LEARNING TASK-STATE ABSTRACTIONS FOR LONG-HORIZON DATABASE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using database agents must track what they know, what they have tried, and which evidence should guide the next action. We study how structured task-state representations support this sequential decision problem. TSADB combines schema topology, query intent, execution traces, and error semantics into a learned state consumed by a lightweight PPO policy; a frozen language model generates tool arguments. On BIRD-SQL and Spider 2.0-Lite, the system improves execution accuracy by 10.0 and 10.4 percentage points over the strongest matched baseline in each main comparison. A separate training-budget-matched comparison with RL-ReAct yields a 7.9-point BIRD gain. A state-by-reward factorial shows complementary contributions: flat-history encoding reduces accuracy by 9.1 points with the progress reward, while removing that reward costs 10.5 points with the learned state. The state effect grows to 18.5 points on tasks requiring at least 16 calls. Across policy classes, supervised control also benefits from the representation, and PPO adds 13.1 points over supervised control on long tasks in a shared 800-task evaluation. The full system retains 74% of its accuracy under combined schema perturbations. Experiments further examine official conversational interaction, transfer with limited error-projection adaptation, termination, and reward construction. These results support learning compact, structured states for long-horizon database control.

1 INTRODUCTION

A database agent makes decisions with an incomplete view of its task. Inspecting a schema reveals possible joins; executing a query tests a hypothesis; an error changes which recovery action is useful. Over a long interaction, the agent must connect these observations to the user's intent and remember which attempts have already failed. Successful tool use therefore depends on both *state estimation*—organizing available evidence—and *action selection*—deciding what to do next.

Large language models support text-to-SQL, enterprise analytics, and data-science agents (Gao et al., 2024; Pourreza et al., 2025; Wang et al., 2025a; Guo et al., 2024; Hong et al., 2024). Recent systems learn multi-agent SQL reasoning, plan observations under budgets, and incorporate execution feedback into training (Chen et al., 2026; Peng et al., 2026a; Liu et al., 2026a; Dai et al., 2026). The controller's representation is therefore an important design choice: raw histories mix schema descriptions, query results, and errors whose relevance becomes harder to recover as interactions grow.

Database interaction provides structure for learning this representation. Schema relations connect differently named columns with similar query roles; error types distinguish inspection, revision, and optimization needs; recent action–observation pairs track recovery outcomes. Together with intent, these signals describe task progress for tool selection. We ask: *how much does a learned state improve sequential control, and how do its effects compare with progress rewards and policy capacity?*

We introduce TSADB (**T**ask-**S**tate **A**bstraction for **D**ata**B**ase agents). Four encoders combine schema structure, query intent, recent execution, and typed errors into a compact state. A masked policy selects among five tools, while a frozen LLM generates their arguments. We learn the state encoder and policy with PPO, using correctness, efficiency, and incremental schema-coverage feedback. Within each rollout, the relevance vector defining the coverage reward is fixed. This architecture

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.

exposes the information used for control and permits direct substitutions of the state, reward, and policy.

The resulting comparisons reveal complementary roles. On BIRD-SQL, replacing the structured state with flat history costs 9.1 points under the progress reward, and removing that reward costs 10.5 points with the structured state. The state effect grows to 18.5 points for tasks requiring at least 16 calls. A supervised router also benefits from the state, while trajectory-level PPO training further improves long-task completion. Increasing the policy head to a LoRA-tuned small LLM adds 0.8 points in the capacity study. Together, these results motivate improving the evidence supplied to a controller alongside its training objective and capacity.

Our contributions are threefold. **(1)** We develop a learned task-state abstraction that integrates relational context and execution feedback for partially observable database interaction. **(2)** We combine this state with lightweight sequential control and a rollout-fixed progress signal, retaining a frozen LLM for argument generation. **(3)** We evaluate the design through matched comparisons, a baseline adaptation audit, a state-by-reward factorial, and a policy-class ladder, with additional evidence on schema shift, conversational interaction, and transfer.

2 Related Work

Learning and observation design for SQL agents. DualSQL jointly trains schema-linking and SQL-generation agents sharing an LLM backbone (Chen et al., 2026). BAP-SQL controls observation formation through query-risk estimation, SQL rewriting, and execution guards (Peng et al., 2026a). SERL-SQL uses selected execution feedback in a training-time hindsight teacher, while SQL-Trail and MTIR-SQL learn multi-turn reasoning with tool feedback (Liu et al., 2026a; Hua et al., 2026; Xu et al., 2025b). ReEx-SQL incorporates execution-aware RL, and Reward-SQL develops process rewards for training and inference (Dai et al., 2026; Zhang et al., 2025b). These methods motivate distinguishing the reasoning model, the observations it receives, and the learning signal. In TSADB, execution evidence enters an explicit learned state consumed by a separate tool-selection policy; the argument-generating LLM remains frozen.

State, memory, and long-horizon control. Bisimulation-based representation learning studies how control-relevant information can be retained while discarding distractors (Zhang et al., 2021). We use database-specific structural encoders and evaluate their effects through controlled substitutions. CompactionRL jointly learns task execution and context compaction, while memory studies for multi-turn SQL examine alternative memory organizations (Li et al., 2026; Tummalapenta & Addanki, 2026). Recent long-horizon RL studies investigate training efficiency and practical recipes (Cheng et al., 2026b; Wu et al., 2026); Agent Lightning, HiPER, GiGPO, ARPO, and StepAgent address training interfaces, credit assignment, or intermediate feedback (Luo et al., 2025; Peng et al., 2026b; Feng et al., 2025; Dong et al., 2025; Song et al., 2025). Our factorial and policy-class controls connect state design to reward construction and sequential learning.

Database workflows and evaluation. ReAct, Reflexion, DAIL-SQL, MAC-SQL, DS-Agent, ReFoRCE, and SDE-SQL provide complementary reasoning, prompting, refinement, exploration, and multi-agent mechanisms (Yao et al., 2023; Shinn et al., 2023; Gao et al., 2024; Wang et al., 2025a; Guo et al., 2024; Deng et al., 2025; Xie et al., 2026). Our audit records adaptation through backbone and tool-interface alignment. BIRD-Interact and ABISS examine interaction and ambiguity, while Spider 2.0-AIFunc extends evaluation to AI-native SQL functions (Huo et al., 2026; Sullutrone et al., 2026; Liu et al., 2026b). We distinguish these settings from Spider 2.0-Lite and report each interaction protocol explicitly. Appendix C discusses enterprise agents, memory, and cross-language methods in greater depth.

3 Problem Formulation

We model a database task as a finite-horizon partially observable decision process. The latent environment state $x_t \in \mathcal{X}$ includes database and task information; executing action a_t induces a transition $x_{t+1} \sim \mathcal{T}(\cdot \mid x_t, a_t)$ and an observation o_{t+1}. The agent acts from its observable history $h_t = (o_0, a_0, o_1, \ldots, a_{t-1}, o_t)$, which includes user requests, inspected schema context, tool outputs,

and errors. A learned encoder f_ϕ maps this history to $\mathbf{s}_t = f_\phi(h_t)$, a compact approximation to the task state for policy learning.

Structured state. The encoder produces four components: schema topology $\mathbf{z}_t^{\text{sch}}$, query intent $\mathbf{z}_t^{\text{qry}}$, recent execution history $\mathbf{z}_t^{\text{exec}}$, and error signals $\mathbf{z}_t^{\text{err}}$. Their concatenation is projected into the policy input:

$$\mathbf{s}_t = W_{\text{proj}}[\mathbf{z}_t^{\text{sch}}; \mathbf{z}_t^{\text{qry}}; \mathbf{z}_t^{\text{exec}}; \mathbf{z}_t^{\text{err}}] + \mathbf{b}_{\text{proj}}. \tag{1}$$

Each component has width 128 and $\mathbf{s}_t \in \mathbb{R}^{256}$. This representation uses complementary summaries of history; its usefulness as an approximate state is evaluated through task performance and component ablations.

Factorized actions. An action $a_t = (u_t, v_t)$ consists of a discrete tool u_t and its generated argument sequence v_t. The tool set is

$$\mathcal{U} = \{\texttt{schema_inspect}, \texttt{sql_exec}, \texttt{query_optimize}, \texttt{error_analyze}, \texttt{result_validate}\}. \tag{2}$$

The policy $\pi_\theta(u_t \mid \mathbf{s}_t)$ selects the tool. The frozen argument LLM receives the user request, the selected tool's instruction prefix, and the revealed execution context (schema elements and tool outputs observed up to step t); it does not receive the TSA state vector or the policy's action logits. This decomposition learns tool-level control while retaining the LLM's in-context argument generation.

Learning objective. For an episode of at most $H = 20$ calls, we maximize $J(\theta, \phi) = \mathbb{E}[\sum_{t=0}^{H-1} \gamma^t r_t]$, with early termination when a candidate passes the operational validation rule. The training reward combines three signals:

$$r_t = \alpha R_{\text{correct},t} + \beta R_{\text{eff},t} + \gamma_r R_{\text{prog},t}. \tag{3}$$

Here γ discounts future rewards and γ_r weights progress. Correctness supplies task supervision, efficiency accounts for execution cost, and progress provides intermediate feedback before a complete answer is available.

4 METHOD

TSADB combines task-state abstraction, a tool-selection policy, and an adaptive interaction loop (Figure 1). The state encoder and policy learn jointly from execution feedback.

4.1 TASK-STATE ABSTRACTION

Schema encoder. At step t, the schema encoder represents schema elements revealed by observations up to t as a directed graph $G_t = (V_t, E_t)$, with columns as nodes and observed foreign-key relations as edges. Column features encode revealed names, data types, and constraints. A two-layer graph convolutional network (GCN) (Kipf & Welling, 2017) updates these features as

$$\mathbf{h}_v^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \frac{W^{(l)} \mathbf{h}_u^{(l)}}{\sqrt{|\mathcal{N}(v)|\,|\mathcal{N}(u)|}}\right), \tag{4}$$

where σ is ReLU. We mean-pool column embeddings within each table T_k and aggregate table vectors with attention: $\mathbf{z}_t^{\text{sch}} = \sum_k \alpha_k \mathbf{h}_{T_k}$, with $\alpha_k = \text{softmax}_k(\mathbf{w}^\top \mathbf{h}_{T_k})$. The resulting representation combines column semantics with relational structure.

Query-intent encoder. A frozen all-mpnet-base-v2 sentence encoder embeds the conversation q_t, followed by a learned projection: $\mathbf{z}_t^{\text{qry}} = W_q\,\text{Embed}(q_t) + \mathbf{b}_q$. This provides the same intent interface across API-based and open-weight backbones and incorporates refinements supplied in multi-turn tasks.

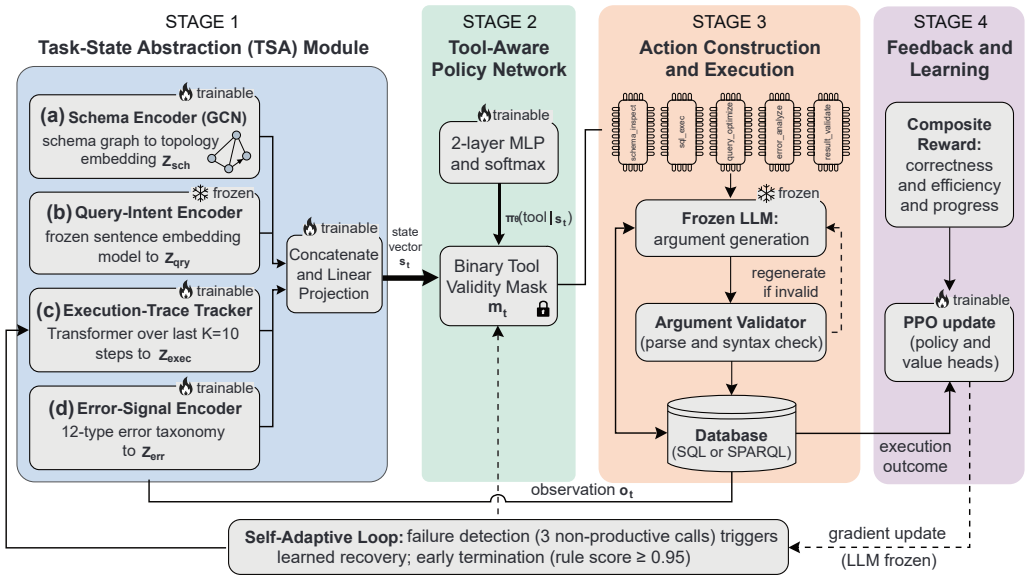

Figure 1: TSADB architecture. Four encoders summarize schema structure, intent, execution history, and errors. A masked policy selects tools; the frozen LLM generates arguments. Execution feedback updates the state and supplies training rewards, while failure signals trigger learned recovery.

