# Learning to Refine: An Agentic RL Approach for Iterative SPARQL Query Construction

Floris Vossebeld[1,2], Shenghui Wang[1]

[1]Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Drienerlolaan 5, 7522 NB Enschede, The Netherlands

[2]Microsoft Netherlands, Evert van de Beekstraat 354, 1118 CZ Schiphol, The Netherlands

**Abstract**

Generating complex, logically-sound SPARQL queries for multi-hop questions remains a critical bottleneck for Knowledge Graph Question Answering, as the brittle nature of one-shot generation by Large Language Models (LLMs) hinders reliable interaction with structured data. Current methods lack the adaptive policies needed to dynamically debug queries based on real-time execution feedback. This paper introduces a novel agentic framework where an LLM learns a resilient policy for the sequential process of iterative SPARQL construction. We show that a compact 3B-parameter model, trained exclusively via outcome-driven Reinforcement Learning (GRPO) without supervised fine-tuning, can learn effective policies for this task, discovering how to systematically recover from execution errors and refine its queries toward a correct answer. On a curated, executable single-answer subset of LC-QuAD 2.0, our agent achieves 49.7% accuracy post-entity-linking, a significant 17.5 percentage point improvement over the strongest iterative zero-shot baseline. Further analysis reveals that while the agent's capability is driven by RL, its performance is enhanced by an explicit deliberative reasoning step that acts as a cognitive scaffold to improve policy precision. This work presents a generalizable blueprint for teaching agents to master formal, symbolic tools through interaction, bridging the gap between probabilistic LLMs and the structured world of Knowledge Graphs.

**Keywords**

Knowledge Graph Question Answering, Agentic Language Models, SPARQL Query Generation, Reinforcement Learning, Iterative Query

## 1. Introduction

Knowledge Graphs (KGs) like DBpedia [1] and Wikidata [2] structure vast information as entities linked by relations [3]. Answering natural language questions using these graphs, Knowledge Graph Question Answering (KGQA), is vital for applications from search to decision support [4, 5]. While single-hop KGQA is well studied, multi-hop questions, requiring reasoning across multiple entities and relations, remain a major challenge due to combinatorial path explosion [6], KG incompleteness [7], and semantic ambiguity [8].

Traditional KGQA methods such as semantic parsing [9], retrieval-based approaches [10], and KG embeddings [11] often apply static reasoning strategies. These methods typically generate a SPARQL query in a single pass or retrieve a fixed subgraph, lacking mechanisms for iterative refinement. As a result, they struggle with long or error-prone queries, and fail to recover from intermediate execution failures.

Large Language Models (LLMs) have recently shown strong capabilities in structured reasoning, particularly when supported by techniques like Chain-of-Thought prompting [12]. Agentic frameworks such as ReAct [13] and StructGPT [14] allow LLMs to combine reasoning with tool use. However, many of these methods either rely on predefined tools or prompting schemes, and do not learn adaptive interaction policies through feedback.

*RAGE-KG 2025: The Second International Workshop on Retrieval-Augmented Generation Enabled by Knowledge Graphs, co-located with ISWC 2025, November 2–6, 2025, Nara, Japan*

†Work performed while the author was an Intern at Microsoft Netherlands (Feb-Jul 2025). The views expressed are the author's and do not necessarily reflect those of Microsoft Corporation.

f.r.vossebeld@student.utwente.nl (F. Vossebeld); shenghui.wang@utwente.nl (S. Wang)

0000-000X-XXXX-XXXX (F. Vossebeld); 0000-0003-0583-6969 (S. Wang)

Consider the question "Which actors starred in movies directed by the director of *Inception*?" Answering this requires identifying the film, finding its director, retrieving the director's other films, and then the actors in those films, each step relying on correct schema navigation, relation selection, and entity disambiguation. Translating this full path into a correct SPARQL query in one shot is error-prone. An alternative is to incrementally build and test parts of the query, adapt based on results, and correct mistakes mid-way, requiring interaction with the KG as a semantic environment.

