., 2024). COMA and QMIX address cooperative credit through learned policy gradients and value factorization (Foerster et al., 2018; Rashid et al., 2020). Our heuristic credit annotates observed dependencies for trajectory selection. POMDPs provide the hidden-state perspective (Kaelbling et al., 1998); options, MAXQ, and HIRO study temporal abstraction in reinforcement learning (Sutton et al., 1999; Dietterich, 2000; Nachum et al., 2018). Building on finite-lookahead analysis (Bertsekas, 1995), we separate grounding error from rollout perturbations and retain both intermediate and terminal state effects.

7 Conclusion

RSI-SQL organizes Text-to-SQL inference as recursive self-improvement of a query's reasoning state. Each round revises grounding hypotheses, records execution evidence, and assigns trajectory credit; the resulting state guides subsequent reasoning. The typed Action Graph preserves the dependencies that connect these updates. On the full BIRD development set, the framework reaches 63.50% and 64.41% EX with two Qwen2.5-14B backbones. The component study shows that retrieval and credit can interfere, identifying their coordination as a central design choice in the improvement loop.

AI USE STATEMENT

Generative AI assistance was used to clarify technical concepts, revise the manuscript's structure and language, propose the title, examine the mathematical claims and proofs, interpret supplied experimental tables, verify selected references, and write code to redraw the figures and format the submission. The mathematical revisions include the grounding discrepancy and rollout perturbation bounds. Quantitative figures use the supplied results; no new benchmark runs were performed during this manuscript revision.

ETHICS STATEMENT

The study uses existing Text-to-SQL benchmarks and introduces no new human-subject data. Generated SQL can execute successfully while answering an unintended question. Deployment therefore requires database permissions, controlled execution, and review of queries and results in sensitive settings. The additional model calls also create a computational cost, reported with the experimental results.

REPRODUCIBILITY STATEMENT

Section 3 defines the interaction roles, graph structure, and trajectory selection rule. Appendix B gives the assumptions and proofs for the local analysis; Appendix C distinguishes full-set, repeated-run, and sampled evaluations. Appendix A records the available implementation and compute details. Figure source files and their tabulated inputs accompany the manuscript source.

REFERENCES

M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2): 174–188, 2002. doi: 10.1109/78.978374.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 1995.

Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 11733–11763. PMLR, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence*, 2018.

Yujian Gan, Xinyun Chen, and Matthew Purver. Exploring underexplored limitations of cross-domain text-to-SQL generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 8926–8931. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.702. URL https://aclanthology.org/2021.emnlp-main.702/.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-SQL empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145, 2024.

Qiang Gao, Zhenping Li, Anqi Zhuo, Yingxiao Zhao, Weibo Geng, and Xiaosong Li. SemanticAgent: A semantics-aware framework for text-to-SQL data synthesis. *arXiv preprint arXiv:2604.21414*, 2026. URL https://arxiv.org/abs/2604.21414.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp.

4524–4535. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1444. URL https://aclanthology.org/P19-1444/.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Ai Jian, Xiaoyun Zhang, Eryu Guo, Wanrou Du, Jingqing Ruan, Jiangbo Pei, Weipeng Zhang, Ke Zeng, and Xunliang Cai. TRUST-SQL: Tool-integrated multi-turn reinforcement learning for text-to-SQL over unknown schemas. *arXiv preprint arXiv:2603.16448*, 2026. URL https://arxiv.org/abs/2603.16448.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998. doi: 10.1016/S0004-3702(98)00023-X.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *International Conference on Learning Representations*, 2025.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. RESDSQL: Decoupling schema linking and skeleton parsing for text-to-SQL. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 13067–13075, 2023a. doi: 10.1609/aaai.v37i11.26535.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. OmniSQL: Synthesizing high-quality text-to-SQL data at scale. *Proceedings of the VLDB Endowment*, 18(12):4695–4708, 2025.

