# AGENTCYPHER: LEARNING TO REVISE AND ACCEPT TEXT-TO-CYPHER QUERIES FROM EXECUTION EVIDENCE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Text-to-Cypher translates natural-language questions into executable queries for retrieving information from property graphs. A key challenge is that queries can execute successfully and return nonempty results despite semantic errors, such as following relationships in the wrong direction. A ReAct formulation makes query execution part of the reasoning process, allowing the model to inspect results and revise its entity choices and graph patterns. We propose AgentCypher, a reinforcement learning framework that integrates query execution, verification, and revision to improve the accuracy of generated Cypher. Specifically, the Grounded Query Execution (GQE) module tests graph patterns constructed from selected schema elements and grounded entities against the target database. The Execution-Guided Verification (EGV) module examines these results and query-specific graph evidence for potential errors in entity grounding, relationship direction, and result multiplicity, guiding query revision and acceptance. The Graded Terminal Reward (GTR) module compares the accepted query's results with reference results, distinguishing exact answers from partial results through penalties for missing, extraneous, and duplicate rows. AgentCypher achieves 82.41% execution accuracy and 88.83% provenance subgraph Jaccard similarity on Cypher-Bench, and 42.13% execution accuracy on Neo4j-Text2Cypher.

## 1 INTRODUCTION

Text-to-Cypher translates natural-language questions into Cypher queries for retrieving information from property graphs (Ozsoy et al., 2025). Property graphs support applications such as social-network analysis, recommendation, and fraud detection (Francis et al., 2018). A natural-language interface lets users express requests such as whom a person follows or which account holders share contact information, while the generated query specifies how to retrieve the answer from the graph.

Existing methods pursue two complementary directions. The first improves query generation through schema and entity alignment. The words and entity mentions in a question may differ from schema terms and stored entity names. Retrieval and alignment methods, such as $R^3$-NL2GQL, connect the question to relevant schema components and graph data (Zhou et al., 2024). However, composing relationships, filters, and aggregations into a Cypher query still introduces errors in graph matching or result projection (Feng et al., 2025). To this end, existing methods fine-tune generation models on curated or synthetic pairs of questions and queries (Ozsoy et al., 2025; Zhang et al., 2026) or through reinforcement learning. For example, Cypher-RI uses execution-based rewards to jointly optimize schema selection and query generation (Su et al., 2025). However, without an execution-feedback loop, the model cannot revise a candidate using its observed results. Passing structural checks and returning nonempty results do not establish that a query answers the question (Tomczak, 2026; Prokopiou et al., 2026). For example, consider the question "Who does Alice follow?" in Figure 1. The incoming `FOLLOWS` pattern returns Carol, whereas the outgoing `FOLLOWS` pattern returns Bob. Both return a name, so successful execution and nonemptiness cannot distinguish them. Similarly, repeated output values may arise from multiple paths to one node or from distinct nodes with the same property value (Francis et al., 2018).

On the other hand, recent work on agent-based query generation offers a second direction based on ReAct-style interaction (Hua et al., 2026). ReAct interleaves reasoning, actions, and observations,



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

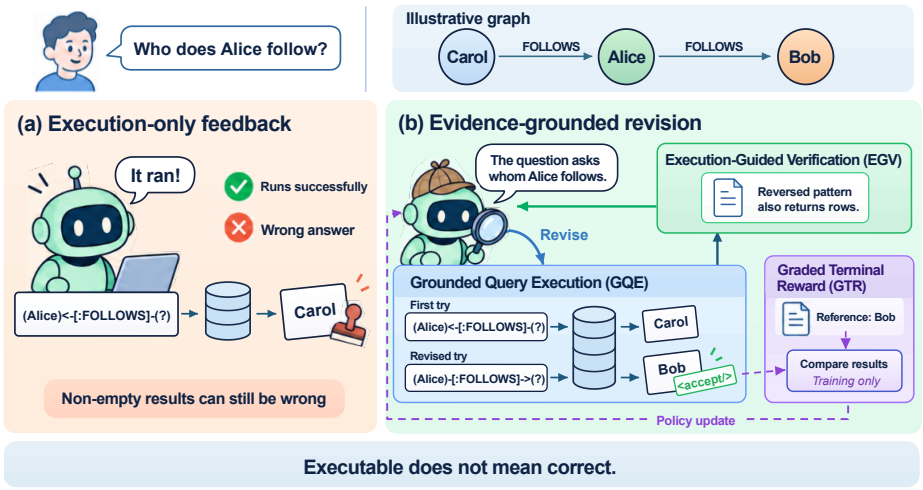



Figure 1: **An executable query can answer the wrong question.** The incoming `FOLLOWS` pattern returns Carol instead of Bob. Evidence that the reversed pattern also matches helps the policy reconsider direction using the question and accept the revised query.

outperforming action-only baselines across question answering, fact verification, and interactive decision-making (Yao et al., 2023). Adaptive Text2GQL and LAST-CQ introduce this interaction mechanism into Cypher generation and improve performance through validation and execution feedback (Niu et al., 2026; Prokopiou et al., 2026). However, the performance of these methods depends heavily on their pretrained backbones, including large models such as DeepSeek-V3 with 671B total parameters (DeepSeek-AI, 2024). Deploying and running such large models incurs substantial hardware and inference costs. To address this challenge, we investigate how ReAct-style reinforcement learning can train a small model for accurate Cypher generation.

Our main idea, illustrated in Figure 1(b), is to use query execution results to guide query revision and acceptance. We propose AgentCypher, a reinforcement learning framework that integrates query execution, verification, and revision to improve the accuracy of generated Cypher. First, the Grounded Query Execution (GQE) module tests graph patterns constructed from selected schema elements and grounded entities against the target database. It returns the candidate query's execution results to the model. Second, the Execution-Guided Verification (EGV) examines these results together with the execution results of additional check queries. It checks for potential errors in entity grounding, relationship direction, and result multiplicity to guide query revision and acceptance. The model interprets these results alongside the question, then revises and executes another query or explicitly accepts its latest attempt. Third, the Graded Terminal Reward (GTR) compares the accepted query's results with reference results during training. It distinguishes exact answers from partial results through penalties for missing, extraneous, and duplicate rows. The proposed AgentCypher achieves $82.41\%$ execution accuracy (EX) and $88.83\%$ provenance subgraph Jaccard similarity (PSJS) on CypherBench.

Our contributions are:

- We propose AgentCypher, a reinforcement learning framework that integrates query execution, verification, and revision to improve the accuracy of generated Cypher.
- We introduce three key modules: Grounded Query Execution (GQE) for testing graph patterns, Execution-Guided Verification (EGV) for guiding query revision and acceptance, and Graded Terminal Reward (GTR) for comparing accepted query results with reference results during training.
- Experiments on two public Text-to-Cypher benchmarks, CypherBench and Neo4j-Text2Cypher, demonstrate that AgentCypher achieves state-of-the-art performance among the compared 7B models on both benchmarks.

## 2 Related Work

**Text-to-Cypher generation.** Text-to-Cypher research addresses data scarcity, schema and entity alignment, and query correction. Text2Cypher, CypherSmith, and CYQUARK provide curated or

graph-grounded synthetic training pairs (Ozsoy et al., 2025; Zhang et al., 2026; Cazzaro et al., 2026). CypherBench supports evaluation of answer accuracy and retrieved graph structure (Feng et al., 2025). Grounding methods align questions with schema elements and stored entities (Zhou et al., 2024), while Cypher-RI couples schema selection with query generation through reinforcement learning and execution-based rewards (Su et al., 2025). Validation and repair frameworks combine structural checks with execution feedback to refine candidates (Niu et al., 2026). However, structural validity and nonempty results do not by themselves establish answer correctness (Tomczak, 2026; Prokopiou et al., 2026). An executable pattern can reverse a relationship or obscure distinctions between repeated matches and distinct entities (Francis et al., 2018). AgentCypher adds query-specific graph evidence about entity use, relationship direction, and result multiplicity, allowing its learned policy to reconsider a query even after successful execution.