Execution-trace tracker. A single-layer, causally masked Transformer (Vaswani et al., 2017) processes the $K = 10$ most recent action–observation pairs:

$$\mathbf{z}_t^{\text{exec}} = \text{MeanPool}\Big(\text{TransEnc}\Big(\{(\mathbf{e}_{a_i}, \mathbf{e}_{o_{i+1}})\}_{i=\max(0,t-K)}^{t-1}\Big)\Big). \tag{5}$$

The fixed trace window bounds the sequence length of this component as interactions grow, while preserving the order of recent attempts and outcomes.

Error-signal encoder. We map errors to 12 categories: syntax, type mismatch, missing table, missing column, ambiguous reference, constraint violation, timeout, empty result, permission denied, connection error, resource limit, and unknown. We maintain their normalized frequencies $\mathbf{f}_t \in \mathbb{R}^{12}$ over the interaction. The representation $\mathbf{z}_t^{\text{err}} = W_e[\mathbf{f}_t; \mathbf{e}_{\text{last_err}}] + \mathbf{b}_e$ combines accumulated failures with the latest error. The taxonomy was derived from 5,214 training-database error messages and covers 98.3% of development errors, with an explicit unknown category. Adding error types expands the frequency vector and requires only W_e adaptation; the schema encoder, trace tracker, and policy can remain frozen. Appendix A lists the categories, and Appendix E evaluates their extension.

4.2 Tool Selection and Adaptive Interaction

The policy uses a two-layer MLP with a 256-unit hidden layer and LayerNorm. Given logits $\mathbf{l}_t$ and a validity mask $\mathbf{m}_t \in \{0,1\}^{|\mathcal{U}|}$, it selects tools according to

$$\pi_\theta(u_t \mid \mathbf{s}_t, \mathbf{m}_t) = \text{softmax}(\mathbf{l}_t + \log \mathbf{m}_t)_{u_t}. \tag{6}$$

For example, `error_analyze` is available after a failure and `result_validate` requires a successful query. Masking assigns zero probability to inapplicable tools. A lightweight validator applies dialect-aware parsing, syntax checks, and deterministic rules to candidate SQL using observable task information. Rejected arguments are regenerated; each retry counts toward the call budget, and validation and regeneration time count toward the time limit.

Recovery and termination. Three consecutive non-productive calls, or repetition of the same error type, trigger state re-encoding and recovery. A call is productive when it returns non-empty, error-free output. The trigger is rule-based; the next action is selected by the learned policy, which may switch

from query execution to schema inspection. For termination, `result_validate` scores execution success, output shape, non-emptiness, dialect and safety signals, and simple question-derived top-k/Boolean conditions. Fixed penalties combine these checks into a deterministic score, and the agent stops at $s_{\text{rule}} \geq 0.95$. This inference procedure operates on the question, schema, candidate query, and observed execution outcome.

4.3 Execution Feedback and Policy Learning

Correctness and efficiency. During training, R_{correct} for `sql_exec` is 1 when its result matches the gold answer and 0 otherwise; other tools receive $+0.1$ for valid, non-empty outputs. Execution cost contributes

$$R_{\text{eff}} = -\lambda_{\text{time}} \log(1 + t_{\text{exec}}) - \lambda_{\text{len}} \mathbb{I}[|\text{query}| > L_{\max}], \tag{7}$$

with time in seconds, $\lambda_{\text{time}} = 0.01$, $\lambda_{\text{len}} = 0.1$, and $L_{\max} = 500$ tokens.

Dense progress. With V indexing columns for the episode's progress reward, let $\mathbf{c}_t \in \{0,1\}^{|V|}$ mark columns inspected or used, and let $\bar{\mathbf{u}}^{\text{rel}} \in \mathbb{R}^{|V|}$ be the softmax-normalized relevance vector computed from query-intent and column embeddings at the episode start. We stop gradients through both inputs and hold this snapshot fixed throughout the rollout. The relevance projection learns during a 200-episode warmup and is then frozen. At each step, we compute

$$R_{\text{prog},t} = \cos(\mathbf{c}_{t+1}, \bar{\mathbf{u}}^{\text{rel}}) - \cos(\mathbf{c}_t, \bar{\mathbf{u}}^{\text{rel}}), \qquad \cos(\mathbf{c}, \mathbf{u}) = \frac{\mathbf{c}^\top \mathbf{u}}{\|\mathbf{c}\|_2 \|\mathbf{u}\|_2 + 10^{-8}}. \tag{8}$$

This rewards progress toward relevant schema coverage before the final query is correct. The denominator also handles the initial zero-coverage vector. Joint validation-set grid search selects $(\alpha, \beta, \gamma_r) = (0.6, 0.15, 0.25)$; Appendix J reports reward dynamics and marginal weight sensitivity.

Joint training. We use PPO (Schulman et al., 2017) with a shared state encoder and separate policy/value heads. The TSA projections, GCN, trace Transformer, and policy/value heads learn from the policy/value objective; the reward relevance projection follows the warmup-and-freeze schedule above. The sentence encoder and argument-generating LLM remain frozen. We refer to this complete configuration as `C-full`. Training uses 2,000 episodes, batch size 32, learning rate 3×10^{-4}, clipping ratio 0.2, and GAE parameter 0.95. Four A100 GPUs require approximately 18 wall-clock hours (72 GPU-hours). Algorithm 1 summarizes the procedure; Appendix A provides additional implementation details.

5 Experiments

5.1 Experimental Setup

Tasks and metrics. The core benchmarks are BIRD-SQL, Spider 2.0-Lite, and DS-Bench (Li et al., 2023; Lei et al., 2024; Jing et al., 2025). BIRD uses the cleaned 1,534-task development split and execution accuracy (EX). Spider 2.0-Lite contains 547 enterprise tasks across BigQuery, Snowflake, and SQLite; we use the official evaluation suite without oracle table hints. DS-Bench has 466 analysis and 74 modeling tasks and uses task success rate (TSR). Its interface replaces SQL execution with Python `data_exec` and inspects file/column metadata. Spider4SPARQL tests SPARQL execution over RDF graphs; BIRD-Interact-Lite adds official conversational evaluation on 300 tasks and a separate supplied-follow-up transfer protocol (Kosten et al., 2025b;a; Huo et al., 2026).

Matched comparison design. We train a single policy on 500 tasks from BIRD-SQL (300) and classic Spider 1.0 (200), with 200 validation tasks (Yu et al., 2018). Unless specified otherwise, systems use frozen GPT-5, the same benchmark-specific tool interface, and caps of 20 calls and 120 seconds, including argument-regeneration attempts. In the matched BIRD comparison, inputs comprise the user question and information exposed through this interface; the benchmark's oracle evidence field is excluded. RL controls share PPO, the composite reward, and a 2,000-episode budget. DS-Bench and BIRD-Interact-Lite are held out; SPARQL uses a frozen policy with 50 episodes of error-projection adaptation. Every policy-ladder row uses the same stratified 800-task BIRD subset, retaining all 16+-call tasks. Full-development-set and subset results are reported separately.

Table 1: Core results with frozen GPT-5. EX and TSR are percentages; ± denotes standard deviation across three seeds. All methods use the shared benchmark-specific interface.

Method	BIRD EX	Spider 2.0-Lite EX	DS-Bench TSR
ReAct	61.8±1.0	24.1±0.9	34.7±1.4
DAIL-SQL	68.1	27.9±0.7	37.1±1.1
Reflexion	65.4±1.1	26.5±0.9	39.8±1.3
MAC-SQL	69.5	31.6±0.6	40.3±1.0
DS-Agent	69.8±0.9	29.8±0.8	45.2±1.2
SDE-SQL-style	69.2	33.1±0.6	40.1±1.1
ReFoRCE	72.0	34.2±0.6	41.0±1.1
TSADB (`C-full`)	**82.0**±0.6	**44.6**±0.5	**55.1**±0.9

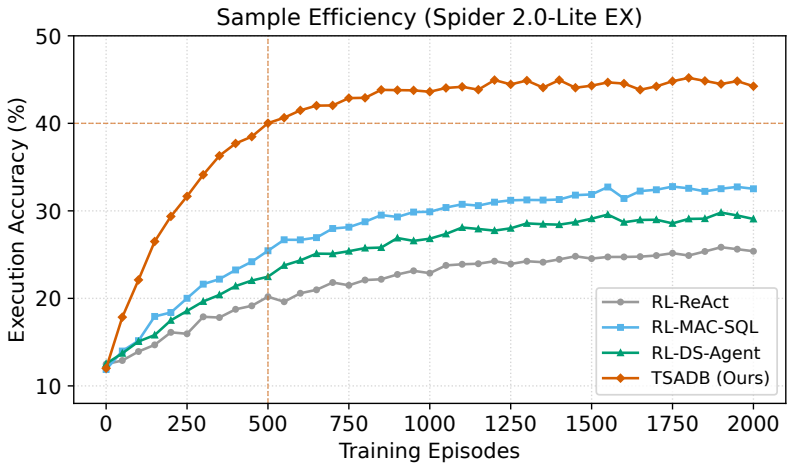

Figure 2: Learning efficiency on Spider 2.0-Lite under the same PPO procedure, composite reward, and 2,000-episode budget.

Baseline audit and uncertainty. The main comparison includes ReAct, DAIL-SQL, Reflexion, MAC-SQL, DS-Agent, ReFoRCE, and SDE-SQL-style. Appendix B records original-configuration reproduction, backbone alignment, and tool alignment for the audited families. It retains workflow decomposition, consensus, and exploratory probes in the adapted interfaces. The SDE-SQL-style label identifies its larger adaptation change. Each benchmark uses a single canonical result set shared by all tables and plots. Experiments use three seeds; tables show seed standard deviations where available. Reported gap intervals use 10,000 paired task-level bootstrap resamples with task and seed correspondence preserved. Appendix A gives dimensions and protocols.

5.2 Task Completion

Table 1 reports performance on the three core benchmarks. TSADB achieves 82.0% BIRD EX, 10.0 points above ReFoRCE. On Spider 2.0-Lite, 44.6% EX exceeds ReFoRCE by 10.4 points, with a reported paired-bootstrap 95% interval of $[7.8, 12.9]$ for the gap. On DS-Bench, 55.1% TSR exceeds DS-Agent by 9.9 points (21.9% relative). These comparisons span SQL generation and multi-step Python operations under the corresponding shared interfaces. The audit in Appendix B makes the stages of baseline adaptation explicit; its tool-matched column agrees with the BIRD comparison above.

5.3 Learning Efficiency and Task Horizon

For the separate training-budget-matched comparison, we give the RL-enhanced baselines the same PPO procedure, composite reward, and episode budget. TSADB reaches 40% Spider 2.0-Lite EX in approximately 500 episodes and converges to 44.6%. RL-enhanced ReAct approaches 26% after

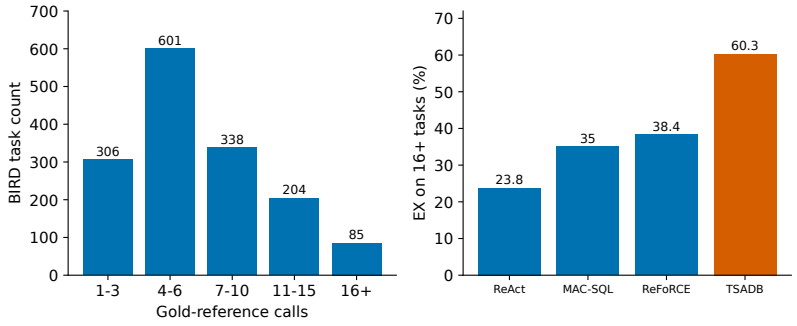

Figure 3: BIRD task counts by canonical gold-solution horizon (left) and EX on the 85 tasks with 16+ calls (right). Horizon is defined independently of the evaluated agent.

Table 2: Execution accuracy (%) under schema perturbations on Spider 2.0-Lite. *Retention* is the fraction of unperturbed accuracy retained under Full Shuffle.

Method	None	Col. Ren.	Tab. Spl.	Full	Retention
ReAct	24.1	18.9	16.2	12.3	51.0%
MAC-SQL	31.6	25.3	22.1	17.4	55.1%
DS-Agent	29.8	24.1	21.5	16.0	53.7%
ReFoRCE	34.2	28.9	25.7	19.5	57.0%
TSADB	44.6	40.8	38.1	33.0	**74.0%**

more than 1,500 episodes, and RL-enhanced MAC-SQL plateaus near 33% (Figure 2). On BIRD, RL-ReAct obtains 74.1% EX and RL-MAC-SQL 71.7%; `C-full` improves over RL-ReAct by 7.9 points (95% paired-bootstrap CI $[5.6, 10.1]$, $p < 0.001$). This comparison keeps the training budget distinct from the non-RL workflow comparison in Table 1.