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 C.-C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 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*, volume 36, 2023b.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and Jingren Zhou. XiYan-SQL: A novel multi-generator framework for text-to-SQL. *arXiv preprint arXiv:2507.04701*, 2025. URL https://arxiv.org/abs/2507.04701.

Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. SQL-R1: Training natural language to SQL reasoning model by reinforcement learning. *arXiv preprint arXiv:2504.08600*, 2025. URL https://arxiv.org/abs/2504.08600.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019.

Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/e6384711491713d29bc63fc5eeb5ba4f-Paper.pdf.

Mohammadreza Pourreza and Davood Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems*, 2023.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Özcan, and Sercan O. Arik. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *International Conference on Learning Representations*, 2025a.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan O. Arik. Reasoning-SQL: Reinforcement learning with SQL tailored partial rewards for reasoning-enhanced text-to-SQL. *arXiv preprint arXiv:2503.23157*, 2025b. URL https://arxiv.org/abs/2503.23157.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. doi: 10.52202/075280-2997.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9895–9901. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.779. URL https://aclanthology.org/2021.emnlp-main.779/.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2):181–211, 1999.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. CHESS: Contextual harnessing for efficient SQL synthesis, 2024.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 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*, pp. 7567–7578. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.677. URL https://aclanthology.org/2020.acl-main.677/.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 540–557, Abu Dhabi, UAE, January 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.36/.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022. URL https://arxiv.org/abs/2203.11171.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. doi: 10.52202/068431-1800.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *First conference on language modeling*, 2024.

Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. OpenSearch-SQL: Enhancing text-to-SQL with dynamic few-shot and consistency alignment. *arXiv preprint arXiv:2502.14913*, 2025. URL https://arxiv.org/abs/2502.14913.

Guanming Xiong, Junwei Bao, Hongfei Jiang, Yang Song, and Wen Zhao. Multi-turn interactions for text-to-SQL with large language models. *arXiv preprint arXiv:2408.11062*, 2024. URL https://arxiv.org/abs/2408.11062.

Haolin Yang, Jipeng Zhang, Zhitao He, Alexander Zhou, and Yi R Fung. Mars-sql: A multi-agent reinforcement learning framework for text-to-sql. *arXiv preprint arXiv:2511.01008*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822. Curran Associates, Inc., 2023a. doi: 10.52202/075280-0517.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Gaurav Nuti, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. Arctic-Text2SQL-R1: Simple rewards, strong reasoning in text-to-SQL. *arXiv preprint arXiv:2505.20315*, 2025. URL https://arxiv.org/abs/2505.20315.

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. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, 2018.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488. Curran Associates, Inc., 2022. doi: 10.52202/068431-1126.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017. URL https://arxiv.org/abs/1709.00103.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 62138–62160. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

A NOTATION AND IMPLEMENTATION DETAILS

A.1 NOTATION

Table 4: Principal symbols and their roles.

Symbol	Meaning
q, D, y	Natural-language question, schema, and output SQL.
z_t, b_t	Hidden task interpretation/configuration and a belief over it.
x_t, o_t, a_t	Observable interaction history, observation, and action.
$\mathcal{M}_t, \mathcal{P}_t, \mathcal{T}_t$	Recursive reasoning state, weighted grounding particles, and scored candidate trajectories.
$\mathcal{G}_t, \kappa(v)$	Typed Action Graph and vertex creation index.
$\pi_t^{(i)}, w_t^{(i)}$	Grounding hypothesis and its normalized weight.
P, ϕ, E	Fixed linear dimension reduction, symbolic grounding map, and grounded-symbol encoder.
$s_t, \widehat{s}_t$	Encodings of complete history and grounded abstraction in the same space.
A_t, ρ_t, δ_t	Regularized grounding operator, residual energy ratio, and abstraction discrepancy.
H, B, M	Rollout depth, candidate-action budget, and particle count.
c_t, C_i, L_i	Interaction credit, accumulated trajectory credit, and trajectory length.
J_t, ξ_t, e_t	Surrogate Jacobian, step perturbation, and state error.
L_R, L_V	Lipschitz constants for surrogate reward and value on the assumed domain.
S_t, D_t	Independent-noise sensitivity sum and dependent-noise L^2 bound.