While integrating LLMs with KGs is actively researched [15, 16, 14], methods often rely on fixed interaction logic, prompting strategies applied to base models, or using KG information primarily as retrieved context. They typically do not involve fine-tuning the model specifically to learn adaptive policies for the iterative construction of structured SPARQL queries based on execution feedback, which is crucial for handling the complexity and potential errors inherent in multi-step KG interactions.

This paper addresses this gap by proposing an agentic framework in which a language model learns a policy for iterative SPARQL query construction through interaction with a knowledge graph. The agent operates in a think–act–observe loop: it reasons about the current state (`<think>`), generates a SPARQL query or final answer (`<query>`, `<answer>`), and receives execution feedback from the KG (`<query_result>`). Rather than relying on static, one-shot generation, the agent adapts its strategy based on results, including errors or empty outputs, progressively refining its queries. To enable this behavior, we fine-tune a compact LLM using Group Relative Policy Optimization (GRPO), a reinforcement learning algorithm designed for sparse, outcome-based rewards. The agent learns not only to generate queries, but to interpret feedback and dynamically debug or explore, improving robustness in complex multi-hop scenarios.

This motivates the following research questions:

Research questions

**RQ1:** How can an LLM learn to *iteratively* build and refine SPARQL queries using execution feedback to answer complex multi-hop KG questions?

**RQ2:** Can reinforcement learning effectively train such an agent to produce accurate answers from outcome signals alone?

**RQ3:** How does this iterative, RL-guided approach compare with static or prompt-only baselines on the LC-QuAD 2.0 benchmark?

The remainder of this thesis details related work (§2), presents our methodology (§3), details the experiments and results (§4 and §5), and discusses the findings and discussion and conclusions (§6).

## 2. Related work

Multi-hop KGQA requires combining structured reasoning, language understanding, and interaction. We review prior work on traditional KGQA methods, agentic LLMs, and reinforcement learning, highlighting how our approach integrates symbolic interaction, tool-based reasoning, and adaptive query construction to address the unique challenges of multi-hop KGQA.

**Multi-hop KGQA and traditional approaches** Knowledge Graph Question Answering (KGQA) maps natural language questions to structured answers by reasoning over triples in a knowledge graph (KG) [5, 10]. While single-hop questions can be resolved through direct relations, multi-hop KGQA requires compositional reasoning across multiple entities and relations [17, 18]. This increases complexity due to path explosion [6], KG incompleteness and noise [7], and semantic ambiguity [8].

Traditional KGQA methods fall into three categories: semantic parsing, retrieval-based approaches, and embedding-based reasoning. Semantic parsing methods aim to generate formal queries such as SPARQL [9, 19], but are brittle to linguistic variation and require substantial supervision [20, 21]. Retrieval-based methods extract subgraphs for ranking [10, 6] but often struggle with complex logic.

Embedding-based approaches reason in vector space [11, 17], sacrificing interpretability and logical precision. Critically, these approaches apply fixed computation and lack iterative refinement mechanisms based on intermediate feedback, hindering performance on complex multi-hop tasks.

**Agentic LLMs and symbolic interaction in KGQA** Recent work leverages LLMs as agents that combine internal reasoning with external actions. Agentic frameworks such as ReAct [13] and MRKL [22] allow LLMs to operate in a loop of `<think>` → `<act>` → `<observe>`, interacting with tools to solve complex tasks. In KGQA, systems like StructGPT [14] and Think-and-Graph [23] extend this idea by giving LLMs access to navigation tools (e.g., retrieving neighbors or relations). However, these tools are often predefined, and reasoning policies are static or heuristic-driven, limiting adaptivity.

Our work shifts the focus from tool-based navigation to formal query generation. Inspired by ARTIST [24], we treat SPARQL construction as the agent's primary action. The model alternates between `<think>`, `<query>`, and `<answer>` tags, learning to refine its reasoning through symbolic interaction with the KG. This reframes KGQA as a dynamic decision-making process grounded in executable feedback.

This strategy also relates to test-time compute scaling [25], where additional reasoning effort is allocated adaptively. Some approaches use inference-time sampling or search [26, 27]; others explicitly train models to optimize reasoning under compute constraints [28, 29]. Our work falls in the latter category, focusing on training an agent to effectively use interaction cycles for symbolic query refinement.