**ReAct and feedback-driven agents.** ReAct interleaves reasoning, tool actions, and observations, allowing an agent to revise its decisions using information gathered from an external environment (Yao et al., 2023). Reflexion extends feedback use through verbal reflection and memory (Shinn et al., 2023), while RLEF trains code-generation policies to use execution feedback over successive attempts (Gehring et al., 2025). In database querying, Self-Debugging combines execution results with generated explanations for query repair (Chen et al., 2024), and ReEx-SQL and SQL-Trail optimize multi-turn interaction with SQL execution engines (Dai et al., 2026; Hua et al., 2026). Recent SQL agents also use database profiling and value-search tools (Kakkar et al., 2026). These studies establish precedents for iterative tool use, semantic revision, and learning from execution outcomes. For Cypher, graph-specific observations can expose how directed paths, node identities, and projected values contribute to the returned answer. AgentCypher combines these interaction mechanisms with graph-specific verification and GTR, training revision and acceptance against the accepted query's answer correctness.

## 3 METHOD

We train the model to decide from execution results whether to revise or accept a query, since a query that executes successfully can still ground the wrong entity, reverse a relationship, or return the wrong projection. Figure 2 summarizes the three components of AgentCypher. GQE executes candidate queries built from selected schema elements and grounded entities, and EGV runs additional check queries whose results, together with the execution results, guide revision and acceptance. During training, GTR scores the accepted query by penalizing missing, extraneous, and duplicate rows relative to the reference results.

### 3.1 TASK FORMULATION

Given a natural-language question $q$ and the schema $S$ of a target property graph $\mathcal{G}$, the task is to generate a Cypher query. Conditioned on $q$ and $S$, policy $\pi_\theta$ generates and revises candidate queries through interaction with the graph environment to obtain the final output $c_\tau$:

$$(q, S) \xrightarrow{\pi_\theta \text{ interacting with } \mathcal{G}} c_\tau. \tag{1}$$

To improve query generation quality, we train the policy to call entity lookup and query execution tools and use their outputs to revise candidate queries. The initial history and action space are

$$h_0 = (q, S), \qquad \mathcal{A} = \mathcal{A}_{\text{ent}} \cup \mathcal{A}_{\text{try}} \cup \{\texttt{<accept/>}\}. \tag{2}$$

An action in $\mathcal{A}_{\text{ent}}$ carries entity names for lookup. An action in $\mathcal{A}_{\text{try}}$ carries a candidate query for execution and verification. The `<accept/>` action terminates the episode without generating a new query. At ReAct turn $t$, the policy takes the preceding history $h_{t-1}$ as input and generates action $a_t$:

$$a_t \sim \pi_\theta(\cdot \mid h_{t-1}), \qquad h_t = h_{t-1} \parallel (a_t, o_t). \tag{3}$$

Here $\parallel$ denotes sequence concatenation, and $o_t$ is the tool observation returned for action $a_t$. For entity lookup, $o_t$ contains entity candidates. For a query trial, it contains query execution results and additional check results where applicable. After each tool call, the action $a_t$ and tool reply $o_t$ are appended to the history to form $h_t$. If interaction continues, the policy uses $h_t$ as input to the next ReAct turn and generates $a_{t+1}$ based on the previous actions and observations.

Let $\tau = (a_1, o_1, \ldots, a_n, o_n)$ denote the episode and $I_\tau = \{t : a_t \in \mathcal{A}_{\text{try}}\}$ the indices of its query trials. The output is

$$c_\tau = \begin{cases} \text{payload}(a_{\max I_\tau}), & \text{valid acceptance at turn } n,\ I_\tau \neq \emptyset, \\ \perp, & \text{otherwise}, \end{cases} \tag{4}$$

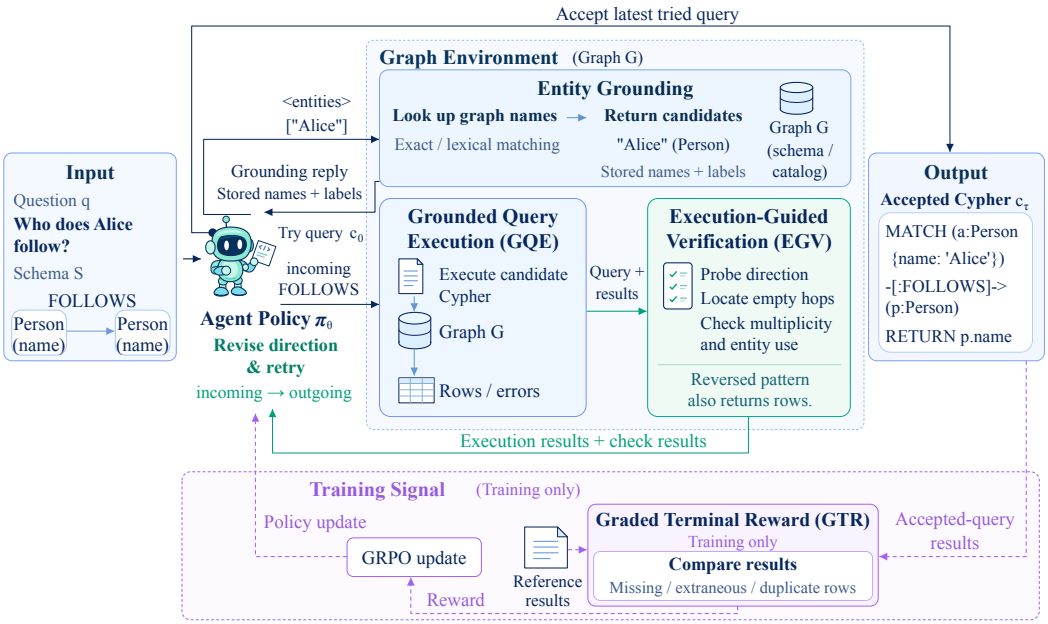



Figure 2: Overview of the AgentCypher framework. The policy grounds entities, executes candidate queries, and uses graph evidence from the EGV to guide revision and acceptance. During training, the purple dashed path compares the accepted query's results with reference results to compute GTR.

where $\mathrm{payload}(a_t)$ is the candidate query carried by a query action. Acceptance is valid only when `<accept/>` is the sole action in its turn. The output is therefore the latest tried query. $\perp$ denotes failure to accept a query, with no fallback to an earlier executable query. GQE executes $c_t$ (§3.2), EGV checks its result (§3.3), and the two outputs form the observation $o_t = (o_t^{\mathrm{GQE}}, \phi_t)$. The verification gate then decides whether the next action revises or accepts $c_t$, and during training GTR scores the accepted query $c_\tau$ (§3.4).

### 3.2 Grounded Query Execution

Cypher generation can fail when entity mentions do not match stored names or when constructed paths do not connect the intended entities. To address this limitation, we introduce GQE, which executes candidate queries and returns their results to the policy for query revision during generation. Entity lookup provides candidate names and labels, and query execution results guide the policy in revising its entity choices and graph patterns.

To ground entity references, the policy emits names from the question through `<entities>`. The request and successful lookup observation are represented by

$$\begin{aligned} \mathcal{M}_t &= \mathrm{payload}(a_t), \qquad a_t \in \mathcal{A}_{\mathrm{ent}}, \\ o_t &= \mathrm{Lookup}_{\mathcal{G}}(\mathcal{M}_t) = \{(m, \mathcal{C}_m, b_m) : m \in \mathcal{M}_t\}. \end{aligned} \tag{5}$$

Here $\mathcal{M}_t$ contains the emitted names, and $\mathcal{C}_m$ lists candidate stored names and labels retrieved by exact and lexical matching for name $m$. The indicator $b_m$ is 1 if the lookup finds a stored name that matches $m$ exactly as written and 0 otherwise. The policy selects entities using the question and these candidates. We denote by $\mathcal{N}_t$ the stored spellings confirmed by all lookups up to turn $t$, which EGV later uses to check entity use. The returned candidates are limited in number and response length. Using the schema and grounded entities, the policy constructs a candidate query $c_t$ and submits it through `<try>`. Let $P_t = [\![c_t]\!]_{\mathcal{G}}$ denote the result rows when execution succeeds. The execution observation is

$$o_t^{\mathrm{GQE}} = \begin{cases} (\mathrm{OK},\ |P|,\ \mathrm{preview}(P)), & \text{execution succeeds}, \\ (\mathrm{ERROR},\ e), & \text{execution fails}, \end{cases} \tag{6}$$

where $|P|$ is the total row count and $e$ is the error message. GQE thus has two outputs: the policy observes only $o_t^{\mathrm{GQE}}$, whereas the full result $P_t$ is passed to EGV. By combining entity lookup with query execution, GQE lets the policy resolve entity mentions and revise graph patterns.