Evaluating the longest gold-solution horizons highlights the advantage in sustained interaction (Figure 3). On tasks requiring at least 16 calls, it achieves 60.3% success, compared with 38.4% for ReFoRCE and 23.8% for ReAct: gaps of 21.9 and 36.5 points. The BIRD horizon bins contain 306, 601, 338, 204, and 85 tasks, respectively. The 21.9-point gap on the 85 longest tasks has a reported 95% paired-bootstrap interval of $[15.2, 28.3]$ ($p < 0.001$). The long-task advantage is consistent with the role of structured state in tracking progress and recovery across many decisions.

5.4 Schema-Shift Robustness

We perturb Spider 2.0-Lite schemas through column renaming (30% of names), table splitting (20% of large tables), and semantically permissible type changes (25% of column types). Full Shuffle combines all three, affecting 50% of schema elements. Logical semantics are manually checked after perturbation. Under Full Shuffle, TSADB retains 74.0% of its original accuracy (33.0% versus 44.6%), compared with 51.0–57.0% for the baselines. Table splitting changes graph structure as well as query construction, yet TSADB retains 85.4% (38.1% versus 44.6%). Schema inspection updates the graph available to the encoder, while error signals distinguish the recovery actions needed after a mismatch. The encoder ablations below further examine these structural components.

5.5 State, Reward, and Policy Controls

State and reward are complementary. The flat-state control concatenates the eight most recent observations and uses a flat encoder with the same parameter budget; TSA uses its structured schema, intent, error, and ten-pair trace components. Correctness and efficiency feedback are retained in every factorial cell. In Table 3, the state effect is 9.1 points with PR and 6.9 without it. The PR effect is 10.5 points with TSA and 8.3 with flat history. Their difference-in-differences is $82.0 - 72.9 - 71.5 + 64.6 = 2.2$ points, indicating positive complementarity on the EX scale. Both components make substantial contributions to overall completion.

Table 3: State-by-progress-reward factorial. BIRD EX is evaluated overall and on the 16+-call bucket; retention is the fraction of Spider 2.0-Lite accuracy retained under Full Shuffle. PR denotes the progress reward.

Configuration	BIRD EX	BIRD 16+ EX	Shuffle retention
TSA + PR (`C-full`)	82.0	60.3	74%
Flat history + PR	72.9	41.8	58%
TSA, no PR	71.5	47.5	70%
Flat history, no PR	64.6	35.2	53%

Table 4: Policy-class ladder on the same stratified 800-task BIRD subset, including all 16+-call tasks. Controls share the tool interface, evaluation budget, and mask; learned sequential policies use the specified common reward.

Policy	Overall EX	16+ EX
Random valid action	21.7	8.9
Heuristic finite-state router	61.4	29.5
Supervised MLP, flat state	67.9	38.2
Supervised MLP, TSA	73.6	47.2
Contextual bandit, TSA	75.2	46.0
Behavior cloning, TSA	76.0	49.6
GRPO, TSA	79.6	—
PPO, TSA (`C-full`)	82.0	60.3

The effects differ by task condition. On 16+-call tasks, replacing TSA costs 18.5 points with PR, while removing PR costs 12.8 with TSA. Under schema shuffle, those changes reduce retention by 16 and 4 percentage points, respectively. Thus, the structural state is particularly useful when decisions span many steps or schema observations shift. The control contrasts the complete structured representation with recent flat history, including their respective memory organizations.

Representation helps beyond PPO. With supervised MLP control, TSA improves EX from 67.9 to 73.6, a 5.7-point gain (Table 4). PPO improves over supervised TSA control by 8.4 points overall and 13.1 on long tasks; behavior cloning and the contextual bandit lie between supervised control and PPO overall. The stronger long-task performance of PPO is consistent with learning consequences across trajectories. Supervised control uses gold-trajectory cross-entropy, whereas behavior cloning uses successful on-policy trajectories. The random and heuristic routers require no policy training; offline optimization costs are separate from the cost of obtaining trajectories.

Learning budget and policy capacity. On BIRD, the reported learning checkpoints reach 73.6% EX at approximately 180 episodes, 76.9% at 500, and 82.0% at 2,000 (Appendix B); Spider learning curves appear separately in Figure 2. The capacity study replaces the MLP tool policy with a LoRA-tuned small LLM, adding 0.8 EX at four times the training cost. Replacing the schema GCN with an MLP costs 2.8 points, and replacing PPO with GRPO costs 2.4. These substitutions, together with the 9.1-point flat-state drop, favor investing in a useful state representation alongside the controller. The complete component table, including masking, recovery, trace, and error encoders, appears in Appendix I.

Termination and reward construction. Execution-success-only stopping yields 80.1 EX, and removing the termination rule yields 78.4; oracle stopping reaches 82.7. The latter comparison leaves a 0.7-point gap above `C-full`. On 16+-call tasks, removing termination gives 53.9 EX versus 60.3. Termination therefore contributes to completion and efficiency, while the state and reward controls reveal additional gains. Removing the reward-freezing protocol (E8a) yields 81.4, and using frozen off-the-shelf mpnet relevance (E8b) yields 79.8. The 2.2-point E8b gap indicates that useful progress feedback is also available from a fixed relevance signal. Table 5 reports the call and token costs; Appendix B details the reward variants.

Table 5: Termination controls on BIRD. Token cost is normalized to `C-full`; the oracle uses evaluation information and serves as a diagnostic reference.

Stopping configuration	EX	Mean calls	Relative tokens
`C-full` (rule score ≥ 0.95)	82.0	11.2	1.00
Execution success only	80.1	13.0	1.19
No termination rule	78.4	14.9	1.37
Oracle termination	82.7	10.8	0.96

5.6 Interaction, Adaptation, and Cost

Under the official c-Interact user simulator on 300 BIRD-Interact-Lite tasks, TSADB achieves 62.4% task success, compared with 54.8% for RL-ReAct and 51.9% for the MAC-SQL-style control. The separate supplied-follow-up, clarification-free protocol yields 60.7% TSR versus ReFoRCE's 51.6%; Appendix E reports the two protocols separately. On Spider4SPARQL, adapting only the error projection for 10, 25, and 50 episodes produces 52.7, 54.6, and 56.5% EX, compared with 50.0 before adaptation and 57.9 with full retraining. This pattern supports minimal-adaptation transfer through the error interface while retaining the learned policy.

The TSA module takes approximately 16 ms of an 846 ms inference step, under 2% overhead. Mean BIRD task time is 13.6 s, compared with 11.2 s for ReAct and 24.8 s for MAC-SQL. At fixed width, graph processing scales as $O(|V| + |E|)$ and column-state storage as $O(|V|)$; the trace window is bounded. Backbone, embedding, error-extension, and cost studies are retained in Appendices F–H.

Error semantics and local extension. The taxonomy is built from 5,214 training-database errors: mpnet embeddings and HDBSCAN yield 34 clusters, which are manually consolidated into 12 categories including UNKNOWN. The mapper combines regular-expression/keyword rules (71.4% of messages), a string-feature GBDT (23.9%), and an LLM fallback (4.7%). Accuracy is 96.2% on 500 manually checked messages. Coverage reaches 98.7% on training errors, 98.3% on development, 98.0% on test, and 97.4% on two additional held-out databases. This construction gives the controller reusable failure signals while leaving uncommon errors in UNKNOWN. Adding three vendor-specific categories and adapting only W_e for 50 episodes yields 81.7% EX on the associated held-out evaluation, compared with 72.3% without those categories and 81.9% with full retraining. The local update incorporates backend-specific feedback while keeping the policy, schema encoder, and trace tracker fixed.

6 Discussion and Conclusion

TSADB learns a compact state that connects schema relations, intent, execution outcomes, and error semantics to tool selection. The factorial shows substantial contributions from both state and progress feedback; the policy ladder shows that the representation also benefits supervised control and that sequential training further improves long-task completion. Together with schema-shift and interaction results, these findings support structured state learning as a practical component of database agents.

The controls clarify the trade-offs between representation, supervision, and sequential optimization. Reward shaping supplies intermediate feedback, while schema and error structure help organize the evidence available at inference. The factorial interaction and termination controls show why these mechanisms should be evaluated together: each changes a different part of the interaction loop, and their effects depend on the remaining configuration. This suggests a practical evaluation pattern for future agents—compare representations within a fixed controller, then compare controllers over the same representation and task set.

The five-tool interface and error categories reflect relational workflows. Document stores, graph traversal, and changing permissions offer useful tests of broader applicability. The 98.3% development error coverage motivates richer semantics for residual UNKNOWN cases. Service-model variability and the 300-task conversational evaluation call for broader repeatability studies, while the fixed trace window motivates adaptive memory. These directions extend the observed benefits of structured state and lightweight control.

AI USE STATEMENT

Generative AI (OpenAI Codex) assisted with drafting and revising portions of the manuscript, language editing, manuscript organization, literature lookup, LaTeX formatting, and consistency checks. The authors supplied the research methods, experimental results, and clarifications of evaluation protocols. AI assistance supported manuscript preparation. The authors are responsible for the final text, references, and scientific claims.

REPRODUCIBILITY STATEMENT

Section 4 defines the state, tool policy, rewards, and optimization. Appendix A gives the training algorithm, dimensions, and benchmark setup. Appendix B records baseline alignment, canonical results, policy comparisons, reward controls, and taxonomy construction. Appendices E–J provide transfer protocols, backbone studies, costs, detailed ablations, and reward analyses. Tables distinguish seed standard deviations from paired-bootstrap intervals and identify the shared subset used for policy comparisons.

ETHICS STATEMENT

Database agents can expose sensitive records or execute costly and unsafe operations. Our framework includes argument validation and inference-time safety checks. Deployment should pair these mechanisms with access controls, query budgets, and oversight appropriate to the data and operations involved. The evaluation uses the public benchmarks and controlled protocols described in the paper.

REFERENCES

Asim Biswal, Chuan Lei, Xiao Qin, Aodong Li, Balakrishnan Narayanaswamy, and Tim Kraska. AgentSM: Semantic Memory for Agentic Text-to-SQL. arXiv:2601.15709, 2026. URL http://arxiv.org/abs/2601.15709v1.

Saumya Chaturvedi, Aman Chadha, and Laurent Bindschaedler. SQL-of-Thought: Multi-agentic Text-to-SQL with Guided Error Correction. arXiv:2509.00581, 2025. URL http://arxiv.org/abs/2509.00581v2.

Shijie Chen, Yu Gan, Yeounoh Chung, Jiani Zhang, Quannan Li, Sravan Babu Bodapati, Cody J. Greer, Yu Su, and Fatma Ozcan. DualSQL: Text-to-SQL with Multi-Agent Reinforcement Learning, 2026. URL https://arxiv.org/abs/2609.18135.

Xin Cheng, Shuo He, Lang Feng, HaiYang Xu, Ming Yan, Lei Feng, and Bo An. Beyond Trajectory-Level Attribution: Graph-Based Credit Assignment for Agentic Reinforcement Learning. arXiv:2605.26684, 2026a. URL http://arxiv.org/abs/2605.26684v2.

Zelei Cheng, Amritansh Mishra, Sambit Sahu, and William Campbell. Efficient Reinforcement Learning for Long-Horizon Tool-Use Agentic Tasks, 2026b. URL https://arxiv.org/abs/2608.10357.

Chengxiao Dai, Yue Xiu, and Dusit Niyato. From Graph Retrieval to Schema Realization: Counterfactual Validation for Text-to-SPARQL over Heterogeneous Knowledge Graphs. arXiv:2508.01815, 2025. URL http://arxiv.org/abs/2508.01815v3.

Yaxun Dai, Wenxuan Xie, Xialie Zhuang, Tianyu Yang, Ziyi Liu, Haiqin Yang, Yiying Yang, Yuhang Zhao, Pingfu Chao, and Wenhao Jiang. ReEx-SQL: Reasoning with Execution-Aware Reinforcement Learning for Text-to-SQL. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 824–847. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.35. URL https://aclanthology.org/2026.acl-long.35/.

DeepSeek-AI. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*, 2025.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. ReFoRCE: A Text-to-SQL Agent with Self-Refinement, Consensus Enforcement, and Column Exploration. arXiv:2502.00675, 2025. URL http://arxiv.org/abs/2502.00675v5.

Guanting Dong et al. Agentic Reinforced Policy Optimization. *arXiv preprint arXiv:2507.19849*, 2025.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-Group Policy Optimization for LLM Agent Training. In *Advances in Neural Information Processing Systems*, 2025.