A.2 REPRESENTATION AND GRAPH OPERATIONS

State extraction serializes the question, schema, candidates, and observations, then applies the frozen backbone, fixed token reduction, and P. Symbol encoding uses the same reduction and projection. The 1,536-dimensional FP32 vectors are cached by exact serialized context. A repair or new observation requires its corresponding encoding.

For $E_R = U\Sigma V^\top$, Eq. (4) gives

$$A_t = U \ \mathrm{diag}\left(\frac{\sigma_j^2}{\sigma_j^2 + \lambda}\right) U^\top.$$

Its eigenvalues lie in $[0, 1)$ when $\lambda > 0$. It contracts selected directions according to their singular values and removes directions orthogonal to their span. Idempotence is unnecessary for Proposition 1, which uses an exact add-and-subtract identity.

For an interaction with contributing edges $\mathcal{E}_t$, let $g_{t,e} \geq 0$ denote their relative relevance. When $\sum_{e \in \mathcal{E}_t} g_{t,e} > 0$, normalized allocation assigns

$$c_{t,e} = c_t \frac{g_{t,e}}{\sum_{e' \in \mathcal{E}_t} g_{t,e'}}.$$

These annotations identify which dependencies receive the interaction credit. The trajectory score in Eq. (8) sums hop credit once per interaction, rather than counting the same hop again for each incident edge. The score is undiscounted; γ belongs to the diagnostic rollout model.

The method specification includes a low-compatibility expansion trigger with threshold 0.01. Normalized weights require a separate scale for that trigger: since $\max_i w_i \geq 1/M$, a condition $\max_i w_i < 0.01$ cannot activate with $M = 64$. Unnormalized compatibility and normalized particle mass must therefore be distinguished when implementing expansion. The ESS rule uses normalized mass.

A.3 AN ILLUSTRATIVE ACTION GRAPH

The example uses `Tryout(student_id, decision)` with values `yes`/`no`. Both candidate predicates execute. The value poll updates the grounding state, which supports replacing `'accepted'` with `'yes'` in the next round. The graph retains the observation behind this refinement, and the question counts tryout records.

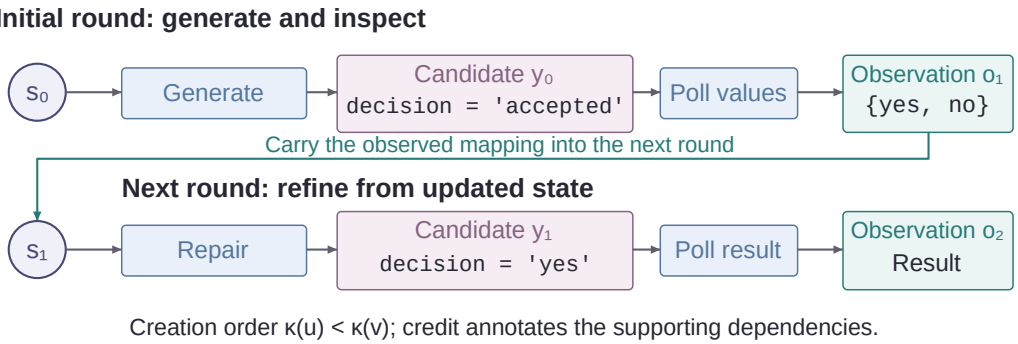


Figure 4: An illustrative recursive update for Figure 1. The first candidate uses an unsupported literal. Polling exposes the stored values, and the next state incorporates that observation before repair. Candidate SQL vertices are distinct from executor observations. Creation indices order the dependencies across rounds; the Critic annotates them with credit.