**Reinforcement learning for iterative query generation** While supervised fine-tuning enables LLMs to imitate reasoning (e.g., Chain-of-Thought prompting [30]), it relies heavily on high-quality demonstrations and struggles with long-horizon credit assignment. Reinforcement learning (RL) offers a more flexible alternative, enabling agents to learn from outcome-based interaction.

We build on *Group Relative Policy Optimization (GRPO)*, a recent RL algorithm designed for sparse, symbolic environments. GRPO has shown success in math problem solving [28], SQL generation [31], and general tool use [24]. In our setting, GRPO allows a compact LLM to learn symbolic refinement strategies from task-level rewards alone, recovering from syntax errors, adapting query structure, and issuing exploratory probes. This enables robust multi-hop reasoning without requiring step-level supervision or hand-coded recovery heuristics.

## 3. Methodology

Our approach transforms multi-hop KGQA from a one-shot generation task into an iterative, sequential decision-making problem. We developed an autonomous agent, powered by a Large Language Model (LLM), that learns an optimal policy for constructing and refining SPARQL queries through live interaction with a Knowledge Graph (KG). The agent operates within a Reinforcement Learning (RL) framework, where its behavior is optimized to maximize a reward signal reflecting the accuracy and validity of its actions.

The agent's core is an interaction loop, conceptually illustrated in Figure 1. In each turn, the agent: 1) **analyzes** the history of the task, including the initial question and all previous KG interactions; 2) **reasons** about the next best step within a `<think>` block; and 3) **acts** by generating either a new SPARQL query (`<query>`) or a final answer (`<answer>`). This cycle repeats until the agent confidently terminates the process. This section details the formal problem definition, the mechanics of the agent-environment interaction, and the RL-based training process used to learn the query refinement policy.

### 3.1. Formalism: an agentic Markov Decision Process

We model the iterative query construction task as a finite-horizon Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$.

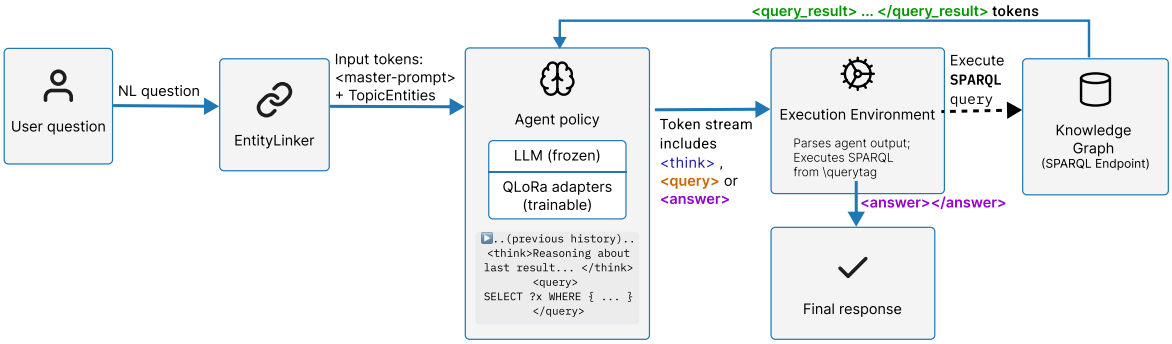



**Figure 1:** The end-to-end agentic inference loop. The agent policy (LLM + QLoRA adapters) receives the current state (question and history) and generates reasoning (`<think>`) followed by an action: either a SPARQL query (`<query>`) or the final answer (`<answer>`). The SPARQL query is executed against the KG, with the outcome (`<query_result>`) updating the state for the next iteration. The loop terminates when the agent produces an answer.