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. doi: 10.14778/3641204.3641221. URL https://www.vldb.org/pvldb/vol17/p1132-gao.pdf.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 16813–16848. PMLR, 2024.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Lingyao Zhang, Mingchen Zhuge, et al. Data Interpreter: An LLM Agent for Data Science. In *Advances in Neural Information Processing Systems*, 2024.

Harper Hua, Zhen Han, Zhengyuan Shen, Jeremy Lee, Patrick Guan, Qi Zhu, Sullam Jeoung, Yueyan Chen, Yunfei Bai, Shuai Wang, Vassilis Ioannidis, and Huzefa Rangwala. SQL-Trail: Multi-Turn Reinforcement Learning with Interleaved Feedback for Text-to-SQL, 2026. URL https://arxiv.org/abs/2601.17699.

Zhenyu Huang et al. Reinforcing LLM Agents via Policy Optimization with Action Decomposition. *International Conference on Learning Representations*, 2025.

Nan Huo, Xiaohan Xu, Jinyang Li, Per Jacobsson, Shipei Lin, Bowen Qin, Binyuan Hui, Xiaolong Li, Ge Qu, Shuzheng Si, Linheng Han, Edward Alexander, Xintong Zhu, Rui Qin, Ruihan Yu, Yiyao Jin, Feige Zhou, Weihao Zhong, Yun Chen, Hongyu Liu, Chenhao Ma, Fatma Ozcan, Yannis Papakonstantinou, and Reynold Cheng. BIRD-INTERACT: Re-imagining Text-to-SQL Evaluation for Large Language Models via Lens of Dynamic Interactions. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2510.05318.

Yan Ji, Bo Xu, Jie Shi, Jiaqing Liang, Deqing Yang, Y. J. Mao, Hai Chen, and Yanghua Xiao. Skeletons Matter: Dynamic Data Augmentation for Text-to-Query. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1214–1236. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.64. URL https://aclanthology.org/2025.emnlp-main.64.pdf.

Chuang Jiang, Mingyue Cheng, Xiaoyu Tao, Qingyang Mao, Jie Ouyang, and Qi Liu. TableMind: An Autonomous Programmatic Agent for Tool-Augmented Table Reasoning. *arXiv preprint arXiv:2509.06278*, 2025a.

Pengcheng Jiang, Jiacheng Lin, Zhiyi Shi, Zifeng Wang, He Li, Yichen Wu, Ming Zhong, Peiyang Song, Qizheng Zhang, Heng Wang, Xueqiang Xu, Hanmei Xu, Pengrui Han, Dylan Zhang, Jiashuo Sun, Chaoqi Yang, Kun Qian, Tian Wang, Changran Hu, Manling Li, Quanzheng Li, Hao Peng, Sheng Wang, Jingbo Shang, Chao Zhang, Jiaxuan You, Liyuan Liu, Pan Lu, Yu Zhang, Heng Ji, Yejin Choi, Dawn Song, Jimeng Sun, and Jiawei Han. Adaptation of Agentic AI: A Survey of Post-Training, Memory, and Skills. arXiv preprint, 2025b. URL https://arxiv.org/pdf/2512.16301.

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. DSBench: How Far Are Data Science Agents from Becoming Data Science Experts? In *International Conference on Learning Representations*, 2025.

Thomas N Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. *International Conference on Learning Representations*, 2017.

Catherine Kosten, Davide Lanti, Philippe Cudré-Mauroux, and Kurt Stockinger. Bootstrapping text-to-SQL resources for knowledge graph question answering. In *2025 IEEE Swiss Conference on Data Science (SDS)*. IEEE, 2025a. doi: 10.1109/sds66131.2025.00009. URL https://doi.org/10.1109/sds66131.2025.00009.

Catherine Kosten, Farhad Nooralahzadeh, and Kurt Stockinger. Evaluating the effectiveness of prompt engineering for knowledge graph question answering. *Frontiers in Artificial Intelligence*, 7:1454258, 2025b. doi: 10.3389/frai.2024.1454258. URL https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2024.1454258/pdf.

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. arXiv:2411.07763, 2024. URL http://arxiv.org/abs/2411.07763v2.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin 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*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html.

Xinzhe Li. A Review of Prominent Paradigms for LLM-Based Agents: Tool Use, Planning, and Feedback Learning. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 9760–9779, 2025.

Yujiang Li, Zhenyu Hou, Yi Jing, Jie Tang, and Yuxiao Dong. CompactionRL: Reinforcement Learning with Context Compaction for Long-Horizon Agents, 2026. URL https://arxiv.org/abs/2607.05378.

Chengxi Liao, Tao Xu, Zulong Chen, Chuanfei Xu, Yiyan Wang, Xinyun Wang, Yanlong Zhang, Xiaojun Chen, Zhibo Yang, and Zeyi Wen. EntSQL: A Benchmark for Grounding Text-to-SQL in Long-Context Enterprise Knowledge. arXiv:2606.03363, 2026. URL http://arxiv.org/abs/2606.03363v3.

Tao Liu, Tao Feng, Xiangheng Li, Jinwang Song, Yifan Li, Xiaoqing Cheng, Dixuan Zhang, Siquan Li, Lin Lan, Hongying Zan, Kunli Zhang, and Chao Wu. SERL-SQL: Selective Hindsight Distillation for Text-to-SQL Reinforcement Agentic Learning, 2026a. URL https://arxiv.org/abs/2608.00485.

Tianyang Liu, Canwen Xu, Fangyu Lei, Nikki Lijing Kuang, Jixuan Chen, Tao Yu, Julian McAuley, Zhewei Yao, and Yuxiong He. Spider 2.0-AIFunc: Extending Real-World Text-to-SQL to AI-Native SQL Workflows, 2026b. URL https://arxiv.org/abs/2607.06229.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent Lightning: Train ANY AI Agents with Reinforcement Learning. *arXiv preprint arXiv:2508.03680*, 2025.

Jaehyun Nam, Jaewoong Yoon, Jiayuan Chen, Jinwoo Shin, Sercan O. Arik, and Tomas Pfister. DS-STAR: Data Science Agent via Iterative Planning and Verification. *arXiv preprint arXiv:2509.21825*, 2025.

Chong Peng, Pin Qian, Su Wang, Yihang Chen, and Varun Sah. BAP-SQL: Budget-Aware Observation Planning for Agentic Text-to-SQL, 2026a. URL https://arxiv.org/abs/2608.02876.

Jiangweizhi Peng, Yuanxin Liu, Ruida Zhou, Charles Fleming, Zhaoran Wang, Alfredo Garcia, and Mingyi Hong. HiPER: Hierarchical Reinforcement Learning with Explicit Credit Assignment for Large Language Model Agents. arXiv:2602.16165, 2026b. URL http://arxiv.org/abs/2602.16165v2.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. A Survey on Employing Large Language Models for Text-to-SQL Tasks. *ACM Computing Surveys*, 58(2):1–37, 2025. doi: 10.1145/3737873. URL https://arxiv.org/pdf/2407.15186.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. *Advances in Neural Information Processing Systems*, 36, 2023.

Gundeep Singh, Parsa Kavehzadeh, Jing Xia, Xue-Yong Fu, Julien Bouvier Tremblay, Md Tahmid Rahman Laskar, Vincent Lum, and Shashi Bhushan TN. Beyond Text-to-SQL: An Agentic LLM System for Governed Enterprise Analytics APIs. arXiv:2605.21027, 2026. URL http://arxiv.org/abs/2605.21027v2.

Sithursan Sivasubramaniam, Cedric Osei-Akoto, Yi Zhang, Kurt Stockinger, and Jonathan Fuerst. SM3-Text-to-Query: Synthetic Multi-Model Medical Text-to-Query Benchmark. arXiv preprint, 2024. URL https://arxiv.org/pdf/2411.05521.

Yifan Song, Zhirui Deng, Zhicheng Dou, Yutao Zhu, Ji-Rong Wen, Ruibin Xiong, Mang Wang, and Weipeng Chen. From Novice to Expert: LLM Agent Policy Optimization via Step-wise Reinforcement Learning. In *Proceedings of The Web Conference*, 2025.

Giovanni Sullutrone, Luca Sala, Sania Aftar, Georgia Koutrika, and Sonia Bergamaschi. ABISS: Evaluating Text-to-SQL Systems Through Agent Interaction, 2026. URL https://arxiv.org/abs/2607.23340.

Linzhuang Sun, Tianyu Guo, Hao Liang, Yuying Li, Qifeng Cai, Jingxuan Wei, Bihui Yu, Wentao Zhang, and Bin Cui. Rethinking Text-to-SQL: Dynamic Multi-turn SQL Interaction for Real-world Database Exploration, 2025. URL https://arxiv.org/abs/2510.26495.

Ravi Kumar Tummalapenta and Suman Addanki. Memory Architectures for Multi-Turn Text-to-SQL: A Benchmark and Empirical Study, 2026. URL https://arxiv.org/abs/2605.26394.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30, 2017.

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 *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 540–557. Association for Computational Linguistics, 2025a.

Yihan Wang, Peiyu Liu, and Xin Yang. LinkAlign: Scalable Schema Linking for Real-World Large-Scale Multi-Database Text-to-SQL. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 977–991. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.emnlp-main.51. URL https://aclanthology.org/2025.emnlp-main.51.pdf.

Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. PlanGenLLMs: A Modern Survey of LLM Planning Capabilities. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19497–19521. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.958. URL https://aclanthology.org/2025.acl-long.958.pdf.

Xixi Wu, Qianguo Sun, Ruiyang Zhang, Chao Song, Junlong Wu, Yiyan Qi, and Hong Cheng. Demystifying Reinforcement Learning for Long-Horizon Tool-Using Agents: A Comprehensive Recipe, 2026. URL https://arxiv.org/abs/2603.21972.

Wenxuan Xie, Yaxun Dai, and Wenhao Jiang. SDE-SQL: Enhancing Text-to-SQL Generation in Large Language Models via Self-Driven Exploration with SQL Probes. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2507–2525. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.116. URL https://aclanthology.org/2026.acl-long.116.pdf.

Weikai Xu, Chengrui Huang, Shen Gao, and Shuo Shang. LLM-Based Agents for Tool Learning: A Survey. *Data Science and Engineering*, 10(4):533–563, 2025a. doi: 10.1007/s41019-025-00296-9. URL https://link.springer.com/content/pdf/10.1007/s41019-025-00296-9.pdf.

Zekun Xu, Siyu Xia, Chuhuai Yue, Jiajun Chai, Mingxue Tian, Xiaohan Wang, Wei Lin, Haoxuan Li, and Guojun Yin. MTIR-SQL: Multi-turn Tool-Integrated Reasoning Reinforcement Learning for Text-to-SQL, 2025b. URL https://arxiv.org/abs/2510.25510.

Kaiyue Yang, Yuyan Bu, Jingwei Yi, Yuchi Wang, Biyu Zhou, Juntao Dai, Songlin Hu, and Yaodong Yang. When Lower Privileges Suffice: Investigating Over-Privileged Tool Selection in LLM Agents. arXiv:2606.20023, 2026a. URL http://arxiv.org/abs/2606.20023v2.

Xiao Yang et al. R&D-Agent: Automating Data-Driven AI Solution Building through LLM-Powered Automated Research, Development, and Evolution. arXiv preprint arXiv:2505.14738, 2025.

Zhaorui Yang, Huawei Zheng, Sen Yang, Yuhui Zhang, Haoxuan Li, Zhizhen Yu, Xuan Yi, Chen Hou, Defeng Xie, Chao Hu, Minfeng Zhu, Dazhen Deng, Haozhe Feng, Danqing Huang, Yingcai Wu, Peng Chen, and Wei Chen. ProSPy: A Profiling-Driven SQL-Python Agentic Framework for Enterprise Text-to-SQL. arXiv:2606.05836, 2026b. URL http://arxiv.org/abs/2606.05836v1.

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*, 2023.

Jing Ye, Yiwen Duan, Yonghong Yu, Victor Ma, Yang Gao, and Xing Chen. Beyond Text-to-SQL: Can LLMs Really Debug Enterprise ETL SQL? arXiv:2601.18119, 2026. URL http://arxiv.org/abs/2601.18119v1.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. A Survey on Evaluation of LLM-based Agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26690–26714. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1330. URL https://aclanthology.org/2026.findings-acl.1330.pdf.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning Invariant Representations for Reinforcement Learning without Reconstruction. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2006.10742.

Chenchen Zhang. From Reasoning to Agentic: Credit Assignment in Reinforcement Learning for Large Language Models. arXiv:2604.09459, 2026. URL http://arxiv.org/abs/2604.09459v2.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhongzhi Li, Xiangyuan Xue, Yijiang Li, Yifan Zhou, Yang Chen, Chen Zhang, Yutao Fan, Zihu Wang, Songtao Huang, Francisco Piedrahita-Velez, Yue Liao, Hongru Wang, Mengyue Yang, Hong Ji, Jun Wang, Shuicheng Yan, Philip H. S. Torr, and Lei Bai. The Landscape of Agentic Reinforcement Learning for LLMs: A Survey. arXiv preprint, 2025a. URL https://arxiv.org/pdf/2509.02547.