### 3.3 EXECUTION-GUIDED VERIFICATION

A Cypher query can execute successfully and still be wrong. An empty result does not show which part of the pattern fails to match, and a nonempty result can come from a reversed relationship, from one node reached through several paths, or from a query that omits a grounded entity. EGV addresses this limitation with two parts. First, it executes check queries on each successfully executed candidate and returns their results as observations, making these errors visible for both empty and nonempty results. Second, a verification gate specifies how the policy uses these observations before acceptance: it must relate each reported finding to the question and either revise the query or explain why the finding does not affect the answer. The checks report facts about the query, while judging whether the query answers the question remains the policy's decision.

When $c_t$ executes successfully, EGV takes the outputs of GQE and returns

$$\phi_t = \mathcal{V}(c_t, P_t, \mathcal{G}, \mathcal{N}_t) \subseteq \Phi, \qquad \Phi = \{\varphi_{\text{loc}}, \varphi_{\text{dir}}, \varphi_{\text{rep}}, \varphi_{\text{ent}}\}, \qquad o_t = (o_t^{\text{GQE}}, \phi_t). \tag{7}$$

When execution fails, $\phi_t = \emptyset$ and $o_t = (o_t^{\text{GQE}})$. Here $\mathcal{V}$ denotes the check procedure, whose inputs are the candidate query $c_t$, its result rows $P_t$, the target graph $\mathcal{G}$, and the confirmed spellings $\mathcal{N}_t$. The set $\Phi$ contains four check results: $\varphi_{\text{loc}}$ localises an empty result and reports the associated constraint checks, $\varphi_{\text{dir}}$ reports whether the reversed path still matches, $\varphi_{\text{rep}}$ counts rows, distinct nodes, and distinct property values, and $\varphi_{\text{ent}}$ lists resolved entity names absent from the query. The returned subset $\phi_t$ depends on the result. An empty result receives $\varphi_{\text{loc}}$, whereas a nonempty result receives those of $\varphi_{\text{dir}}$, $\varphi_{\text{rep}}$, and $\varphi_{\text{ent}}$ that apply to the query. The observation $o_t$ is returned in one reply and appended to the history as $h_t$ in equation 3.

**Localisation on an empty result.** A path is a sequence of node patterns $n_i$ connected by relationship patterns $r_i$ within a `MATCH` clause. For a selected path with $K$ node patterns in the initial `MATCH` clause, the prefixes are

$$\psi_1 = (n_1), \qquad \psi_k = (n_1, r_1, n_2, \ldots, r_{k-1}, n_k) \quad (2 \le k \le K), \tag{8}$$

so $\psi_k$ contains the first $k$ node patterns and their connecting relationships, and $\psi_K$ is the complete path. We execute the prefixes in order and find the first one that returns no rows:

$$\ell(c_t) = \min\big\{\, k \in \{1, \ldots, K\} \;:\; [\![\psi_k]\!]_{\mathcal{G}} = \emptyset \,\big\}, \qquad \ell(c_t) = \infty \text{ if no prefix is empty.} \tag{9}$$

When $\ell(c_t) < \infty$, we report the first empty prefix $\psi_{\ell(c)}$. When $\ell(c_t) = \infty$, we report that the checked path has matches, and the check does not localise the cause of the empty result.

**Direction.** We reverse every arrow of the selected path $\psi_K$ and re-run the pattern:

$$v_{\text{dir}}(c_t) = \mathbf{1}\big[\, [\![\text{rev}(\psi_K)]\!]_{\mathcal{G}} \neq \emptyset \,\big]. \tag{10}$$

The value is reported in both cases. A value of 1 means that both the original and the reversed patterns match, so a nonempty result alone cannot determine the intended direction. A value of 0 means that the reversed pattern has no match.

**Repetition and entity use.** In a nonempty result, several rows may refer to the same node, and distinct nodes may share a property value. We distinguish these cases with three counts computed by one query over the result, and we compare the entity spellings $\mathcal{N}$ confirmed by entity lookup against the query text:

$$v_{\text{rep}}(c_t) = \big(\, |P_t|,\ |\pi_{\text{node}}(P_t)|,\ |\pi_{\text{val}}(P_t)| \,\big), \qquad v_{\text{ent}}(c_t) = \{\, n \in \mathcal{N} \;:\; n \not\sqsubseteq c_t \,\}. \tag{11}$$

Here $\pi_{\text{node}}$ and $\pi_{\text{val}}$ project the result onto the nodes and property values behind its rows. When $|P_t| > |\pi_{\text{node}}(P_t)|$, a node is reached through several paths. When $|\pi_{\text{node}}(P_t)| > |\pi_{\text{val}}(P_t)|$ and the returned property is a name, we report the names shared by distinct nodes.

**Verification gate.** The gate specifies how the policy uses $o_t$ before acceptance. The policy observes the history $h_t$, which ends with $o_t = (o_t^{\text{GQE}}, \phi_t)$, and chooses its next action as

$$a_{t+1} = \begin{cases} \texttt{<accept/>}, & g_\theta(h_t) = 1, \\ a \in \mathcal{A}_{\text{try}}, & \text{otherwise}, \end{cases} \tag{12}$$

where $g_\theta(h_t) \in \{0, 1\}$ is the policy's judgement that $c_t$ executed successfully and answers $q$. Before setting $g_\theta = 1$, the policy must relate each finding in $\phi_t$ to $q$ and either revise the query or explain why the finding does not affect the answer. Because a finding is a fact rather than a verdict, $g_\theta$ may be 1 when $P_t$ is empty or findings remain. Accepting makes $c_t$ the output $c_\tau$ in equation 4; otherwise, the revised query $c_{t+1}$ passes through GQE and EGV again. The gate is specified in the prompt, and EGV does not programmatically block acceptance.

### 3.4 GRADED TERMINAL REWARD

GTR aims to train revision and acceptance from the quality of the accepted answer. Cypher-RI assigns a fixed execution penalty to executable queries whose results disagree with the reference (Su et al., 2025). GTR distinguishes partial results through penalties for missing, extraneous, and duplicate rows, while keeping exact answers strictly preferred. These comparisons use reference results only during training. GTR takes the accepted query $c_\tau$ produced by the gate. Let $P_\tau = [\![c_\tau]\!]_\mathcal{G}$ and $\Gamma = [\![c^\star]\!]_\mathcal{G}$ denote the execution results of the accepted and reference queries, respectively. For the graded penalties, $m_X(u)$ counts row $u$, $\mathrm{supp}(X)$ contains distinct rows, and $|X|$ is the total row count. These row statistics use sorted, stringified cell values, separately from EX. The relation $P_\tau \equiv \Gamma$ denotes exact result matching under the benchmark protocol in §4.1. Three penalty shares describe how that result differs from the reference:

$$p_{\mathrm{dup}} = \frac{|P_\tau| - |\mathrm{supp}(P_\tau)|}{|P_\tau|}, \quad p_{\mathrm{miss}} = \frac{\sum_u \big(m_\Gamma(u) - m_{P_\tau}(u)\big)^+}{|\Gamma|}, \quad p_{\mathrm{extra}} = \frac{\sum_{u \notin \mathrm{supp}(\Gamma)} m_{P_\tau}(u)}{|P_\tau|}, \tag{13}$$

and the terminal reward is

$$R(c_\tau, \mathcal{G}) = \begin{cases} +2.0, & P_\tau \equiv \Gamma, \\ \max\big(-2.0,\ -0.5 - 1.5\,(p_{\mathrm{dup}} + p_{\mathrm{miss}} + p_{\mathrm{extra}})\big), & \mathrm{supp}(P_\tau) \cap \mathrm{supp}(\Gamma) \neq \emptyset, \\ -2.0, & \text{otherwise, including } c_\tau = \bot. \end{cases} \tag{14}$$

A surplus copy of a row the reference contains is charged to $p_{\mathrm{dup}}$ alone. A repeated foreign row is charged to both $p_{\mathrm{dup}}$ and $p_{\mathrm{extra}}$, and we do not correct for the overlap: the sum is clipped well before it matters, since the clamp in equation 14 binds once $p_{\mathrm{dup}} + p_{\mathrm{miss}} + p_{\mathrm{extra}} \geq 1$. The one axis that carries information is how much of the reference a partially correct answer recovered, and it is the only one graded.