**Table 1**
Key components of the agentic KGQA framework.

| Component | Role and implementation |
|---|---|
| **Agent policy** ($\pi_\theta$) | An instruction-tuned LLM (`Qwen2.5-3B-Instruct`) with QLoRA adapters ($\theta$) that generates think-act sequences. |
| **Environment** | Executes SPARQL queries against a self-hosted Wikidata endpoint and returns structured feedback (`<query_result>`). |
| **State** ($s_t$) | The full, structured conversation history, including all prior agent actions and environment observations. |
| **Action** ($a_t$) | A text generation containing a `<think>` block followed by either a `<query>` or `<answer>` block. |
| **Reward** ($R(\tau)$) | A terminal, composite reward signal evaluating structural validity, final answer correctness, and efficiency. |
| **Learning algorithm** | Group Relative Policy Optimization (GRPO) to fine-tune the policy $\pi_\theta$ based on the outcome-based reward $R(\tau)$. |

**State ($s_t \in \mathcal{S}$)** A state is the full conversation history at turn $t$. It is a structured text sequence comprising the initial prompt, the user's question, and all subsequent agent turns and environment observations: $s_t = (\text{prompt}, m_1, o_1, m_2, o_2, \ldots)$.

**Action ($a_t \in \mathcal{A}$)** An action is the complete text sequence generated by the agent in a single turn. It must conform to one of two valid structures: a reasoning block followed by a query, or a reasoning block followed by a final answer.

$$a_t = \begin{cases} \texttt{<think>}\ldots\texttt{</think>}\ \texttt{<query>}\ldots\texttt{</query>} \\ \texttt{<think>}\ldots\texttt{</think>}\ \texttt{<answer>}\ldots\texttt{</answer>} \end{cases}$$

The action space $\mathcal{A}$ is the vast set of all possible text generations that adhere to this format.

**Transition function ($\mathcal{P}$)** The state transition $P(s_{t+1} \mid s_t, a_t)$ is largely deterministic. Given state $s_t$ and a query action $a_t$, the environment executes the SPARQL query from $a_t$. The resulting observation $o_{t+1}$ (the content of the `<query_result>` block) is appended to the history to form the next state

$s_{t+1} = s_t \circ a_t \circ o_{t+1}$. The episode terminates if the agent produces an answer action, exceeds the maximum number of turns, or generates a malformed action.

**Reward function ($\mathcal{R}$)** The reward $R(\tau)$ is a terminal, outcome-based reward assigned at the end of a full trajectory $\tau$. It is a composite signal designed to evaluate the success of the agent's multi-turn strategy, as detailed in Section 3.3.1.

**Policy ($\pi_\theta$)** The agent's policy is the LLM itself, parameterized by a set of trainable QLoRA adapter weights $\theta$. The policy $\pi_\theta(a_t \mid s_t)$ maps the current state (history) to a probability distribution over the action space. Our objective is to find the optimal weights $\theta^*$ that maximize the expected terminal reward.

## 3.2. Agent–environment loop

One interaction turn

1. **Think → Act** : $\pi_\theta$ appends a `<think>` block *plus* either `<query>` (SPARQL) or `<answer>`.
2. **Environment** :
   - 2.1. `<answer>` ⇒ episode ends.
   - 2.2. `<query>` ⇒ KG executes; reply comes back as `<query_result>`.
   - 2.3. malformed output ⇒ abort and error flag.
3. Loop until success or turn limit $T_{\max}$.

### 3.2.1. Knowledge-graph execution environment

For RL we require a fast, quota-free SPARQL endpoint. We therefore deploy a containerised *qEndpoint* (truthy Wikidata HDT) inside our Azure VNet. A lightweight `aiohttp` client issues queries asynchronously, with an in-memory LRU cache, pre-flight syntax checks (`rdflib`), and automatic retry/back-off. The result is a private, low-latency endpoint that sustains the thousands of queries demanded by training.

### 3.2.2. Agent Prompting and Structured Actions.

The agent's policy is guided by a detailed system prompt, which provides task instructions, defines the required interaction format, and includes few-shot examples of successful refinement trajectories.

A critical technique during RL training is **loss masking**. The policy's parameters $\theta$ are updated only based on the log-probabilities of tokens generated by the agent (i.e., within `<think>`, `<query>`, and `<answer>`). Tokens from the environment (the initial prompt and all `<query_result>` blocks) are masked out from the loss calculation. This follows best practices from frameworks like ARTIST [24] and focuses the learning signal squarely on the agent's decision-making policy, rather than wasting capacity trying to predict deterministic environment outputs.