Yuxin Zhang, Meihao Fan, Ju Fan, Mingyang Yi, Yuyu Luo, Guoliang Li, Bin Wu, and Wenchao Zhou. Reward-SQL: Boosting Text-to-SQL via Stepwise Execution-Aware Reasoning and Process-Supervised Rewards, 2025b. URL https://arxiv.org/abs/2505.04671.

A Implementation and Evaluation Details

A.1 Training and inference

Algorithm 1 gives the joint optimization procedure.

Algorithm 1 TSADB Training via PPO

Require: Training tasks $\mathcal{D}$, frozen LLM $\mathcal{L}$, learning rate η
Initialize encoder f_ϕ, policy π_θ, and value head V_ψ
for episode $= 1$ to N_{ep} **do**
  Sample a task from $\mathcal{D}$; initialize history h_0 and representation $\mathbf{s}_0 = f_\phi(h_0)$
  Snapshot relevance $\bar{\mathbf{u}}^{\text{rel}}$ with gradients stopped on both inputs
  for each step $t < H$ with remaining call budget **do**
    Tool mask $\mathbf{m}_t$ from $\mathbf{s}_t$
    Sample tool $\sim \pi_\theta(\cdot \mid \mathbf{s}_t, \mathbf{m}_t)$
    Generate tool arguments with $\mathcal{L}$; validate and form action a_t
    Execute a_t; observe o_{t+1}
    Reward r_t via Equation (3)
    $h_{t+1} \leftarrow (h_t, a_t, o_{t+1})$; $\mathbf{s}_{t+1} \leftarrow f_\phi(h_{t+1})$
    if early termination **then**
      break
    end if
  end for
  Freeze the relevance projection after the 200-episode warmup
  GAE advantages; PPO update of encoder ϕ, policy θ, and value head ψ
end for
return Trained encoder f_ϕ and tool policy π_θ

Policy parameterization. The policy's two-layer MLP maps the projected state through a ReLU hidden layer to tool logits, $\mathbf{l}_t = W_2\,\mathrm{ReLU}(W_1\mathbf{s}_t + \mathbf{b}_1) + \mathbf{b}_2$, with LayerNorm in the implementation. The validity mask is applied before softmax as in Equation (6). The state projection in Equation (1) has $W_{\text{proj}} \in \mathbb{R}^{256\times 512}$ and $\mathbf{b}_{\text{proj}} \in \mathbb{R}^{256}$, combining four 128-dimensional components. The policy and value heads share this state encoder during training.

Error categories. The twelve categories are syntax, type mismatch, missing table, missing column, ambiguous reference, constraint violation, timeout, empty result, permission denied, connection error, resource limit, and unknown. They were derived from MySQL, PostgreSQL, and SQLite errors in BIRD-SQL and Spider training databases. An extension by k types changes the frequency vector to $\mathbb{R}^{12+k}$ and adapts its projection W_e.

Argument and result validation. Tool prefixes specify argument formats and examples. Candidate SQL is parsed and checked before execution; rejected outputs are regenerated. Each regeneration attempt consumes one call from the 20-call allowance, and its validation and generation time contributes to the 120-second limit. The matched comparisons apply the same accounting to retries. At termination, nonfatal check failures receive fixed score penalties, while unavailable execution,

unsafe SQL, or execution failure receive low fixed scores. The threshold is 0.95. Gold answers provide training rewards and evaluation labels; the operational validator uses inference-time information.

A.2 BENCHMARKS AND PROTOCOLS

Benchmarks We evaluate on five benchmarks that together probe complex SQL synthesis, data-science automation, cross-query-language generalization, and multi-turn interaction.

BIRD-SQL (Li et al., 2023) contains 12,751 text-to-SQL pairs across 95 databases spanning 37 domains, with real-world noisy data and domain-knowledge description files. We use the cleaned development split (1,534 examples). The canonical matched comparison excludes the oracle evidence field from method inputs; methods receive the user question and information exposed through the shared interface. Canonical EX results appear in Table 1; available VES values for unchanged configurations appear in Appendix B.

Spider 2.0 (Lei et al., 2024) evaluates language models on real-world enterprise text-to-SQL workflows involving complex cloud or local data across multiple database systems and diverse transformation operations. We evaluate the official 547-task Spider 2.0-Lite benchmark using the official Spider 2.0 evaluation suite without oracle table hints. Spider 2.0-Lite covers BigQuery, Snowflake, and SQLite backends. Absolute accuracies are far lower than on classic academic Spider (GPT-4o reports approximately 10%), making this a demanding enterprise stress test; we additionally construct a schema-perturbation variant for robustness evaluation.

DS-Bench (Jing et al., 2025) provides 466 data-analysis and 74 data-modeling tasks from ModelOff and Kaggle, requiring multi-step reasoning over large files with diverse operations. We report task success rate (TSR). For DS-Bench, the five tools are adapted: `sql_exec` is replaced by a `data_exec` tool that runs Python code for data loading, cleaning, and transformation, while `schema_inspect` inspects file structure and column metadata. All baselines are given the same adapted tool set; text-to-SQL-specific baselines (MAC-SQL, DAIL-SQL, ReFoRCE, SDE-SQL-style) use `data_exec` for their code-generation steps. This ensures fair comparison while testing whether TSADB's state-abstraction and policy-learning mechanism generalizes beyond pure SQL.

Spider4SPARQL tests cross-query-language generalization by requiring natural-language-to-SPARQL translation over RDF knowledge graphs (Kosten et al., 2025b;a). We adapt TSADB's tool set so that `sql_exec` issues SPARQL and extend the error taxonomy with three graph-specific error types (Appendix E). We report EX.

BIRD-Interact-Lite (Huo et al., 2026) evaluates dynamic multi-turn text-to-SQL interactions and directly stresses the query-intent encoder and execution-trace tracker. We report official c-Interact task success on 300 tasks using the user simulator. A separate custom clarification-free transfer protocol uses the same-sized benchmark setting: follow-up requests are supplied from the gold interaction trajectories, but the agent cannot request clarification and retains the standard five-tool action space. We report multi-turn TSR under this five-tool, supplied-follow-up protocol; official c-Interact and a-Interact use their respective clarification and action interfaces.

Schema perturbation protocol We construct four perturbation levels on the official Spider 2.0-Lite benchmark. *Column Rename* replaces 30% of column names with semantically equivalent alternatives. *Table Split* decomposes 20% of large tables into two related tables connected by a new foreign key. *Type Change* modifies 25% of column data types where semantically permissible. *Full Shuffle* applies all three simultaneously, affecting 50% of schema elements. We manually verify that all perturbed databases preserve logical semantics.

Baselines We compare against seven baselines: **ReAct** (Yao et al., 2023) (thought–action–observation loop with the same tool suite), **DAIL-SQL** (Gao et al., 2024) (optimized few-shot prompting), **MAC-SQL** (Wang et al., 2025a) (three-agent workflow), **DS-Agent** (Guo et al., 2024) (case-based reasoning), **Reflexion** (Shinn et al., 2023) (verbal self-reflection), and two recent state-of-the-art agentic systems: **ReFoRCE** (Deng et al., 2025) (a strong published self-refinement baseline with column exploration) and **SDE-SQL-style** (Xie et al., 2026) (self-driven exploration with SQL probes). All baselines use the same backbone LLM and the same benchmark-specific tool suite for fair comparison.

Backbone LLMs Unless stated otherwise, all systems use a frozen **GPT-5** backbone. To isolate framework-level from model-level effects, Appendix G reports a backbone-sensitivity study across GPT-5, GPT-5.5, and three open-weight models (DeepSeek-V3, Qwen2.5-72B, Llama-3.1-70B), together with a fine-tuning study.

Training protocol We train a *single* policy on 500 tasks from BIRD-SQL (300) and classic Spider 1.0 (Yu et al., 2018) (200) training splits, with 200 tasks for validation. DS-Bench and BIRD-Interact-Lite are held out: DS-Bench tests transfer through the adapted `data_exec` tool, while BIRD-Interact-Lite uses the same policy under the custom clarification-free protocol. Spider4SPARQL uses the frozen policy with only W_e fine-tuned for 50 episodes (Appendix E), testing transfer with minimal adaptation.

Model dimensions The four TSA component widths are 128 and are projected to $d{=}256$. The GCN has two layers (hidden 128); the execution-trace Transformer has four heads (hidden 128); the policy MLP has a 256-unit hidden layer with LayerNorm. Query intent uses a frozen all-mpnet-base-v2 sentence embedding (API-compatible across all backbones). Column features use GloVe-300d (trade-off quantified in Appendix I). Inference allows $H{=}20$ calls, including regeneration attempts, within 120s. Experiments use three seeds. Tables report seed standard deviations where available; paired-bootstrap intervals are reported for the specified contrasts in Appendix B.

B Baseline Audit and Additional Controlled Experiments

B.1 Baseline adaptation and canonical results

Table 6 traces the baseline audit from reproduction in the original configuration to backbone matching and tool matching. The original-configuration column reports experimental reproductions; external published scores depend on each paper's benchmark, model, and evaluation setup. In the shared interface, MAC-SQL preserves its workflow decomposition, ReFoRCE preserves refinement and consensus, and SDE-SQL-style wraps exploratory probes through the supported query-execution interface. The tool-matched results are also used in the BIRD main table for the corresponding method families. RL-ReAct and RL-MAC-SQL are separate trained controls.

Table 6: BIRD baseline adaptation audit (EX, %). The backbone-to-tool change is computed within a row. Standard deviations are shown for original and backbone-matched reproductions; the tool-matched column contains the canonical means.

Baseline family	Original config.	Backbone-matched	Tool-matched	Change
RL-ReAct	74.1±0.9	74.9±0.8	74.1	−0.8
RL-MAC-SQL	71.2±1.1	72.5±0.9	71.7	−0.8
MAC-SQL	69.4±1.0	70.3±0.8	69.5	−0.8
ReFoRCE	71.9±1.2	73.2±1.0	72.0	−1.2
DAIL-SQL	67.6±0.7	68.8±0.6	68.1	−0.7
SDE-SQL-style	70.4±1.1	71.6±0.9	69.2	−2.4

Backbone-to-tool changes range from -0.7 to -1.2 points for five audited families. SDE-SQL changes by -2.4 points and is labeled SDE-SQL-style to identify the adapted implementation. These observed changes quantify the effect of interface alignment in the evaluated runs. The BIRD main-table maximum among the seven baseline families is ReFoRCE's 72.0; RL-ReAct's 74.1 belongs to the separate RL comparison.

Efficiency scores. The available BIRD VES results are 57.9±1.2 for ReAct, 61.7±1.0 for Reflexion, 65.9±1.1 for DS-Agent, and 78.6±0.7 for `C-full`. The other audited baseline configurations are evaluated by EX in Table 1.

B.2 Uncertainty and task horizon

The reported confidence intervals use 10,000 task-level paired-bootstrap resamples, preserving correspondence between methods and seeds. The BIRD gap against RL-ReAct is 7.9 points, with a

95% interval of $[5.6, 10.1]$ and $p < 0.001$. The Spider 2.0-Lite gap against ReFoRCE is 10.4 points, with interval $[7.8, 12.9]$. These intervals describe paired differences; seed standard deviations in the main table describe variation across training/evaluation runs.

Horizon is determined independently of an evaluated agent by decomposing a gold solution into a canonical sequence of tool calls. It is a reference measure of task length, and an agent can use a different number of calls. The BIRD bins 1–3, 4–6, 7–10, 11–15, and 16+ contain 306, 601, 338, 204, and 85 tasks, totaling 1,534. The 16+ difference between `C-full` and ReFoRCE is 21.9 points ($[15.2, 28.3]$, $p < 0.001$). The policy ladder retains these 85 tasks in a stratified 800-task subset; every ladder row uses this same subset. The detailed ablation table separately defines long horizon as 10+ calls.

B.3 State controls and learning budget

The flat-history control concatenates the most recent eight observations and matches the state-encoder parameter budget. It retains the correctness and efficiency rewards, tool mask, and inference budget. The structured state additionally organizes schema topology, current intent, cumulative error frequencies, and the ten most recent action–observation pairs. The factorial therefore compares these two representation designs with and without the same progress-reward component. Overall state effects are 9.1 points with PR and 6.9 without; reward effects are 10.5 with TSA and 8.3 with flat history. The positive 2.2-point interaction indicates complementarity on the accuracy scale.