**Policy optimization.** We train the policy with group relative policy optimization (GRPO) (Shao et al., 2024). For each question, we sample a group of $N$ trajectories $\tau_1, \ldots, \tau_N$ from $\pi_{\theta_{\mathrm{old}}}$ and assign each trajectory $\tau_i$ the terminal reward $R_i = R(c_{\tau_i}, \mathcal{G})$ in equation 14. Following trajectory-level reinforcement learning for tool-using language models (Jin et al., 2025; Sheng et al., 2024), a token-level mask sets $M_{i,t} = 1$ for tokens emitted by $\pi_\theta$, including entity names, candidate queries, and acceptance actions, and $M_{i,t} = 0$ for environment replies, which are excluded from the loss. The actor objective is

$$\mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} \frac{1}{\sum_t M_{i,t}} \sum_{t:\, M_{i,t}=1} \min\Big(\rho_{i,t}\hat{A}_i,\ \mathrm{clip}\big(\rho_{i,t}, 1-\epsilon, 1+\epsilon\big)\hat{A}_i\Big)\right], \tag{15}$$

where $\rho_{i,t}$ is the probability ratio between $\pi_\theta$ and $\pi_{\theta_{\mathrm{old}}}$ for token $\tau_{i,t}$, $\epsilon$ is the clip ratio, and $\hat{A}_i$ is the group-normalized advantage,

$$\hat{A}_i = \frac{R_i - \mathrm{mean}(R_1, \ldots, R_N)}{\mathrm{std}(R_1, \ldots, R_N)}. \tag{16}$$

The advantage $\hat{A}_i$ is shared by all unmasked tokens of $\tau_i$, so a group in which all trajectories receive the same reward contributes no gradient.

## 4 EXPERIMENT

### 4.1 DATASETS AND EVALUATION METRICS

We evaluate AgentCypher on CypherBench (Feng et al., 2025) and Neo4j-Text2Cypher (Ozsoy et al., 2025). For CypherBench, we train on four graphs (Art, Biology, Soccer, Terrorist Attack) and evaluate on the complete test split of 2,348 questions over seven unseen graphs (Company, Fictional Character, Flight Accident, Geography, Movie, NBA, and Politics). Neo4j-Text2Cypher pairs questions with graph schemas and reference Cypher queries from multiple graph databases. The graph-pattern analyses and ablations below use CypherBench. On CypherBench, we report Execution Accuracy (EX), the percentage of predicted queries whose execution results match those of the reference queries; Provenance Subgraph Jaccard Similarity (PSJS), the Jaccard similarity between the subgraphs matched by the two queries' `MATCH` clauses; and the percentage of queries that execute successfully (Exec.). On Neo4j-Text2Cypher, we report EX and Google-BLEU (G-BLEU), the token $n$-gram overlap between predicted and reference queries.

Table 1: Performance comparison on CypherBench and Neo4j-Text2Cypher. The best results are shown in bold.

| | CypherBench | | | Neo4j-Text2Cypher | |
|---|---|---|---|---|---|
| Method | EX (%) | PSJS (%) | Exec. (%) | EX (%) | G-BLEU (%) |
| Qwen2.5-Coder-7B (Hui et al., 2024) | 13.12 | 22.65 | 62.95 | 12.06 | 44.37 |
| Gemma-2-9B (Gemma Team, 2024) | 18.61 | 30.67 | 68.57 | 14.49 | 51.04 |
| Llama-3.1-8B (Dubey et al., 2024) | 18.82 | 30.98 | 90.67 | 11.86 | 41.30 |
| GPT-4o-mini | 31.43 | 45.91 | 87.39 | 25.82 | 59.62 |
| Gemini-1.5-Pro (Gemini Team, 2024) | 39.95 | 57.70 | 86.03 | 23.07 | 58.45 |
| GPT-4o (OpenAI, 2024) | 60.18 | 76.87 | 94.90 | 31.73 | 62.93 |
| $R^3$-NL2GQL (Zhou et al., 2024) | 23.94 | 38.12 | 64.83 | 19.27 | 47.91 |
| Text2Cypher-Gemma-3-27B (Ozsoy et al., 2025) | 38.71 | 55.53 | 92.97 | 29.84 | 50.10 |
| CypherSmith (Zhang et al., 2026) | 74.80 | 77.50 | 99.20 | — | — |
| Cypher-RI (Su et al., 2025) | 69.59 | 75.21 | 99.28 | 30.61 | 60.13 |
| AgentCypher (ours) | **82.41** | **88.83** | **99.87** | **42.13** | **73.24** |

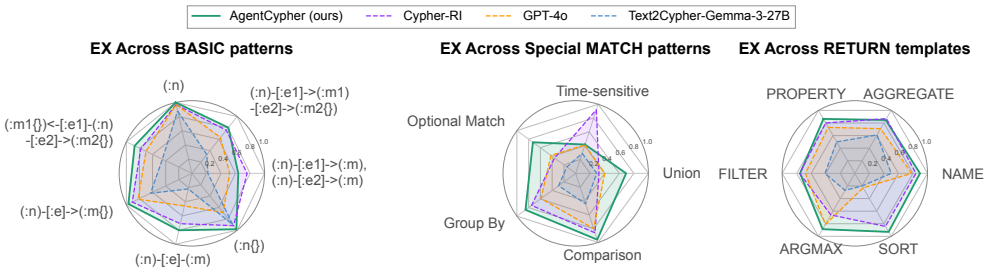



Figure 3: Execution accuracy by query pattern on CypherBench.

### 4.2 Comparison with State-of-the-Art Methods

Table 1 compares AgentCypher with general-purpose models and Text-to-Cypher-specific methods. The general-purpose models are Qwen2.5-Coder-7B, Gemma-2-9B, Llama-3.1-8B, GPT-4o-mini, Gemini-1.5-Pro, and GPT-4o. The Text-to-Cypher-specific models are $R^3$-NL2GQL, Text2Cypher-Gemma-3-27B, CypherSmith, and Cypher-RI. Cypher-RI (Su et al., 2025) learns schema selection within GRPO-based query generation. It uses query execution when computing training rewards. However, it does not use execution feedback to revise candidate queries before submission. AgentCypher additionally returns candidate-query execution results through GQE and check results through EGV to the policy, allowing revision before acceptance. GTR distinguishes partially correct answers through penalties for missing, extraneous, and duplicate rows. On CypherBench, AgentCypher exceeds Cypher-RI by 12.82 and 13.62 percentage points in EX and PSJS, respectively, while their executable percentages differ by only 0.59 points. The larger gains in answer accuracy and subgraph overlap show an advantage beyond producing executable queries. On Neo4j-Text2Cypher, AgentCypher also exceeds Cypher-RI's reported EX and G-BLEU by 11.52 and 13.11 percentage points, respectively. AgentCypher achieves 42.13% EX and 73.24% G-BLEU on Neo4j-Text2Cypher (Table 1). These scores exceed the reported Cypher-RI results by 11.52 and 13.11 percentage points, respectively. The two metrics evaluate answer agreement and query-text overlap, providing complementary views of performance on this dataset. As with the CypherBench comparison, imported baseline scores do not constitute a matched-budget ablation of individual components.

### 4.3 Performance across Graph Patterns

Figure 3 compares AgentCypher with Cypher-RI, GPT-4o, and Text2Cypher-Gemma-3-27B across BASIC patterns, special `MATCH` categories, and `RETURN` templates on CypherBench. As shown in the left panel, AgentCypher performs well on patterns with named-node constraints. It achieves 98.31% on constrained single-node queries, 96.91% on single-relation patterns with a named endpoint, and 87.39% on two-branch patterns connecting two named neighbours. On two-hop chains ending at a named node, it reaches 80.55%, compared with 63% for GPT-4o and 33% for Text2Cypher-Gemma-3-27B. These results are achieved because GQE uses entity lookup and execution feedback. Entity lookup provides candidate names and labels, and execution results allow the policy to test paths connecting the selected entities. EGV additionally checks resolved entity use

and probes the first `MATCH` path to guide revisions of entity constraints and relation traversal before acceptance. On single relations between unconstrained nodes, AgentCypher reaches $80.28\%$, above Cypher-RI's $71\%$. On patterns with two relations between the same nodes, AgentCypher achieves $63.83\%$, below Cypher-RI's $76\%$. Our current feedback leaves a gap in checking the semantics of these queries: GQE returns query execution results, while EGV checks graph matching, relationship direction, and result multiplicity within the generated query. These checks do not explicitly verify that both relations bind the same node pair in the roles required by the question, or that the query returns the requested entity or property. The middle panel shows higher execution accuracy for AgentCypher than Cypher-RI on comparison, group-by, optional match, and union queries. The clearest separations are on optional match and union. On optional match, AgentCypher reaches $72.76\%$, compared with $36\%$ for Cypher-RI and $42\%$ for GPT-4o. Its $95.92\%$ on comparison and $85.38\%$ on group-by also indicate strong performance on comparisons of node-property values and grouping. On union, AgentCypher reaches $69.35\%$, compared with Cypher-RI's $32\%$. Time-sensitive queries are an exception to these gains: AgentCypher achieves $41.67\%$, below Cypher-RI's $92\%$. Time-sensitive queries require checking relationship validity intervals, including their boundaries and open-ended end dates. Among failed predictions, we observe omitted temporal filters, exact start-year conditions replacing interval membership, and lifespan conditions replacing relationship dates, indicating inconsistent translation of temporal requirements. The right panel compares six `RETURN` templates. AgentCypher reaches $86.48\%$ on property projection and $88.68\%$ on argmax, compared with $80\%$ and $66\%$ for Cypher-RI, respectively. These results indicate stronger performance in retrieving properties and selecting entities by extreme values. On sort, AgentCypher achieves $92.98\%$, exceeding Cypher-RI's $84.21\%$ and GPT-4o's $23\%$. The gains therefore extend to ordering as well as property projection and argmax.