A.4 Compute environment

The reported experiments used an NVIDIA A800-SXM4-80GB GPU, dual Intel Xeon Platinum 8380 processors at 2.30 GHz, and approximately 1 TB of system memory. The environment used Ubuntu 22.04, NVIDIA driver 545.23.08, CUDA 12.3, cuDNN 9.0, and PyTorch 2.3. SQL execution, evaluation, and trajectory logging run on CPU; backbone inference runs on GPU. No model training is involved. Reproduction additionally requires the stage prompts, model revisions, decoding settings, projection/pooling specification, evaluator versions, and query identifiers used by each run.

B Proofs and diagnostic interpretation

Proof of Proposition 1. Lipschitz continuity and the triangle inequality yield

$$\begin{aligned}|V(s_t) - V(\widehat{s}_t)| &\le L_V \|s_t - \widehat{s}_t\|_2 \\ &\le L_V \big(\|s_t - A_t s_t\|_2 + \|A_t s_t - \widehat{s}_t\|_2\big).\end{aligned}$$

By the definition of ρ_t, the first norm is exactly $\sqrt{\rho_t(\|A_t s_t\|_2^2 + \epsilon)}$. The second is δ_t. This establishes Eq. (10), including the stabilizer. No orthogonality assumption on A_t is required. □

B.1 Accumulated state perturbations

Consider a fixed action sequence $a_{0:H-1}$, a clean surrogate trajectory s_t, and a perturbed trajectory $\widetilde{s}_t$. This coupling holds actions fixed so that transition sensitivity can be separated from a change in the decision rule. Let $e_t = \widetilde{s}_t - s_t$ and

$$e_{t+1} = J_{t+1} e_t + \xi_t, \qquad e_0 = 0, \tag{13}$$

where J_{t+1} is a local transition Jacobian and ξ_t includes the step perturbation and any linearization residual covered by its moment bound.

Lemma 1 (Independent and dependent perturbations)**.** *Conditioned on a clean trajectory, suppose the J_k are fixed and the ξ_t are independent, centered, and satisfy $\mathbb{E}\|\xi_t\|_2^2 \le \sigma^2$. Then*

$$\mathbb{E}\|e_t\|_2^2 \le \sigma^2 S_t, \qquad S_t = \sum_{i=0}^{t-1} \prod_{k=i+2}^{t} \|J_k\|_2^2, \quad S_0 = 0. \tag{14}$$

If the Jacobians and noise are dependent but $\|J_k\|_2 \le a_k$ almost surely and $(\mathbb{E}\|\xi_i\|_2^2)^{1/2} \le \sigma_i$, the valid bound is

$$(\mathbb{E}\|e_t\|_2^2)^{1/2} \le D_t := \sum_{i=0}^{t-1} \sigma_i \prod_{k=i+2}^{t} a_k. \tag{15}$$

The second bound follows from the triangle inequality in L^2 and retains the effect of possible dependence. Applying the first bound requires Jacobians fixed along a clean trajectory and conditionally independent perturbations. The second requires almost-sure envelopes a_k; median sensitivities measured along noisy executions describe a different statistic.

Proof of the independent-noise part of Lemma 1. Condition on the clean trajectory and define $F_{t,i} = J_t J_{t-1} \cdots J_{i+2}$ for $i < t-1$, with $F_{t,t-1} = I$. Unrolling Eq. (13) gives

$$e_t = \sum_{i=0}^{t-1} F_{t,i} \xi_i.$$

The matrices are fixed under this conditioning. Independence and zero conditional mean give $\mathbb{E}\langle F_{t,i}\xi_i, F_{t,j}\xi_j\rangle = 0$ for $i \neq j$. Therefore,

$$\begin{aligned}\mathbb{E}\|e_t\|_2^2 &= \sum_{i=0}^{t-1} \mathbb{E}\|F_{t,i}\xi_i\|_2^2 \\ &\leq \sigma^2 \sum_{i=0}^{t-1} \|F_{t,i}\|_2^2 \leq \sigma^2 \sum_{i=0}^{t-1} \prod_{k=i+2}^{t} \|J_k\|_2^2.\end{aligned}$$