## 3.3. Policy optimization via Group Relative Policy Optimization (GRPO)

We used Reinforcement Learning to fine-tune the agent's policy, specifically choosing Group Relative Policy Optimization (GRPO) [28]. This allows the agent to learn complex, sequential strategies from interactive experience and sparse, outcome-based rewards.

We fine-tuned the agent's policy using Group Relative Policy Optimization (GRPO) [28], a reinforcement learning algorithm well-suited to sparse, outcome-based tasks. GRPO compares the terminal reward of each trajectory against others in a group sampled from the same prompt, using relative performance to compute an advantage signal. This enables learning effective query refinement strategies without requiring a learned value function.

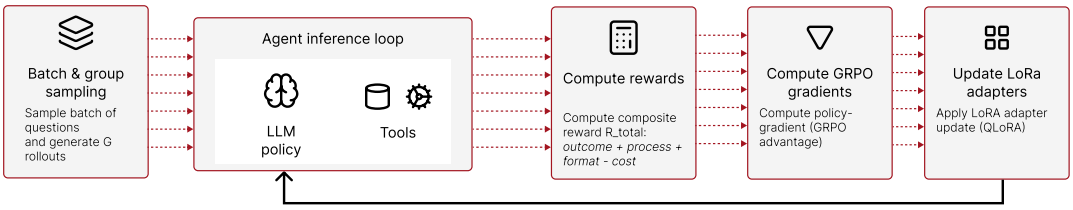



**Figure 2:** The end-to-end RL fine-tuning cycle. A batch of questions is sampled, and for each, the agent (LLM + LoRA adapters) generates $G$ rollouts using the iterative inference loop. The composite reward $R(\tau)$ is computed for each trajectory. GRPO uses these rewards to calculate a policy gradient, which is then used to update the LoRA adapter weights.

### 3.3.1. Reward design for effective learning

Terminal reward $R(\tau)$

$$R(\tau) = \begin{cases} -1, & \text{if trajectory } \textit{not} \text{ structurally valid,} \\ 1 \;+\; R_{\text{ans}}(\tau) \;-\; \big(0.1\, N_{\text{err}} + 0.02\, T\big), & \text{otherwise.} \end{cases}$$

**Structural validity** correct tag format and termination with `<answer>`.

**Answer term** $R_{\text{ans}}(\tau) = \begin{cases} +0.5 & \text{if judge deems the answer correct,} \\ -0.2 & \text{otherwise} \end{cases}$

**Cost term** $0.1\, N_{\text{err}} + 0.02\, T$, where $N_{\text{err}}$ is the number of failed SPARQL executions and $T$ the number of agent turns.

### 3.3.2. Training Protocol and Implementation.

The policy $\pi_\theta$ was fine-tuned using QLoRA [32] with the Unsloth library's optimizations for memory and speed. The training, depicted in Figure 2, was executed on an Azure ML compute cluster with NVIDIA H100 GPUs. The GRPO training loop proceeds as follows:

1. **Rollout generation:** Sample a batch of questions. For each question, generate $G = 16$ full agentic trajectories using the current policy $\pi_\theta$.
2. **Reward calculation:** Compute the composite reward $R(\tau)$ for each of the $G$ trajectories.
3. **Policy update:** Use the GRPO objective to calculate the policy gradient, where trajectories with a reward greater than their group's average contribute positively.
4. **Parameter update:** Update the LoRA adapter weights $\theta$ via the AdamW optimizer, applying loss masking and KL-divergence regularization to maintain stability.

This cycle repeats, progressively improving the agent's policy.