### 4.4 Ablation Studies

**Effectiveness of Individual Modules.** Our baseline is Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) trained with GRPO. Table 2 evaluates the contributions of these modules through different combinations. Adding GQE raises EX from $72.34\%$ to $78.40\%$ and PSJS from $76.52\%$ to $85.34\%$. These gains demonstrate that GQE can effectively guide query revision using execution results, helping the policy refine its entity choices and graph patterns before acceptance. Adding EGV to GQE further increases EX and PSJS by $2.41$ and $1.19$ percentage points, reaching $80.81\%$ and $86.53\%$, respectively. This improvement supports the value of additional check results for guiding query revision and acceptance when execution status and result previews leave entity grounding, relationship direction, or result multiplicity ambiguous. Adding GTR to GQE instead raises EX to $79.55\%$ and PSJS to $88.25\%$. These gains demonstrate the effectiveness of GTR's graded training signal for improving query revision and acceptance. By penalizing missing, extraneous, and duplicate rows, GTR distinguishes partially correct results and gives a graded training signal for query revision and acceptance. This supports the complementary roles of execution results for query revision, additional check results for revision and acceptance, and graded terminal rewards for training these decisions.

Table 2: Ablation study of GQE, EGV, and GTR on CypherBench.

| Setting | EX (%) | PSJS (%) | Exec. (%) |
|---|---|---|---|
| Baseline | 72.34 | 76.52 | 99.33 |
| Baseline + GQE | 78.40 | 85.34 | 99.27 |
| Baseline + GQE + EGV | 80.81 | 86.53 | 99.64 |
| Baseline + GQE + GTR | 79.55 | 88.25 | 99.56 |
| Baseline + GQE + EGV + GTR | 82.41 | 88.83 | 99.87 |

**Effectiveness of GQE.** We compare AgentCypher ReAct with four interaction designs on CypherBench (Table 3). *Vanilla ReAct* interleaves reasoning, task-specific actions, and observations for question answering and interactive decision making (Yao et al., 2023). *Execution-feedback RL* trains policies to revise queries using execution errors and returned rows (Dai et al., 2026; Hua et al., 2026). *Tool-augmented RL* combines Cypher execution, column profiling, and value search to align question phrases with stored values (Kakkar et al., 2026). *Graph-query repair* combines validation against the live schema, query correction, and empty-result relaxation, including case-insensitive and substring matching (Prokopiou et al., 2026). It stops refinement once a validated query returns nonempty results. For Cypher generation, relational value-search tools require adaptation to node names and labels, while the relaxation in Prokopiou et al. (2026) does not specify a separate candidate entity lookup stage. Unresolved entity ambiguity can produce empty results or

queries over unintended nodes, which may contribute to the lower scores in Table 3. GQE retrieves candidate stored names and labels so the policy can revise its entity choices and graph patterns using query execution results. This comparison evaluates the complete AgentCypher ReAct setting, while Table 2 examines the contribution of GQE. AgentCypher ReAct achieves the highest EX and PSJS at $82.41\%$ and $88.83\%$, respectively. It exceeds Vanilla ReAct by approximately $7.97$ and $9.20$ percentage points in EX and PSJS. The strongest alternatives are graph-query repair in EX and tool-augmented RL in PSJS. AgentCypher ReAct surpasses them by approximately $3.10$ and $2.41$ percentage points, respectively. These gains support the effectiveness of AgentCypher ReAct in using entity candidates and query execution results to guide query revision and acceptance.

Table 3: Comparison of ReAct settings on CypherBench.

| Setting | EX (%) | PSJS (%) |
|---|---|---|
| Vanilla ReAct (Yao et al., 2023) | 74.44 | 79.63 |
| Execution-feedback RL (Dai et al., 2026; Hua et al., 2026) | 75.72 | 80.29 |
| Tool-augmented RL (Kakkar et al., 2026) | 79.15 | 86.42 |
| Graph-query repair (Prokopiou et al., 2026) | 79.31 | 85.71 |
| AgentCypher | 82.41 | 88.83 |

**Analysis of GTR.** GTR targets a failure mode specific to Cypher generation: graph pattern matching often yields partially correct answers, where an under-constrained pattern returns extraneous rows, an over-constrained pattern misses reference rows, and multiple matching paths return duplicate rows that EX counts. We compare GTR with five alternative terminal rewards on CypherBench (Table 4), keeping the GQE–EGV interaction, the acceptance protocol. None of the alternatives separates these errors. Execution reward (Yao et al., 2026) and execution-tiered reward (Su et al., 2025) assign the same reward to every incorrect answer. Text-level (Pourreza et al., 2025b) and structure-level (Stoisser et al., 2025) partial rewards grade resemblance to the reference query rather than the returned answer. Result-level partial reward (Papicchio et al., 2026) grades row overlap without distinguishing missing, extraneous, and duplicate rows. GTR instead penalizes missing, extraneous, and duplicate rows separately while keeping exact answers strictly preferred, so a partially correct answer receives a reward that reflects how it deviates from the reference. As a result, GTR achieves the highest EX and PSJS at $82.41\%$ and $88.83\%$, improving over execution reward by $1.60$ and $2.30$ percentage points. Text- and structure-level rewards raise PSJS but lower EX, since a query can resemble the reference while returning a different answer. Result-level partial reward is the strongest alternative in EX at $81.54\%$, and structure-level partial reward is the strongest in PSJS at $88.11\%$. GTR exceeds them by $0.87$ and $0.72$ percentage points, respectively. These results indicate that the gain of GTR comes from grading the partially correct answers that graph pattern matching produces.

Table 4: Comparison of terminal rewards on CypherBench.

| Reward | EX (%) | PSJS (%) |
|---|---|---|
| Execution reward (Yao et al., 2026) | 80.81 | 86.53 |
| Execution-tiered reward (Su et al., 2025) | 80.13 | 86.60 |
| Text-level partial reward (Pourreza et al., 2025b) | 79.15 | 87.23 |
| Structure-level partial reward (Stoisser et al., 2025) | 80.00 | 88.11 |
| Result-level partial reward (Papicchio et al., 2026) | 81.54 | 88.10 |
| AgentCypher | 82.41 | 88.83 |

## 5 Conclusion

We have proposed AgentCypher, a reinforcement learning framework that trains a language model to revise and accept Cypher queries using execution results. Grounded Query Execution (GQE) executes candidate queries constructed from selected schema elements and grounded entities, and returns their results to the policy for revision. Execution-Guided Verification (EGV) runs additional check queries that report evidence on entity grounding, relationship direction, and result multiplicity, guiding the policy's revision and acceptance decisions. Graded Terminal Reward (GTR) compares the accepted query's results with reference results during training, penalizing missing, extraneous, and duplicate rows while keeping exact answers strictly preferred. Experiments on CypherBench and Neo4j-Text2Cypher show that AgentCypher outperforms the compared methods, and the ablations show that each module improves execution accuracy. These results suggest that successful execution alone does not justify accepting a query, and that evidence about how a query matches the graph helps a small model decide when to revise it.

## REFERENCES


Francesco Cazzaro, Jessica Lennon, and Ariadna Quattoni. Achieving precise text-to-cypher via grounded knowledge graph data generation. *arXiv preprint arXiv:2606.14325*, 2026. URL https://arxiv.org/abs/2606.14325.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/2460396f2d0d421885997dd1612ac56b-Abstract-Conference.html.