The empty product equals one. This gives Eq. (14). □

Proof of the dependent-noise part of Lemma 1. Let $m_t = (\mathbb{E}\|e_t\|_2^2)^{1/2}$. Minkowski's inequality and the almost-sure deterministic envelope for the Jacobian imply

$$m_{t+1} \leq (\mathbb{E}\|J_{t+1}e_t\|_2^2)^{1/2} + (\mathbb{E}\|\xi_t\|_2^2)^{1/2} \leq a_{t+1}m_t + \sigma_t.$$

Induction from $m_0 = 0$ gives Eq. (15). Neither zero mean nor independence is needed for this part. It can consequently be looser than the independent-noise result. □

Proof of Proposition 2. For an intermediate reward,

$$\mathbb{E}|r(\widetilde{s}_t, a_t) - r(s_t, a_t)| \leq L_R d_t.$$

For the terminal term, adding and subtracting $V(\widetilde{s}_H)$ gives

$$\begin{aligned}\mathbb{E}|V_k(\widetilde{s}_H) - V(s_H)| &\leq \mathbb{E}|V_k(\widetilde{s}_H) - V(\widetilde{s}_H)| + \mathbb{E}|V(\widetilde{s}_H) - V(s_H)| \\ &\leq \|V_k - V\|_\infty + L_V d_H.\end{aligned}$$

Summing the discounted terms proves Eq. (12). Jensen's inequality converts either second-moment bound in Lemma 1 into a valid choice of d_t. □

Consequences for depth selection. Even with $J_k = 0$ at every step, e_H can contain the final perturbation ξ_{H-1}; this explains the terminal-state term $\gamma^H L_V d_H$. Under a uniform envelope $a_k \leq a$ and $\sigma_i \leq \sigma$, the dependent-noise result becomes

$$D_t \leq \sigma \sum_{j=0}^{t-1} a^j = \begin{cases}\sigma(1-a^t)/(1-a), & a \neq 1, \\ \sigma t, & a = 1.\end{cases}$$

This envelope must hold throughout the surrogate trajectory. Average measured norms and symbolic error counts do not establish it.

B.2 Recorded local measurements

The original procedure samples 500 states and 10,000 state pairs. $\widehat{L}_V$ is a median of observed value-difference/distance ratios; the reported 90th-percentile values are 2.34, 2.67, and 3.12 for the three rows. $\widehat{L}_\phi$ summarizes the corresponding abstraction-map ratios. The columns $\widehat{\varepsilon}_{\text{abs}}$ and $\widehat{\delta}$ summarize value discrepancy and representation fidelity, respectively. Central differences with step 10^{-5} provide the per-step sensitivity summary $\widehat{j}$ from noisy rollout states. It is distinct from the cumulative quantity S_t in Eq. (14).

Table 5: Local diagnostic summaries. Ratio summaries and state-error estimates have reported 95% bootstrap intervals. These descriptive statistics are distinct from the worst-case constants in the analysis.

Benchmark	$\widehat{L}_\phi$	$\widehat{L}_V$	$\widehat{\varepsilon}_{\mathrm{abs}}$	$\widehat{\delta}$	$\widehat{j}$
Spider-DK	$.73 \pm .05$	$1.21 \pm .12$	$.089 \pm .009$	$.022 \pm .004$	$.92 \pm .08$
BIRD-Dev	$.68 \pm .06$	$1.35 \pm .15$	$.112 \pm .013$	$.031 \pm .005$	$1.14 \pm .12$
Spider2-SQLite	$.81 \pm .07$	$1.44 \pm .18$	$.156 \pm .018$	$.048 \pm .008$	$1.31 \pm .15$

These measurements have no matched clean-trajectory series or per-benchmark ρ_t column in the available evaluation. Consequently, they cannot numerically validate the clean-trajectory bound or establish which benchmark has greater context dilution. They describe the states sampled by the implementation. The mathematical statements retain explicit assumptions independently of these measurements.

C Evaluation protocols and additional results

Table 6: Evaluation units used by the different evidence sets.