## 3.4. Design choices and scope

Several key design choices shaped this research. A key design choice was to proceed directly to RL fine-tuning, deliberately bypassing a supervised fine-tuning (SFT) phase on gold trajectories. This tests a critical hypothesis for the field: *can the combination of a powerful base model's instruction-following ability and a well-designed RL reward signal be sufficient to learn complex, tool-using behaviors, thereby reducing the dependency on costly, expert-curated demonstration data?* Second, our reward function's heavy penalty for structural and execution errors was a deliberate choice to force the agent to prioritize

generating valid and executable SPARQL above all else. Finally, we must acknowledge two critical scope limitations that bound our claims: our system's performance is evaluated on a curated subset of single-answer questions, and it relies on pre-linked entities provided in the dataset. We did not address the significant challenges of entity linking or multi-answer aggregation, which remain out of scope for this thesis.

# 4. Experimental Setup

This section outlines the experimental setup used to evaluate our agentic reinforcement learning approach to multi-hop KGQA. We describe the dataset curation process, model configuration, and knowledge graph environment, followed by training details and a report on compute and energy usage for reproducibility. We then present the comparative baselines and the evaluation methodology used to assess performance.

## 4.1. Dataset

Following §3.2.1, we re-execute every gold query of LC-QuAD 2.0 against the frozen 2023-12-21 Wikidata HDT dump and keep a triple $\langle q, \mathit{query}, \mathit{answer} \rangle$ only if the query (i) succeeds, (ii) returns exactly one row, and (iii) yields a valid RDF term. The resulting corpus preserves entity, literal, and boolean answers while discarding noisy items (Table 2).

**Table 2**
Curated single-binding subset of LC-QuAD 2.0.

| Split | Original size | Curated size |
|---|---|---|
| Train | 19 344 | 5 112 |
| Test | 4 836 | 1 279 |

## 4.2. Core models and KG environment

We use the `unsloth/Qwen2.5-3B-Instruct-bnb-4bit` model[1], selected for its instruction-following quality and compatibility with 4-bit QLoRA fine-tuning. Parameter-efficient training is performed using Unsloth's QLoRA implementation with commonly used hyperparameters: rank 64, $\alpha = 16$, dropout 0.05, learning rate $5 \times 10^{-6}$, group size $G = 16$, KL coefficient $\beta = 0.04$, and batch size 128. These values were selected through light, manual trial-and-error only; No systematic hyperparameter tuning was conducted.

The agent interacts with a private SPARQL endpoint (qEndpoint v2.5.2) loaded with the 2023-12-21 "truthy" HDT dump of Wikidata. All SPARQL queries are executed asynchronously with memoization, exponential backoff, and a 3-second timeout.

## 4.3. Training and compute setup

We fine-tune the agent using Group Relative Policy Optimization (GRPO), a reinforcement learning algorithm well-suited for sparse, outcome-based rewards. At each update step, $G = 16$ full roll-outs are sampled for each of 128 training questions. Terminal rewards are computed based on final answer correctness (see Section 3.3.1), and the LoRA weights are optimized using AdamW with a learning rate of $5 \times 10^{-6}$ and KL coefficient $\beta = 0.04$. Each interaction episode is capped at ten `<think>`–`<query>` cycles, and individual SPARQL executions are limited to a 3-second timeout.

Training is performed over a single epoch, converging in 11.5 hours on an NVIDIA H100 GPU (94 GB). This process consumed approximately 4.6 kWh of energy, which corresponds to an estimated 1.7

[1]Model commit ID `2672b58` on the HuggingFace Hub

kg $CO_2$e under the 2024 Dutch grid emission factor (0.37 kg $CO_2$e/kWh). While the model is relatively compact, reinforcement learning remains computationally intensive, and further work is needed to evaluate the scalability and energy efficiency of this approach at larger scales or across multiple domains.

### 4.4. Comparative baselines

We compare our agentic RL model to three baselines:

**B1: Direct QA (Zero-Shot CoT)** The base model answers from its parametric knowledge only, prompted with two chain-of-thought exemplars.

**B2: One-Shot SPARQL** A single-turn prompt instructs the model to emit a full SPARQL query; decoding uses temperature 0.2 and top-p 0.95.

**B3: Prompt-Guided Iterative Agent** Our think-query loop without RL; identical prompt as the RL-tuned agent and greedy decoding.

### 4.5. Evaluation protocol and metrics

We evaluate KG-based agents using three key end-to-end metrics: answer accuracy, query executability, and interaction length.