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, July 2026. doi: 10.18653/v1/2026.acl-long.35. URL https://aclanthology.org/2026.acl-long.35/.

DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. URL https://arxiv.org/abs/2412.19437.

Abhimanyu Dubey et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Yanlin Feng, Simone Papicchio, and Sajjadur Rahman. CypherBench: Towards precise retrieval over full-scale modern knowledge graphs in the LLM era. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8934–8958, 2025. doi: 10.18653/v1/2025.acl-long.438. URL https://aclanthology.org/2025.acl-long.438/.

Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, pp. 1433–1445. Association for Computing Machinery, 2018. doi: 10.1145/3183713.3190657. URL https://doi.org/10.1145/3183713.3190657.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: Grounding code LLMs in execution feedback with reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 19034–19055. PMLR, 2025. URL https://proceedings.mlr.press/v267/gehring25a.html.

Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Yue Gong, Chuan Lei, Xiao Qin, Kapil Vaidya, Balakrishnan Murali Narayanaswamy, and Tim Kraska. SQLens: An end-to-end framework for error detection and correction in text-to-SQL. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=on6Hf0KP20.

Harper Hua, Zhen Han, Zhengyuan Shen, Meng-Chieh Lee, Sheng Guan, Qi Zhu, Sullam Jeoung, Yueyan Chen, Yunfei Bai, Shuai Wang, Vassilis N. Ioannidis, and Huzefa Rangwala. SQL-Trail: Multi-turn reinforcement learning with interleaved feedback for text-to-SQL. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 36224–36246. Association for Computational Linguistics, July 2026. doi: 10.18653/v1/2026.acl-long.1677. URL https://aclanthology.org/2026.acl-long.1677/.

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

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Pratik Kakkar, Chandra Dhir, Ravi Shankar, Pareekshit Reddy Gaddam, and Anup Shirgaonkar. ReToolSQL: Agentic reinforcement learning for robust text-to-SQL, August 2026. URL https://arxiv.org/abs/2608.27796. Preprint.

Fang Niu, Chaokun Wang, Hang Zhang, and Songyao Wang. Adaptive Text2GQL: Integrating structural twig linking and evolutionary in-context learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17663–17680, 2026. doi: 10.18653/v1/2026.acl-long.804. URL https://aclanthology.org/2026.acl-long.804/.

OpenAI. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. Text2Cypher: Bridging natural language and graph databases. In *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*, pp. 100–108. International Committee on Computational Linguistics, 2025. URL https://aclanthology.org/2025.genaik-1.11/.

Simone Papicchio, Simone Rossi, Luca Cagliero, and Paolo Papotti. Think2SQL: Blueprinting reward density and advantage scaling for effective text-to-SQL reasoning. *Transactions on Machine Learning Research*, 2026. URL https://openreview.net/forum?id=NxU1KWnpOG.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan Arik. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *International Conference on Learning Representations*, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/974ff7b5bf08dbf9400b5d599a39c77f-Abstract-Conference.html.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan Ö. Arik. Reasoning-SQL: Reinforcement learning with SQL tailored partial rewards for reasoning-enhanced text-to-SQL. In *Second Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=HbwkIDWQgN.

Ioannis Prokopiou, Athanasios Aidinis, Panagiotis-Christos Kyrmpatsos, and Pantelis Vikatos. What drives recovery in agentic text-to-cypher? LAST-CQ: An LLM agent self-refinement framework. *arXiv preprint arXiv:2609.12746*, 2026. URL https://arxiv.org/abs/2609.12746. Accepted at the REALM Workshop at EMNLP 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. HybridFlow: A flexible and efficient RLHF framework. *arXiv preprint arXiv:2409.19256*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Josefa Lia Stoisser, Marc Boubnovski Martell, Lawrence Phillips, Casper Hansen, and Julien Fauqueur. STRuCT-LLM: Unifying tabular and graph reasoning with reinforcement learning for semantic parsing, 2025. URL https://arxiv.org/abs/2506.21575. Preprint.

Hanchen Su, Xuyuan Li, Yan Zhou, Zhuoyi Lu, Ziwei Chai, Haozheng Wang, Chen Zhang, and Yang Yang. Cypher-RI: Reinforcement learning for integrating schema selection into cypher generation. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL

https://papers.neurips.cc/paper_files/paper/2025/hash/906727703322b2e33b37c855fc15e8b7-Abstract-Conference.html.

Nikodem Tomczak. CYGNET: Cypher gate for neural execution triage and cost containment. *arXiv preprint arXiv:2606.04645*, 2026. URL https://arxiv.org/abs/2606.04645.

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. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zhewei Yao, Guoheng Sun, Łukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. Arctic-Text2SQL-R1: Simple rewards, strong reasoning in text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26966–26995. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1345. URL https://aclanthology.org/2026.findings-acl.1345/.

Zeyu Zhang, Kexuan Sun, Zheng Tang, Jens-S. Vöckler, Thien Huu Nguyen, and Thuy Vu. CypherSmith: Transforming text-to-cypher generation for LLMs with synthetic data. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 34665–34682, 2026. doi: 10.18653/v1/2026.acl-long.1601. URL https://aclanthology.org/2026.acl-long.1601/.

Yuhang Zhou, Yu He, Siyu Tian, Yuchen Ni, Zhangyue Yin, Xiang Liu, Chuanjun Ji, Sen Liu, Xipeng Qiu, Guangnan Ye, and Hongfeng Chai. $R^3$-NL2GQL: A model coordination and knowledge graph alignment approach for NL2GQL. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 13679–13692. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.800. URL https://aclanthology.org/2024.findings-emnlp.800/.

# A Implementation Details

The policy is Qwen2.5-Coder-7B-Instruct, trained with GRPO under verl (Sheng et al., 2024) on $4\times$ A800 GPUs with full-parameter optimization. Table 5 summarizes the training hyperparameters. The KL coefficient is zero, so equation 15 is the entire actor objective.

Table 5: Training hyperparameters for AgentCypher.

| Parameter | Value |
|---|---|
| Rollouts per question $N$ | 8 |
| Sampling temperature | 1.0 |
| Training batch size | 512 |
| Mini-batch size | 64 |
| Optimizer | AdamW |
| Learning rate | $1 \times 10^{-6}$ |
| Clip ratio $\epsilon$ | 0.2 |
| KL coefficient | 0 |
| Maximum prompt length (tokens) | 2,048 |
| Maximum response length (tokens) | 4,096 |

The episode allows $T = 5$ assistant turns and 8 query executions. Evaluation decodes at temperature 0 using the training prompt.

# B Additional Analyses

**Analysis of EGV.** EGV targets a failure mode specific to Cypher generation: a structurally valid query can return a nonempty but incorrect answer, for example by reversing a relationship direction, reaching the same node through several paths, or omitting a resolved entity. The verification gate is the review the policy is instructed to perform before `<accept/>`. It is specified in the prompt and does not programmatically block acceptance (§3). We compare the AgentCypher gate with three alternative gates on CypherBench (Table 6), keeping the trained policy, the shared first query, the action parser, and the remaining budgets fixed. Only the acceptance rule changes, and no gate uses the reference answer. None of the alternatives checks whether the result answers the question. The pre-execution structural gate (Tomczak, 2026) validates the query before execution, so it cannot catch a valid query that returns a wrong answer. The nonempty-result stop (Niu et al., 2026; Pourreza et al., 2025a) accepts the first query that returns rows, although a nonempty result need not be correct. The signal-clearance gate (Gong et al., 2025) accepts only after every reported signal is cleared, treating each signal as an error even when it does not affect the answer. The AgentCypher gate instead asks the policy to address each EGV finding in relation to the question: it revises the query when a finding reveals an error and accepts the query when the finding is consistent with the question. As a result, it achieves the highest EX and PSJS at $82.41\%$ and $88.83\%$, improving over the pre-execution structural gate by $3.90$ and $2.82$ percentage points. The signal-clearance gate is the strongest alternative at $80.15\%$ and $87.52\%$, which the AgentCypher gate exceeds by $2.26$ and $1.31$ percentage points, respectively. These results indicate that the gain of EGV comes from interpreting graph evidence against the question rather than from requiring a nonempty or signal-free result.