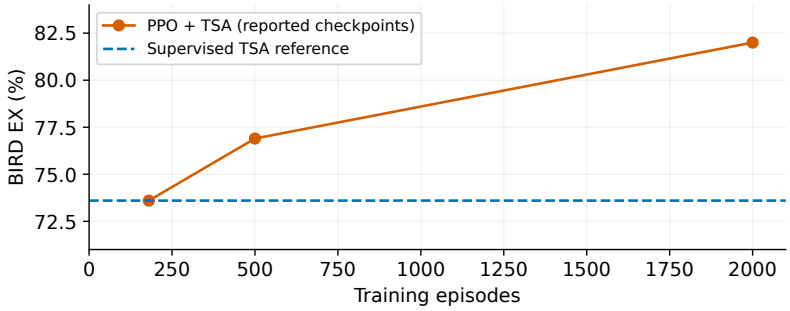

Figure 4: Reported BIRD PPO checkpoints: approximately 180 episodes (73.6 EX), 500 (76.9), and 2,000 (82.0). The horizontal reference is supervised TSA control at 73.6. Markers are the available observations; line segments connect them as a visual guide. Spider 2.0-Lite is plotted separately in Figure 2.

B.4 Reward construction and termination

`C-full` computes the relevance vector at the episode start and holds it fixed for all stepwise coverage differences in that rollout. Both gradient paths into the vector are stopped. The relevance projection is trained during a 200-episode warmup and then frozen; the state and policy continue their specified PPO updates. The fixed snapshot makes the within-rollout coverage changes refer to the same relevance target. E8a removes freezing, while E8b substitutes frozen off-the-shelf mpnet relevance without the learned projection. Their EX scores are 81.4 and 79.8, versus 82.0 for `C-full`; the freezing protocol improves over E8a by 0.6 points.

Table 5 reports termination accuracy, call counts, and relative token costs. Removing the termination rule lowers overall EX by 3.6 points and increases mean calls by 3.7. On the 16+ bucket it yields 53.9 EX, a 6.4-point decrease. These are separate interventions from replacing the state or removing progress feedback, so the comparisons characterize their measured effects under each configuration.

B.5 Error-taxonomy construction

The construction corpus consists of 5,214 error messages from training-split databases. mpnet embeddings and HDBSCAN produce 34 clusters, which are manually consolidated into the 12-category

vocabulary, including UNKNOWN, listed in Appendix A. The resulting frequency vector has dimension 12. The mapper routes 71.4% of messages through deterministic regular-expression/keyword rules, 23.9% through a GBDT using string features, and 4.7% through an LLM fallback. On 500 manually checked messages, mapping accuracy is 96.2%.

Coverage is 98.7% on training errors, 98.3% on development errors, 98.0% on test errors, and 97.4% on two additional held-out databases. Development/test error-type distributions have KL divergence at most 0.08 from the training distribution. The training-only construction corpus establishes the provenance of the vocabulary; coverage and divergence characterize its behavior on the other splits. New types extend the frequency vector and its projection while preserving the other state components.

C Extended Related Work

LLM-based database and text-to-query agents The application of LLMs to database tasks has evolved rapidly from single-turn generation toward interactive, execution-aware agents. The most recent systems make execution feedback central: SDE-SQL (Xie et al., 2026) lets the model probe the live database with exploratory queries before committing to an answer, ReEx-SQL (Dai et al., 2026) trains an execution-aware reasoning policy with reinforcement learning, and ReFoRCE (Deng et al., 2025) is a strong published self-refinement method with consensus enforcement and column exploration. In parallel, agentic enterprise systems such as ProSPy (Yang et al., 2026b) and AgentSM (Biswal et al., 2026) add profiling and semantic memory to scale to large schemas, while governed-API layers move "beyond text-to-SQL" toward safe analytics endpoints (Singh et al., 2026). Slightly earlier work established the multi-agent and prompting foundations these systems build on: MAC-SQL (Wang et al., 2025a) introduced a three-agent decomposition for generation, verification, and rewriting; SQL-of-Thought (Chaturvedi et al., 2025) added guided error correction; DAIL-SQL (Gao et al., 2024) and CHASE-SQL (Pourreza et al., 2025) advanced few-shot and multi-path prompting; and LinkAlign (Wang et al., 2025b) tackled schema linking at multi-database scale. Recent surveys document how quickly this space is maturing (Shi et al., 2025). These systems explore complementary forms of workflow design, execution feedback, memory, and policy learning. TSADB focuses on a structured input to tool selection, using schema graphs and error semantics as inductive biases for a jointly trained state encoder and lightweight policy.

Related work also broadens the data tasks and context these agents handle. DS-STAR studies iterative planning and verification for data science, TableMind studies tool-augmented table reasoning, and EntSQL benchmarks grounding in long-context enterprise knowledge (Nam et al., 2025; Jiang et al., 2025a; Liao et al., 2026).

Reinforcement learning for LLM agents Reinforcement learning has received growing attention for agent training following DeepSeek-R1 (DeepSeek-AI, 2025), and the last year has produced a wave of methods targeting credit assignment in long-horizon tasks. The most recent work makes credit assignment explicit and structured: HiPER (Peng et al., 2026b) introduces hierarchical RL with explicit per-step credit for multi-turn agents, graph-based attribution assigns credit beyond the trajectory level (Cheng et al., 2026a), and dedicated analyses map how reasoning-style credit assignment transfers to agentic settings (Zhang, 2026). Group-based estimators such as GiGPO (Feng et al., 2025) deliver fine-grained credit without a critic, ARPO (Dong et al., 2025) adapts rollouts to entropy fluctuations during tool use, StepAgent (Song et al., 2025) derives intermediate rewards via inverse RL, and POAD (Huang et al., 2025) decomposes optimization to the token level; Agent Lightning (Luo et al., 2025) connects RL training to existing frameworks, including text-to-SQL. Comprehensive surveys chart this emerging "agentic RL" landscape and its open problems (Zhang et al., 2025a; Jiang et al., 2025b). TSADB is complementary: it supplies the *domain-specific state abstraction and dense progress reward* that let standard PPO learn effective database policies, and the state and reward removals have larger effects than the PPO-to-GRPO substitution in our ablations.

Tool-augmented and adaptive agents A broad literature studies tool use, planning, and feedback learning as generic agent capabilities. Reviews organize this space along reasoning enhancement, tool augmentation, multi-agent collaboration, and memory (Li, 2025; Xu et al., 2025a; Wei et al., 2025), and evaluation surveys stress that adaptive, tool-using agents must be judged on multi-step task completion rather than single outputs (Yehudai et al., 2026). Reflexion (Shinn et al., 2023) popularized verbal self-reflection from linguistic feedback, and R&D-Agent (Yang et al., 2025)

demonstrated automated solution building through LLM-powered loops. Recent work also highlights that tool *selection* itself is a safety- and reliability-relevant decision, not a neutral routing step (Yang et al., 2026a). Our contribution is to show that, for the database domain specifically, structured state abstraction over schema topology and a learned error-taxonomy encoder enable substantially more effective and robust tool use than prompt-based reflection alone.

Generalization across query languages and interaction modes Because we evaluate generalization beyond SQL, we connect to work on other query languages and interactive settings. Text-to-SPARQL over knowledge graphs is an active area, with benchmarks and methods spanning prompt engineering (Kosten et al., 2025b), resource bootstrapping across query languages (Kosten et al., 2025a), and schema-realization approaches that stress robustness to heterogeneous graphs (Dai et al., 2025). Multi-model text-to-query benchmarks such as SM3 (Sivasubramaniam et al., 2024) and augmentation techniques that treat query skeletons as structured objects (Ji et al., 2025) further motivate query-language-agnostic state representations. On the interaction axis, BIRD-Interact (Huo et al., 2026) reframes evaluation around dynamic multi-turn interactions, and enterprise ETL-debugging studies (Ye et al., 2026) show that recovery over multiple turns is where current systems fail. TSADB's task-state abstraction is designed to transfer across these settings because it encodes schema *structure* and *error semantics* rather than surface SQL syntax, a hypothesis we test directly in Appendix E.

Recent interaction and training studies. Dynamic multi-turn SQL reasoning (Sun et al., 2025), long-horizon credit assignment (Cheng et al., 2026a; Zhang, 2026), and SQL memory architectures (Tummalapenta & Addanki, 2026) address different parts of the agent loop. State learning in our setting connects these concerns through an explicit controller input. The comparisons below summarize the information pathway rather than rank systems evaluated under different protocols.

Table 7: Representative design distinctions.

Method	Main learning/design target	Information consumer
DualSQL	Joint schema linking and SQL generation	Shared LLM across agents
BAP-SQL	Budgeted observation formation	Agent reasoning and runtime guards
SERL-SQL	Selective hindsight training feedback	LLM policy through RL advantages
Reward-SQL	Execution-aware process rewards	Training and inference reasoning
CompactionRL	Task execution and context compaction	Long-horizon coding agent
TSADB	Structured state and tool policy	Separate lightweight controller

D Discussion and Future Directions

The experiments connect effective database interaction to three complementary sources of structure: schema topology, accumulated error information, and a bounded trace of recent execution. Improvements persist across the evaluated backbones (Appendix G) and across PPO and GRPO, while removing state abstraction or progress feedback produces larger performance drops (Table 12). The cost analysis places these gains in context: the measured state-module overhead is small, and graph processing has an analytical $O(|V| + |E|)$ dependence at fixed feature width (Appendix H).

Representations across data models. The SPARQL study extends schema encoding to RDF class/property graphs, and the supplied-turn BIRD-Interact-Lite protocol exercises intent and trace updates across follow-up requests. Document stores and graph-traversal languages such as Cypher offer further settings in which to examine the representation. Their different structures and operations may require changes to the schema interface or tool set.

Adapting feedback and memory. The error taxonomy originates in relational backends. Projection-only adaptation is effective for the extensions studied here; broader failure modes, such as distributed-query planner errors, motivate richer encoders and adaptation strategies. The fixed trace window

bounds its processing cost, while longer tasks may benefit from selectively retaining older events. These extensions would allow the state representation to adapt to the information demands of the task.

Operational deployment. The experiments use simulated or capped-latency execution environments. Production deployments add concurrency, permission boundaries, and monetary or resource budgets to the decision problem. Backbone errors and biases also remain relevant when a tool policy chooses an otherwise appropriate operation. Evaluating tool selection under least-privilege access, execution budgets, and human oversight is a useful direction for reliable deployment (Yang et al., 2026a).

E Generalization Beyond SQL and Single-Turn Interaction

This appendix details the generalization experiments on the SPARQL and multi-turn benchmarks.

Beyond SQL Table 8 evaluates TSADB on the Spider4SPARQL benchmark (Kosten et al., 2025b), a SPARQL-based knowledge-graph question-answering setting in which natural-language questions must be translated into executable SPARQL over RDF graphs. We keep the architecture unchanged except for routing `sql_exec` to the SPARQL query engine and extending the error taxonomy with three graph-specific error types (missing property, invalid IRI, and ontology mismatch). On Spider4SPARQL, TSADB improves execution accuracy by 8.6 points over the strongest matched transfer baseline in the controlled comparison, indicating that encoding schema *structure* (here, the RDF class/property graph) and error *semantics* transfers across query languages—the GCN treats an RDF ontology graph analogously to a relational foreign-key graph. TSADB obtains 56.5% EX using a frozen policy with only 50 episodes of W_e fine-tuning. SchemaForge (Dai et al., 2025) reports 64.18% EA using a purpose-built heterogeneous-KG pipeline. This external reference follows SchemaForge's published task split, candidate-graph setup, and execution-normalization pipeline; the other rows share our transfer protocol.

Beyond single-turn Under the custom clarification-free BIRD-Interact-Lite protocol, TSADB improves multi-turn TSR by 9.1 points over the strongest matched baseline. This setting tests how the execution-trace tracker and query-intent encoder preserve state across supplied follow-up turns.

BIRD-Interact-Lite custom transfer protocol We use the publicly released BIRD-Interact-Lite (Huo et al., 2026) development split, containing 300 multi-turn tasks. Follow-up requests are supplied from each task's gold interaction trajectory, and the agent receives each follow-up as input but cannot request clarification; it must generate SQL using the standard five-tool action space. This differs from official BIRD-Interact c-Interact, which includes a clarification budget and dialogue, and a-Interact, which defines nine agent actions including clarification. All systems receive the same five-tool suite; `schema_inspect` and `result_validate` let them ground queries before committing. We report TSR: a task succeeds only if its final SQL for every turn produces the correct execution result. The resulting 60.7% TSR measures completion across the supplied turns under this shared interface.