**Accuracy** Correctness is determined by a frozen LLM-based evaluator (`GPT-4o-nano`) shared across all systems, including Direct QA. The evaluator receives the question, gold scalar binding, and the model's `<answer>` response, and returns a Boolean verdict along with a justification. This allows for semantically equivalent but non-identical answers—e.g., paraphrasing, formatting differences, or unit conversions—to be marked as correct, unlike exact string matching.

**Executability rate** The proportion of all SPARQL queries generated by a system that are syntactically valid and execute successfully against the KG—computed as total successful executions divided by the total number of queries issued across the test set.

**Average turns** The mean number of agent interaction steps per question. While not a performance metric, it serves as a diagnostic indicator of the agent's reasoning depth and adaptivity.

## 5. Results

### 5.1. Quantitative performance

Our primary experiment evaluates the **RL-Tuned Iterative Agent** against increasingly capable baselines. As shown in Table 3, performance steadily improves with greater interactivity. Relying on parametric knowledge (**B1**) yields 16.3% accuracy. A single SPARQL query (**B2**) improves this to 19.7%, though hampered by a low 47.7% executability rate. The prompt-guided iterative agent (**B3**) demonstrates the value of a refinement loop, reaching 32.2% accuracy.

Our **RL-Tuned Agent** marks a transformative leap, achieving a final accuracy of **49.7%**—an absolute improvement of 17.5 percentage points over the strongest baseline. This gain is driven by a learned policy for interaction, evidenced by the executability rate soaring to 81.0%. The improvement is statistically significant, confirmed by McNemar's test on the discordant pairs ($n_{\text{RL-correct, baseline-wrong}} = 354$ vs. $n_{\text{RL-wrong, baseline-correct}} = 130$), yielding $\chi^2(1) = 102.75$, $p \ll .001$.

**Table 3**
End-to-end performance on the curated LC-QuAD 2.0 test set (N=1,279). Our RL-Tuned agent outperforms all zero-shot baselines, demonstrating the effectiveness of learning from interaction.

| Model / Approach | Exec. Rate (%) | Acc. (%) | Pass@5 (%) |
|---|---|---|---|
| *Parametric Baseline (No KG Interaction)* | | | |
| B1: Direct QA (CoT) | - | 16.3 | 35.1 |
| *SPARQL-based Baselines (Zero-Shot)* | | | |
| B2: One-Shot SPARQL | 47.7 | 19.7 | 47.2 |
| B3: Prompt-Guided Agent | 54.7 | 32.2 | 61.7 |
| *Our Method (RL Fine-Tuned)* | | | |
| **RL-Tuned Agent** | **81.0** | **49.7** | **77.7** |

## 5.2. Ablation: deconstructing agent performance

To isolate the sources of this gain, we trained a purely **Reactive** agent (no `<think>` block) with the same RL process. Table 4 shows this agent still achieved 48.1% accuracy, confirming that outcome-driven RL is the primary engine of performance, capable of learning effective strategies from interaction alone. However, our main **Deliberative** agent performed best (49.7%), suggesting the `<think>` block acts as a powerful cognitive scaffold. By prompting the model to externalize its plan, the structure regularizes the learning process, leading to a more precise final policy.

**Table 4**
Performance of Deliberative vs. Reactive agents. Both were trained for one epoch with GRPO.

| Model / Approach | Exec. Rate (%) | Accuracy (%) |
|---|---|---|
| Agent-RL (Reactive) | 82.3 | 48.1 |
| **Agent-RL (Deliberative)** | **81.0** | **49.7** |