Table 6: Comparison of verification gates on CypherBench with fixed policy parameters. Repair is measured on shared first queries that return nonempty incorrect answers; harm is measured on initially correct queries. Settings that follow prior methods are our Cypher adaptations.

| Gate | EX (%) | PSJS (%) |
|---|---|---|
| Pre-execution structural gate (Tomczak, 2026) | 78.51 | 86.01 |
| Nonempty-result stop (Niu et al., 2026; Pourreza et al., 2025a) | 79.54 | 86.35 |
| Signal-clearance gate (Gong et al., 2025) | 80.15 | 87.52 |
| AgentCypher | 82.41 | 88.83 |

**Effect of the Interaction-Turn Budget.** Revising a Cypher query takes turns: the policy must execute a candidate, read the GQE and EGV observations, and revise the graph pattern before accepting. The turn limit $T$ therefore bounds how much revision the policy can learn and perform. We train a separate AgentCypher policy for each $T \in \{3, 4, 5, 6, 7\}$ and evaluate it with the same limit on CypherBench (Table 7), keeping all other training and inference settings fixed. Episodes stop upon valid acceptance, and those without acceptance receive EX$= 0$. Increasing $T$ from 3 to 5 raises EX, PSJS, and Exec. by $8.18$, $11.43$, and $14.49$ percentage points, respectively. The low Exec. of $85.38\%$ at $T = 3$ suggests that three turns leave too little room to repair failed queries before acceptance. Increasing $T$ from 5 to 7 adds only $0.52$ and $0.29$ percentage points in EX and PSJS, and Exec. is already near $100\%$ at $T = 5$. These results indicate that five turns suffice for most revisions, so the default $T = 5$ captures most of the gain within the tested range.

Table 7: Effect of the inference turn limit on CypherBench with fixed policy parameters and token caps.

| Turn limit $T$ | EX (%) | PSJS (%) | Exec. (%) |
|---|---|---|---|
| 3 | 74.23 | 77.40 | 85.38 |
| 4 | 80.14 | 85.90 | 95.27 |
| 5 | 82.41 | 88.83 | 99.87 |
| 6 | 82.58 | 88.97 | 99.91 |
| 7 | 82.93 | 89.12 | 99.90 |

# C Case Studies and Trajectory Analysis

This appendix shows four test trajectories of AgentCypher and then analyses all test trajectories. The trajectories come from the evaluation reported in Table 1 (checkpoint after $160$ training updates, decoding at temperature $0$). Tables 8–11 at the end of this appendix list the trajectories turn by turn. The first row gives the question, and horizontal rules separate turns. A turn contains one policy action and the environment reply to it. The system prompt with the graph schema is omitted. Policy actions and environment replies are reproduced verbatim, and execution accuracy is computed by the official scorer. Cases 1, 2, and 4 end with a correct answer after revision, and Case 3 ends with an incorrect answer.

## C.1 Cases

**Case 1: relationship direction under an aggregate.** Table 8 shows a company-graph question. In the schema, `hasCEO` points from `Company` to `Person`. The entity lookup for "CEO" returns only unrelated names, and the policy does not use any of them. The first query in turn 2 follows the arrow from `Person` to `Company`. The pattern has no match, but `count` still returns one row, with value $0$. Execution therefore succeeds with a nonempty result. EGV reports that the reversed pattern returns rows, so the returned row does not settle the direction. In turn 3 the policy changes only the projection and receives the same reply. In turn 4 it reverses the arrow. The query returns $4{,}319$, and EGV reports that the reversed pattern now returns no rows. The policy accepts this query in turn 5, the last turn of the budget, and the accepted answer matches the reference answer.

**Case 2: repeated rows.** Table 9 asks for the spouses of the children of Maggie Murdock. The first query returns Milla Donovan twice, and the preview alone does not show whether the two rows are one person or two people. EGV reports that the two rows come from one node, so the same node is reached by more than one path. The policy adds `WITH DISTINCT s`, the result becomes one row, and the policy accepts it. The reference answer contains one row, so the first result would have been incorrect under execution accuracy.

**Case 3: an unchecked temporal condition.** Case 3 receives the same repetition report as Case 2 but ends with an incorrect answer. Table 10 shows an NBA-graph question about the team LeBron James is *currently* playing for. In the schema, `playsFor` carries `start_year` and `end_year`. The reference query keeps only relationships with a start year and no end year. The first query returns every team LeBron James has played for. EGV reports that the four rows come from three

team nodes, so one team is reached by more than one path. In this one-hop pattern from a fixed player, the repeated team corresponds to more than one `playsFor` relationship to that team. The policy removes the repetition with `WITH DISTINCT t` and accepts three teams. The answer includes the current team, but it also includes two former teams, so the answer is incorrect. The checks report direction and repetition. None of them tests the temporal condition "currently", and the policy does not add a condition on `end_year`.

**Case 4: wrong relationship type.** Case 4 returns to a revision that ends correctly, this time from an empty first result. Table 11 asks for schools attended by heads of government of countries where the Christian National Union operates. The first query connects the party and the country through `memberOf` and returns no rows. EGV localises the failure to this hop and reports that the nodes exist but the relationship between them does not. The policy removes the direction of the arrow, and EGV reports that the hop now matches only when its relationship type is dropped. The policy then replaces `memberOf` with `operatesIn`, which the schema defines from `PoliticalParty` to `Country`. The query returns 29 schools and matches the reference answer.

### C.2 Trajectory analysis

The analysis covers all 2,348 test trajectories behind Table 1. We classify each check result by the fixed sentences EGV emits, for example "It does match when its arrows are reversed". A trajectory counts as revised when it tries at least two distinct queries. To judge the first query, we compare its result with the reference answer using the official comparator. This is possible when the reply lists every row or returns no rows, which holds for 1,602 trajectories.

**Revision and acceptance.** The policy tries a single distinct query in 1,689 trajectories (71.9%) and reaches 90.2% execution accuracy on them. It revises in the remaining 659 trajectories, where the accuracy is 62.4%. Revised trajectories start from harder first queries rather than lose accuracy through revision. Among the 1,602 trajectories with a judged first query, 1,113 first queries are already correct. The policy revises 108 of them, and 2 of these end incorrectly. Among the 489 incorrect first queries, 185 end with a correct accepted answer.

**Empty results.** The first query returns no rows in 278 trajectories, and every one of these questions has a nonempty reference answer. The policy revises 276 of them, and 140 end correctly. The path check reports a failing hop in 231 of these trajectories, and 128 end correctly (55.4%). The outcome depends on the reported change. When the hop matches with its arrows reversed, 91 of 121 end correctly (75.2%). When it matches only with its relationship type dropped, 15 of 43 end correctly, and when it matches only with its name literals dropped, 5 of 21 end correctly. When the check reports that the `MATCH` pattern matches and a later clause empties the result, 10 of 43 end correctly (23.3%). This report localises the failure to a filter but does not name the property the filter should use. The two reports with the lowest success rates, a dropped literal and a later clause, both concern values in the question rather than the graph structure.

**Nonempty results.** The first query returns rows in 2,059 trajectories. For 1,536 of them, no check reports a fact. The policy then revises only 26 times, and 92.3% end correctly. For the other 523, at least one check reports a fact, and the policy revises 347 of them (66.3%). The response depends on the fact. The policy revises after 178 of 187 reports that rows repeat a node. It revises after 58 of 179 reports that the reversed pattern also matches, since this fact does not show that the original direction is wrong.

**Remaining errors.** Of the 413 incorrect trajectories, 116 end without acceptance. In 114 of these, the policy tries four queries and the episode reaches the turn budget. The other 297 accept an incorrect answer. In 185 of them, no check reports a fact about the accepted query. By question type, these 185 are led by unions (52), optional matches (39), two-hop paths (28), and time-sensitive conditions (21). In such cases the result is well formed, and its error concerns an operator or condition the checks do not test, as in Case 3. Among the other 112, the most frequent last report is that the reversed pattern also matches (83).

### C.3 DISCUSSION

**What the three correct cases share.** In Cases 1, 2, and 4 the first query executes without error, yet its result is incorrect. In Cases 1 and 2 the result is also nonempty: a count of 0 and a list with a repeated name. In each case the execution status and the result preview do not show the error. The check result names a concrete property of the result: a direction that both patterns match, a node reached by several paths, or a hop whose edge does not exist between existing nodes. The next query changes exactly that element, and the reply to the accepted query no longer reports the fact. This pattern is not limited to the three cases. After a report that a hop matches with its arrows reversed, 119 of 121 next queries change a relationship direction. After a report that rows repeat a node, all 178 revised next queries change their deduplication. Among the 185 trajectories whose incorrect first query ends with a correct answer, 140 start from an empty result, as in Case 4, and 40 start from a nonempty result, as in Cases 1 and 2.