Error-taxonomy extensibility The 12-type error taxonomy was derived by clustering 5,000+ error messages collected from MySQL, PostgreSQL, and SQLite backends across the BIRD-SQL and Spider training databases; the resulting categories cover 98.3% of observed errors, with residual cases falling into the explicit *unknown* bucket. The taxonomy is extensible without retraining the full model: adding three vendor-specific error types (a proprietary lock-timeout, a partition-pruning warning, and a UDF-permission error) and fine-tuning only the error-encoder projection W_e for 50 episodes yields 81.7% EX on a held-out set exercising those errors (vs. 82.0% for the full model, 72.3% without the new types, and 81.9% with full retraining), while the GCN, execution tracker, and policy remain frozen.

The transfer to SPARQL motivates further evaluation on document stores and graph-traversal languages such as Cypher.

Table 8: Generalization beyond SQL and single-turn interaction under a frozen GPT-5 backbone. Spider4SPARQL uses EX; BIRD-Interact-Lite uses TSR under our custom clarification-free protocol, with supplied follow-up turns and a five-tool interface. The best matched transfer result is in **bold**; the final block gives the external SchemaForge result under its published protocol.

Method	**Spider4SPARQL**	**BIRD-Interact-Lite**
ReAct	38.2	41.5
MAC-SQL	44.7	48.9
DS-Agent	42.1	47.2
ReFoRCE	47.9	51.6
TSADB	**56.5**	**60.7**
External reference (published protocol)		
SchemaForge (Dai et al., 2025)	64.18	—

E.1 Official conversational evaluation

The official c-Interact experiment uses the BIRD-Interact-Lite user simulator and task-success evaluation on 300 tasks. TSADB obtains 62.4% task success, RL-ReAct 54.8%, and the MAC-SQL-style control 51.9%. These values use the official conversational interaction setting, separate from the supplied-gold-follow-up, clarification-free protocol above. Dialogue is evaluated through the benchmark simulator while database operations use the corresponding agent interface.

Table 9: Official c-Interact evaluation on 300 BIRD-Interact-Lite tasks.

Method	Task success (%)
RL-ReAct	54.8
MAC-SQL-style	51.9
TSADB	62.4

E.2 Adaptation budget on Spider4SPARQL

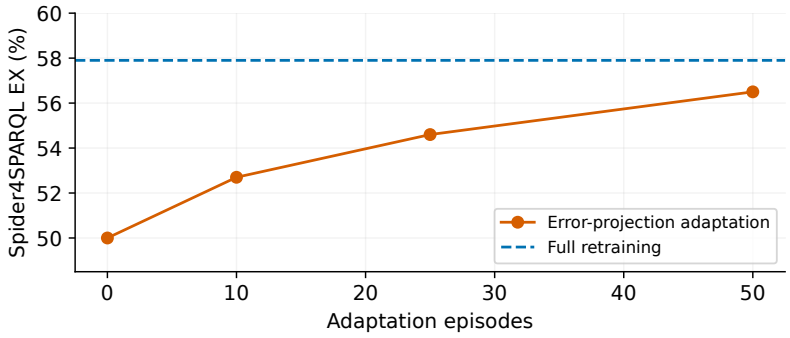

Figure 5: Spider4SPARQL error-projection adaptation with the policy frozen. The observed EX values are 50.0, 52.7, 54.6, and 56.5 at 0, 10, 25, and 50 adaptation episodes; full retraining reaches 57.9. Segments connect the reported observations.

The 0-to-50-episode improvement is 6.5 points, leaving 1.4 points to the full-retraining reference. At 25 episodes, the error-projection update has recovered 4.6 of those 6.5 points. This curve characterizes a limited adaptation budget through the error interface, with the policy and other state components held fixed during the local update.

Table 10: Transfer with frozen GPT-5. Spider4SPARQL uses a frozen policy and 50 episodes of error-projection adaptation. BIRD-Interact-Lite uses held-out tasks with supplied follow-up turns, five tools, and no clarification actions.

Method	Spider4SPARQL (EX)	BIRD-Interact-Lite (TSR)
ReAct	38.2	41.5
MAC-SQL	44.7	48.9
DS-Agent	42.1	47.2
ReFoRCE	47.9	51.6
TSADB	**56.5**	**60.7**

Table 11: Backbone-LLM sensitivity and fine-tuning study on BIRD-SQL (EX, %). "Strongest baseline" is the best of SDE-SQL-style/ReFoRCE per backbone.

Backbone	Strongest base.	TSADB	Δ
GPT-5	72.0	82.0	+10.0
GPT-5.5	76.0	83.7	+7.7
DeepSeek-V3	71.3	79.4	+8.1
Qwen2.5-72B	69.8	77.9	+8.1
Llama-3.1-70B	67.2	75.6	+8.4
Llama-3.1-8B (frozen)	58.4	66.9	+8.5
Llama-3.1-8B (LoRA-tuned)	61.2	69.8	+8.6

F Additional Transfer and Backbone Results

On Spider4SPARQL, TSADB reaches 56.5% EX, 8.6 points above ReFoRCE, after 50 episodes of W_e-only adaptation with the policy frozen. On held-out BIRD-Interact-Lite, it reaches 60.7% TSR, a 9.1-point improvement. This evaluation supplies follow-up requests from gold interaction trajectories and uses five tools without clarification actions; TSR requires correct final execution at every turn. The protocol measures state transfer across supplied turns, whereas official c-Interact and a-Interact evaluations include their respective dialogue and action interfaces. Table 10 reports the matched comparisons; Appendix E details the adapted interfaces and error-taxonomy extension.

Extending error feedback. The error encoder also accommodates new backend failures through local adaptation. Adding three vendor-specific types—a lock timeout, partition-pruning warning, and UDF-permission error—and tuning only W_e for 50 episodes yields 81.7% EX on the corresponding held-out evaluation. The comparison scores are 72.3% without the new types and 81.9% with full retraining; the full-model reference is 82.0%. This result illustrates how new feedback categories can be incorporated while retaining the learned schema, trace, and policy components.

Backbone sensitivity. The framework also improves BIRD-SQL EX by 7.7–10.0 points across GPT-5, GPT-5.5, DeepSeek-V3, Qwen2.5-72B, and Llama-3.1-70B. With Llama-3.1-8B, LoRA tuning improves both systems while preserving an 8.5–8.6-point advantage (Appendix G).

Inference cost. The TSA module takes approximately 16 ms of an 846 ms inference step, under 2% overhead. Mean BIRD-SQL task time is 13.6 s, compared with 11.2 s for ReAct and 24.8 s for MAC-SQL. The per-step total is dominated by LLM argument generation (approximately 590 ms) and database execution (240 ms). At fixed feature width, schema-graph processing scales as $O(|V| + |E|)$ and column-state storage as $O(|V|)$; the trace encoder uses a fixed ten-step window. These state-processing costs depend on schema complexity, while database execution can additionally depend on data volume. The one-time 72 GPU-hour training cost is amortized across subsequent tasks. Appendix H provides the full cost analysis.

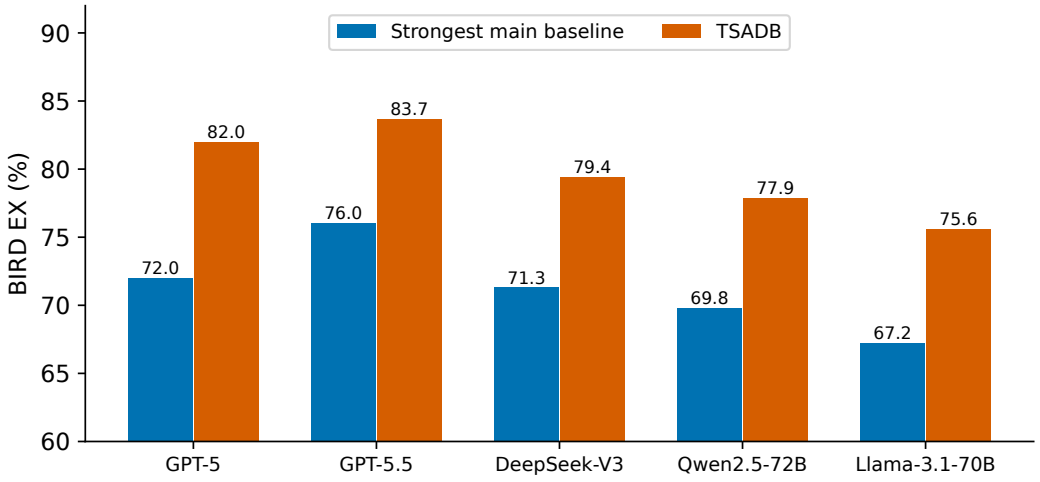

Figure 6: Backbone-LLM sensitivity on BIRD-SQL. TSADB delivers a consistent improvement over the strongest baseline across frontier (GPT-5, GPT-5.5) and open-weight (DeepSeek-V3, Qwen2.5-72B, Llama-3.1-70B) backbones.

G Backbone-LLM Sensitivity and Fine-Tuning Study

Backbone sensitivity Figure 6 and Table 11 report BIRD-SQL EX for TSADB and the strongest baseline across five backbones: GPT-5, GPT-5.5, and three open-weight models (DeepSeek-V3, Qwen2.5-72B, Llama-3.1-70B). TSADB improves over the strongest baseline by a consistent 7.7–10.0 points on every backbone, including the open-weight models, and the improvement persists across this range of argument-generating models. The consistent improvements support the framework across the evaluated backbones.

Fine-tuning study To probe the frozen-LLM assumption directly, we replace the frozen backbone with a small open model, Llama-3.1-8B, and fine-tune it with LoRA while keeping the rest of TSADB intact (last two rows of Table 11). Fine-tuning the small backbone raises absolute accuracy by 2.8–2.9 points for both the baseline and TSADB, and TSADB preserves its ≈8.5-point advantage throughout. These results show that backbone adaptation and structured tool selection can be combined, while a frozen backbone remains a useful operating point when full fine-tuning is impractical.

Policy head: MLP vs. small LLM A natural design question is whether the MLP policy could be replaced by a small LLM that reasons about tool selection in natural language. We compared the 2-layer MLP policy against a Llama-3.2-3B policy head with a classification layer. The accuracy gains are modest—LoRA-tuned $+0.8$ EX and fully fine-tuned $+1.3$ EX over the MLP—while training cost increases from ≈72 GPU-hours (∼0.2M parameters, ≈18 wall-clock hours) to ≈288 GPU-hours (LoRA, ∼8M parameters, ≈72 wall-clock hours) and ≈480 GPU-hours (∼3B parameters, full fine-tuning, ≈120 wall-clock hours). Because the state vector $\mathbf{s}_t$ already distills the decision-relevant context, a compact MLP suffices, and the MLP offers a favorable accuracy–efficiency trade-off.

H Computational Overhead, Latency, and Scalability

Inference latency Figure 7 (left) decomposes per-step latency. The TSA module adds only ≈16 ms per step—under 2% of the ≈846 ms total, which is dominated by the frozen-LLM argument-generation call (≈590 ms) and database execution (≈240 ms). Because TSADB selects tools with a compact MLP rather than an additional LLM call, its per-step latency is comparable to single-agent ReAct and substantially below multi-agent MAC-SQL, whose repeated inter-agent LLM calls inflate every step.

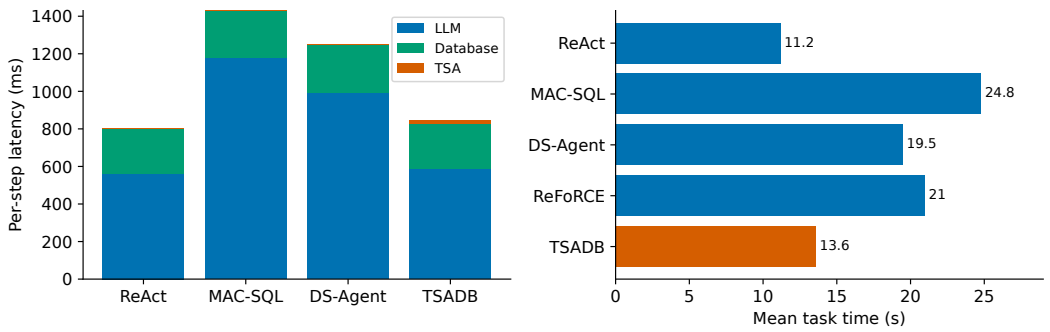

Figure 7: Left: per-step wall-clock latency decomposed into LLM calls, database execution, and the TSA module; the TSA overhead (≈16 ms) is under 2% of a step. Right: average end-to-end task completion time.