Evidence set	Evaluation unit	Reported statistic
Aligned BIRD and component study	All 1,534 BIRD-Dev queries per setting	One-run EX
MAC-SQL comparison	Five samples of 59 BIRD-Dev queries	Mean $\pm$ sample-run SD
Multi-benchmark system rows	Five runs per reported setting	Mean $\pm$ run SD
Particle study	Spider2.0-SQLite, three runs per particle count	Mean EX and calls
Cost sample	100 BIRD-Dev queries	Mean calls, tokens, and latency
Failure coding	50 failed BIRD and 50 failed Spider2-SQLite cases	Within-sample category fractions

Table 7: MAC-SQL on five samples of 59 BIRD-Dev queries. These sampled results have a different evaluation unit from the full-set study.

System	Qwen2.5-14B-Instruct	Qwen2.5-Coder-14B-Instruct
MAC-SQL	23.05 ± 2.30	35.93 ± 1.66

The full-set component values are 53.59/61.80 for the Actor/Critic base, 53.78/61.73 with Top-K retrieval, 56.98/62.53 with credit, 57.00/54.37 with both, and 63.50/64.41 for the full pipeline (Instruct/Coder). These are parallel configurations around the same base, not successive improvements in a monotonic sequence. For example, adding credit to the retrieval setting changes Coder accuracy by $54.37 - 61.73 = -7.36$ points. A performance difference between the full system and this simplified setting cannot be attributed to a single component.

C.1 Particle-count study

The particle counts $16, 32, 64, 128, 256$ yield EX values $17.2, 19.5, 20.7, 21.0, 20.8$ and mean model calls $52.1, 63.4, 78.4, 142.3, 271.6$. The small accuracy difference between 64 and 128 particles accompanies an 81.5% increase in calls. This study documents an accuracy–computation trade-off for its recorded setting; it does not determine the optimal particle count for every backbone or benchmark.

C.2 Earlier negative evaluations

An earlier Spider-Test evaluation reported 82.1% for the coordinated system, compared with 88.0% for the Coder-14B single decoder. Its inspection of 50 failures assigned 42% to irrelevant-schema

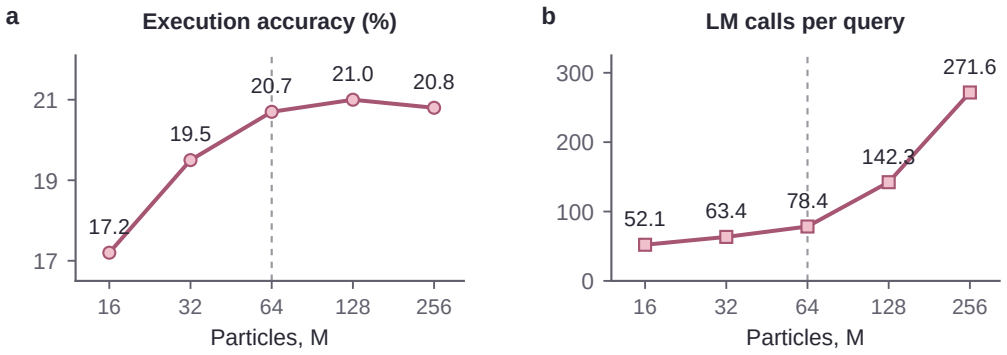


Figure 5: Particle-count study on Spider2.0-SQLite, mean over three runs. Accuracy and model calls use a shared log-base-2 particle-count axis; dashed lines mark $M = 64$. Moving from 64 to 128 particles changes mean accuracy by $+0.3$ percentage points and calls from 78.4 to 142.3. No uncertainty interval was supplied for these means.

anchoring, 28% to trajectory selection, 18% to unnecessary polling, and 12% to credit misassignment. An earlier BIRD comparison also reported 63.5% for the coordinated configuration against 66.1% for the Coder single decoder. These earlier configurations are distinct from the aligned BIRD experiment in Table 1. They retain evidence that extra coordination can be detrimental; they are not combined with the current table to calculate a pooled gain.