**Why the check results matter for acceptance.** The three cases also show how acceptance follows the check results. In each of them, the policy accepts only after the reply to its latest query reports no remaining fact, or reports that the reversed pattern returns no rows. Case 1 shows why a nonempty result alone is not a reason to accept: the aggregate turns a failed match into a single row. The same reply appears in turns 2 and 3 because only the projection changed, and the policy reverses the arrow in turn 4. This required the full turn budget, so the example also shows that a repeated reply costs a turn.

**Limits shown by the incorrect case.** The checks describe the result and the pattern. They do not test whether the query expresses every condition in the question. In Case 3 the repetition report leads to a correct edit, and the accepted query is internally consistent, but it lacks the temporal condition. Such errors form the largest group of remaining errors: in 185 of the 297 accepted incorrect answers no check reports a fact about the accepted query. These counts describe the trajectories of one trained policy. They do not isolate the contribution of EGV, which requires the controlled comparisons described in §4.4.

### C.4 CASE TRAJECTORIES

Table 8: Case 1, relationship direction under an aggregate (correct after revision). Reference answer: 4,319.

| Turn | Speaker | Content |
|---|---|---|
| Question | | How many individuals have served as CEOs of companies at any time, whether in the past or present? |
| 1 | Policy | `<entities>["CEO"]</entities>` |
| | Environment | `'CEO': NOT stored under this exact name. Real names that match: 'CEOs Ltd.' (Company), 'CEO Tony Vũ' (Person), 'Ceotronics' (Company), 'ceo left bd' (Person), 'Ceol na dTonnta' (Company), 'Bof Hotels Ceo Suites Atasehir' (Company), 'Ruineng Ceo International Hotel' (Company), 'Hotel Ceolatto Palace - Aeroporto' (Company)` |
| 2 | Policy | `<try>`<br>`MATCH (p:Person)-[:hasCEO]->(c:Company)`<br>`RETURN count(DISTINCT p)`<br>`</try>` |
| | Environment | `OK -- 1 rows. Reversing every arrow in this pattern also returns rows (4664) -- getting rows back does not settle the direction. First 1: [{"count(DISTINCT p)": 0}]` |
| 3 | Policy | `<try>`<br>`MATCH (p:Person)-[:hasCEO]->(c:Company)`<br>`WITH DISTINCT p`<br>`RETURN count(p)`<br>`</try>` |
| | Environment | `OK -- 1 rows. Reversing every arrow in this pattern also returns rows (4664) -- getting rows back does not settle the direction. First 1: [{"count(p)": 0}]` |
| 4 | Policy | `<try>`<br>`MATCH (p:Person)<-[:hasCEO]-(c:Company)`<br>`WITH DISTINCT p`<br>`RETURN count(p)`<br>`</try>` |
| | Environment | `OK -- 1 rows. Reversing every arrow in this pattern returns no rows -- the direction is fixed by the data. First 1: [{"count(p)": 4319}]` |
| 5 | Policy | `<accept/>` |

Table 9: Case 2, repeated rows (correct after revision). Reference answer: Milla Donovan.

| Turn | Speaker | Content |
|---|---|---|
| Question | | What are the names of the spouses of individuals whose mother is Maggie Murdock? |
| 1 | Policy | `<entities>["Maggie Murdock"]</entities>` |
| | Environment | `'Maggie Murdock': stored exactly as written (Character).` |
| 2 | Policy | `<try>`<br>`MATCH (c:Character {name: 'Maggie Murdock'})<-[:hasMother]-(m:Character)-[:hasSpouse]-(s:Character)`<br>`RETURN s.name`<br>`</try>` |
| | Environment | `OK -- 2 rows. Reversing every arrow in this pattern returns no rows -- the direction is fixed by the data. 2 rows come from 1 distinct nodes -- the same node is reached by more than one path. First 2: [{"s.name": "Milla Donovan"}, {"s.name": "Milla Donovan"}]` |
| 3 | Policy | `<try>`<br>`MATCH (c:Character {name: 'Maggie Murdock'})<-[:hasMother]-(m:Character)-[:hasSpouse]-(s:Character)`<br>`WITH DISTINCT s`<br>`RETURN s.name`<br>`</try>` |
| | Environment | `OK -- 1 rows. Reversing every arrow in this pattern returns no rows -- the direction is fixed by the data. First 1: [{"s.name": "Milla Donovan"}]` |
| 4 | Policy | `<accept/>` |

Table 10: Case 3, an unchecked temporal condition (incorrect). Reference answer: Los Angeles Lakers.

| Turn | Speaker | Content |
|---|---|---|
| Question | | What are the names of the teams that LeBron James is currently playing for? |
| 1 | Policy | `<entities>["LeBron James"]</entities>` |
| | Environment | `'LeBron James': stored exactly as written (Player).` |
| 2 | Policy | `<try>`<br>`MATCH (p:Player {name: 'LeBron James'})-[:playsFor]->(t:Team)`<br>`RETURN t.name`<br>`</try>` |
| | Environment | `OK -- 4 rows. Reversing every arrow in this pattern returns no rows -- the direction is fixed by the data. 4 rows come from 3 distinct nodes -- the same node is reached by more than one path. First 3: [{"t.name": "Miami Heat"}, {"t.name": "Cleveland Cavaliers"}, {"t.name": "Cleveland Cavaliers"}]` |
| 3 | Policy | `<try>`<br>`MATCH (p:Player {name: 'LeBron James'})-[:playsFor]->(t:Team)`<br>`WITH DISTINCT t`<br>`RETURN t.name`<br>`</try>` |
| | Environment | `OK -- 3 rows. Reversing every arrow in this pattern returns no rows -- the direction is fixed by the data. First 3: [{"t.name": "Miami Heat"}, {"t.name": "Cleveland Cavaliers"}, {"t.name": "Los Angeles Lakers"}]` |
| 4 | Policy | `<accept/>` |

Table 11: Case 4, wrong relationship type (correct after revision). Reference answer: 29 distinct schools.

| Turn | Speaker | Content |
|---|---|---|
| Question | | What are the unique schools attended by politicians who have served as heads of government in countries where the Christian National Union operates? |
| 1 | Policy | <entities>["Christian National Union"]</entities> |
| | Environment | 'Christian National Union': stored exactly as written (PoliticalParty). |
| 2 | Policy | <try><br>MATCH (pp:PoliticalParty {name: 'Christian National Union'})<-[:memberOf]-(c:Country)-[:hasHeadOfGovernment]-(p:Politician)<br>UNWIND p.schools_attended AS school<br>WITH DISTINCT school<br>RETURN school<br></try> |
| | Environment | OK -- 0 rows. Nothing in the graph matches `(pp:PoliticalParty {name: 'Christian National Union'})<-[:memberOf]-(c:Country)`. The nodes it names do exist -- it is the relationship between them that the graph does not have. |
| 3 | Policy | <try><br>MATCH (pp:PoliticalParty {name: 'Christian National Union'})-[:memberOf]-(c:Country)-[:hasHeadOfGovernment]-(p:Politician)<br>UNWIND p.schools_attended AS school<br>WITH DISTINCT school<br>RETURN school<br></try> |
| | Environment | OK -- 0 rows. Nothing in the graph matches `(pp:PoliticalParty {name: 'Christian National Union'})-[:memberOf]-(c:Country)`. It does match when its relationship type is dropped. |
| 4 | Policy | <try><br>MATCH (pp:PoliticalParty {name: 'Christian National Union'})-[:operatesIn]-(c:Country)-[:hasHeadOfGovernment]-(p:Politician)<br>UNWIND p.schools_attended AS school<br>WITH DISTINCT school<br>RETURN school<br></try> |
| | Environment | OK -- 29 rows. First 3: [{"school": "Faculty of History, University of Gda\u0144sk"}, {"school": "I Liceum Og\u00f3lnokszta\u0142c\u0105ce im. Miko\u0142aja Kopernika w Gda\u0144sku"}, {"school": "Faculty of Law and Administration of Adam Mickiewicz University in Pozna\u0144"}] |
| 5 | Policy | <accept/> |

## AI USE STATEMENT

An AI assistant supported code inspection, literature synthesis, drafting and revision of the title, abstract, contribution statements, related work, ablation protocols and manuscript framing, and preparation of the LaTeX source. It also assisted in generating and editing the illustrative figures. The complete disclosure and author verification remain to be finalized.