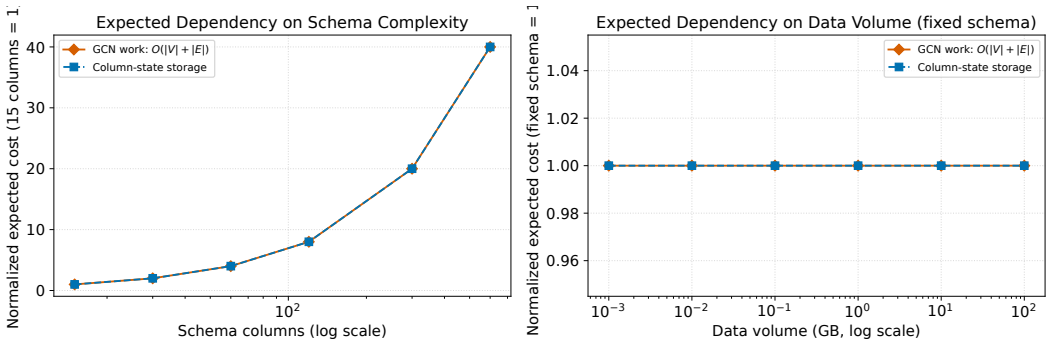

Figure 8: Analytical TSA complexity characterization. Left: expected schema-graph processing and column-state storage increase with schema complexity. Right: at fixed schema and trace window, their expected dependence on database volume is constant. The curves show normalized analytical dependencies.

End-to-end cost Figure 7 (right) reports average task completion time. TSADB completes BIRD-SQL tasks in 13.6 s on average—close to ReAct's 11.2 s and roughly *half* of MAC-SQL's 24.8 s—while achieving markedly higher accuracy. Two mechanisms explain this: more efficient tool selection reduces the number of tool calls per task, and early termination stops refinement once the validator's deterministic rule score reaches its fixed threshold. This combination yields lower total task time than the multi-agent baseline.

Analytical scalability characterization We characterize the scaling of the TSA module analytically. At fixed feature width, schema-graph processing scales with $O(|V| + |E|)$, where $|V|$ is the number of columns and $|E|$ the number of foreign keys, while column-state storage scales with $O(|V|)$. The execution-trace tracker remains bounded because its window is fixed at $K = 10$. Consequently, TSA computation has no direct dependence on database row count when the schema graph and trace window are fixed, although database execution itself may still depend strongly on data volume. Figure 8 plots these normalized expected dependencies; the log-scaled axes are used only for visualization. The plotted curves express these analytical dependencies.

Training cost Training requires a one-time ≈72 GPU-hours on four A100s (≈18 wall-clock hours) (2,000 episodes). This is amortized across all downstream inference while inference costs are reported separately above.

Learning efficiency. Section 5.3 compares learning curves under the same PPO procedure, composite reward, and 2,000-episode budget. The structured state supplies a compact policy input, and masking restricts exploration to applicable tools.

Table 12: Detailed ablation on BIRD-SQL (frozen GPT-5). Δ is the absolute change from the full model overall and on long-horizon tasks (10+ tool calls). The lower block reports architectural and embedding substitutions.

Variant	**EX (%)**	Δ	**LH** Δ
Full TSADB	82.0	—	—
Flat-history state	72.9	-9.1	-15.0
w/o Progress reward R_{prog}	71.5	-10.5	-16.8
w/o Tool masking	76.4	-5.6	-8.1
w/o Adaptive loop	75.3	-6.7	-12.0
w/o Schema encoder (GCN)	77.2	-4.8	-7.3
w/o Error encoder	78.0	-4.0	-8.6
w/o Exec. trace tracker	77.9	-4.1	-9.8
w/o Argument validator	80.6	-1.4	-2.1
Replace PPO with GRPO	79.6	-2.4	-3.0
Replace GCN with MLP	79.2	-2.8	-4.6
Replace Transformer with LSTM	80.4	-1.6	-3.4
GloVe $\rightarrow$ frozen BERT (col. emb.)	82.6	$+0.6$	$+0.5$
MLP policy $\rightarrow$ small-LLM policy (LoRA)	82.8	$+0.8$	$+1.0$

I Detailed Ablation Study

Table 12 provides the full ablation results. The two most impactful removals are the progress reward R_{prog} (-10.5 overall, -16.8 long-horizon) and the TSA module (-9.1 overall, -15.0 long-horizon). Removing R_{prog} removes coverage-based intermediate feedback while retaining correctness and efficiency signals; the 16.8-point long-horizon drop highlights its contribution to learning across many steps. The self-adaptive loop contributes 6.7 points overall but a disproportionate 12.0 on long-horizon tasks, and tool masking contributes 5.6 points, concentrated early in training. Among sub-encoders, the schema encoder is most impactful (-4.8), followed by the execution tracker (-4.1, amplified to -9.8 long-horizon) and the error encoder (-4.0, amplified to -8.6 long-horizon), as expected: longer interactions accumulate more actions to track and more diverse errors to recover from. The explicit argument validator adds 1.4 points by catching malformed queries before execution.

The lower block examines substitutions. Replacing PPO with GRPO costs only 2.4 points, indicating that the structured state matters more than the specific RL estimator. Replacing the GCN with an MLP over flattened schema features costs 2.8 points, supporting the graph inductive bias. Two substitutions *help* marginally but increase computational cost: replacing static GloVe column embeddings with a frozen BERT encoder adds only 0.6 EX while raising per-step latency by 9.2%, and replacing the MLP policy with a LoRA-tuned small-LLM policy adds 0.8 EX at $4\times$ the training cost. These results justify our efficiency-motivated defaults.

J Reward-Component Analysis

Training dynamics. The progress signal rises early, while correctness improves more gradually over training. The efficiency score peaks earlier and then declines as correctness continues to improve, exposing an accuracy–cost trade-off under the deployed weights. The plotted efficiency score is $\tilde{R}_{\text{eff}} = (R_{\text{eff}} - R_{\text{eff}}^{\min})/(R_{\text{eff}}^{\max} - R_{\text{eff}}^{\min} + 10^{-8})$, with the extrema taken across the 41 displayed checkpoints. Larger normalized values indicate greater efficiency. The operational efficiency reward remains the execution-time and query-length penalty defined in Section 4.3.

Marginal and joint weight selection. Each marginal sweep varies one weight while the other two share the remaining unit budget in a fixed ratio. The displayed correctness and efficiency sweeps attain their maxima near 0.5. With no correctness weight, the displayed EX is approximately 52%; larger values improve completion up to the middle of the sweep before accuracy decreases again. A separate joint validation search selects the deployed weights $(\alpha, \beta, \gamma_r) = (0.6, 0.15, 0.25)$ by varying their allocation together. The marginal curves describe sensitivity along their respective fixed-ratio paths, while the joint search determines the configuration used in `C-full`.

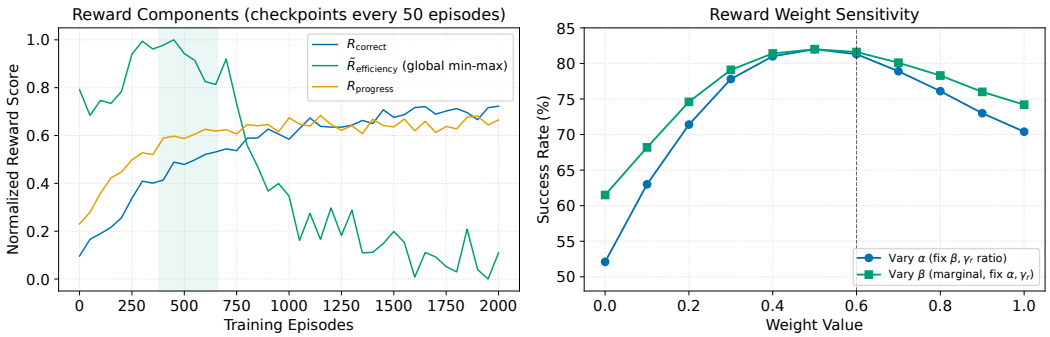

Figure 9: Reward dynamics and weight sensitivity. Left: reported reward components at 41 checkpoints spaced 50 episodes apart; efficiency is min–max normalized across the displayed checkpoints. Right: marginal reward-weight sweeps attain their displayed maxima near 0.5. A separate joint validation search selects $(\alpha, \beta, \gamma_r) = (0.6, 0.15, 0.25)$; the dashed line marks its correctness weight $\alpha = 0.6$.

The factorial in Table 3 isolates the progress term while retaining correctness and efficiency feedback: removing progress costs 10.5 points with TSA and 8.3 with flat history. The fixed-relevance controls in Appendix B further examine how the construction of that intermediate signal affects completion. Together, the reported dynamics, weight sweeps, and controlled removals describe how the reward components support learning.

K Additional Performance Visualizations

These plots complement the numerical results in Sections 5.2, 5.4, and 5.5.

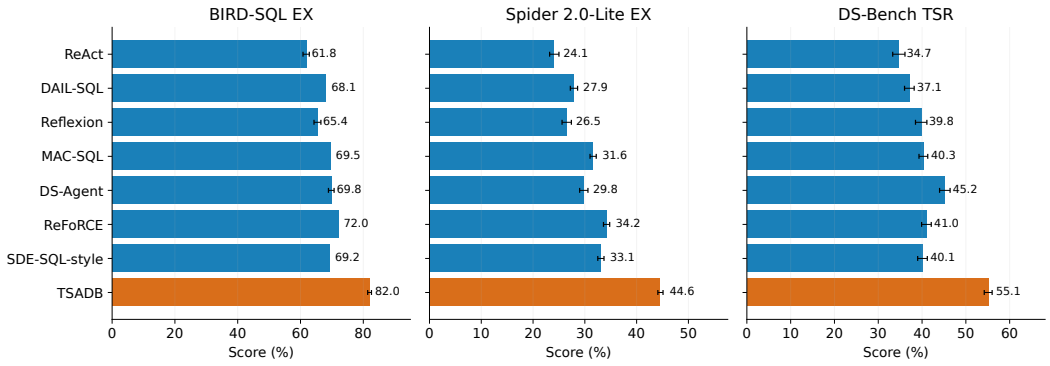

Figure 10: Core task performance under frozen GPT-5. Error bars show three-seed standard deviations where available; BIRD entries reported as means are plotted without error bars.

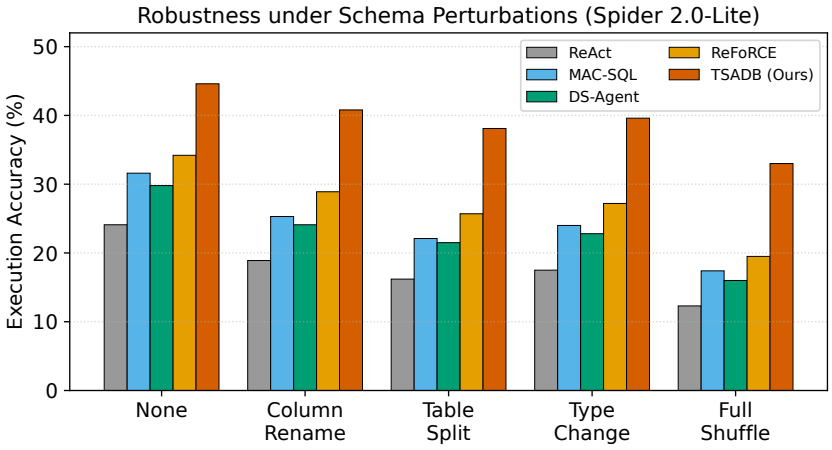

Figure 11: Spider 2.0-Lite execution accuracy under schema perturbations.

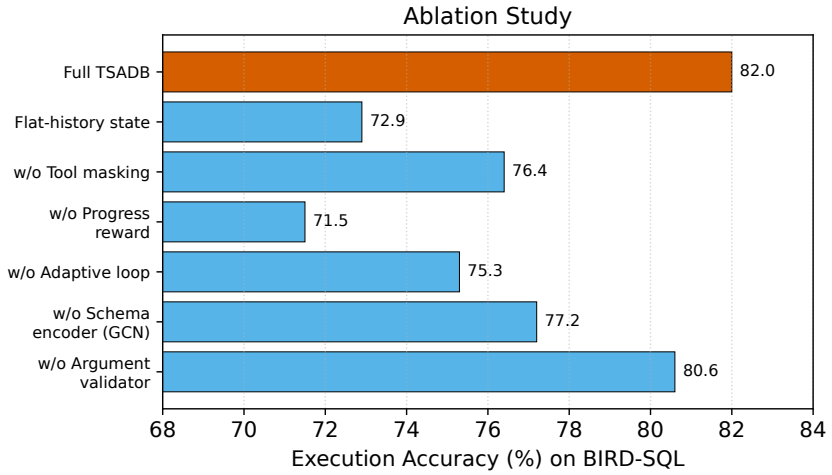

Figure 12: BIRD-SQL component ablations; the complete numerical comparison appears in Table 12.