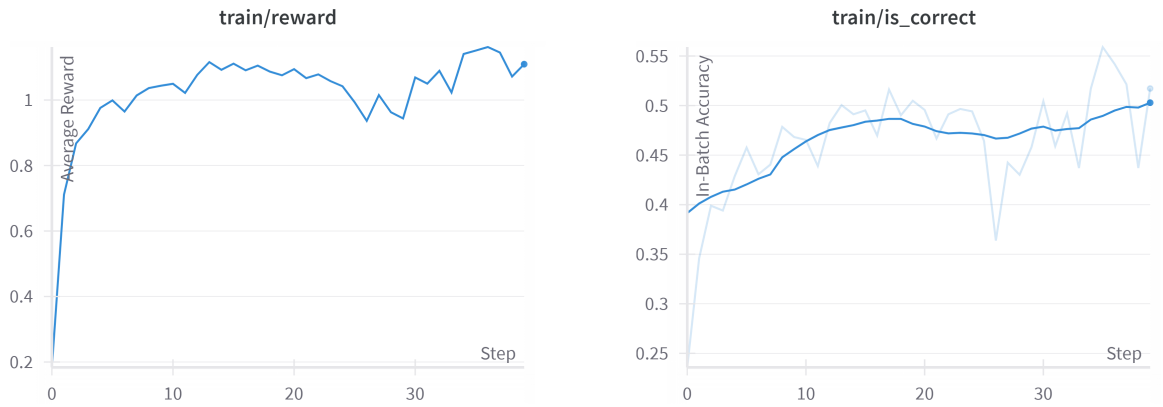
## 5.3. Analysis of learning dynamics

Figure 3 shows the agent learns its policy in layers. Optimizing for reward (a) first drives mastery of SPARQL syntax, as the executability rate (c) rapidly saturates. This foundational skill then enables the agent to improve its semantic reasoning, reflected in the steady rise of in-batch accuracy (b). Crucially, this improved performance is matched by greater efficiency; the average number of turns required per question trends downward (d). The agent learns to be more direct and effective, not just to succeed through brute-force trial and error.

## 5.4. Qualitative and error analysis

### 5.4.1. Error analysis: a shift to higher-quality failures.

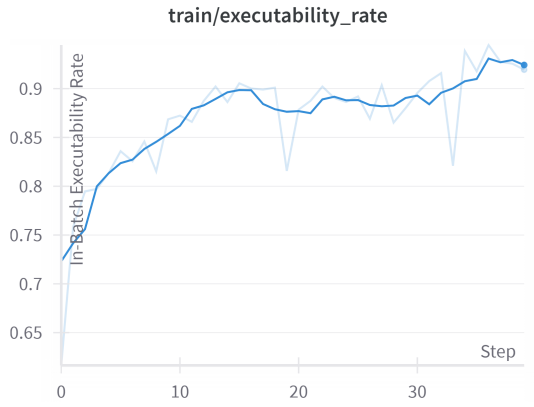
Reinforcement learning induces a crucial shift from syntactic incompetence to semantic reasoning errors (Figure 4). The zero-shot baselines were plagued by fundamental failures: 57% of the One-Shot agent's failures were due to execution errors or refusing to generate a query at all. In stark contrast, our RL-tuned agent nearly eliminated these issues, with such errors accounting for a negligible fraction of failures. Its primary failure mode became *Incorrect Logic* (72.5% of its own failures). The baselines fail because they cannot "speak SPARQL" correctly; our agent has mastered the tool's language and now fails on the much harder problem of reasoning correctly with it.



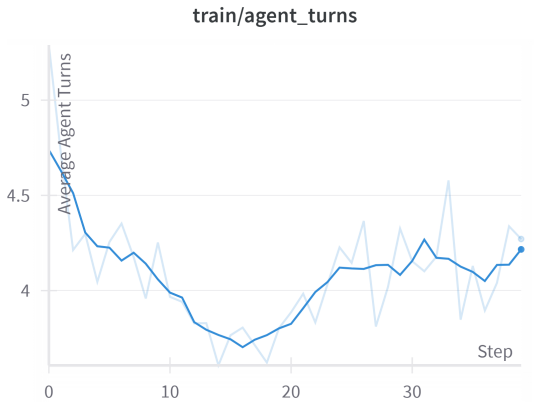

(a) Smoothed Average Reward




(b) In-Batch Accuracy (%)




(c) In-Batch Executability Rate (%)




(d) Average Agent Turns

**Figure 3:** Training dynamics of the RL-Tuned Agent over one epoch (40 training steps). The plots illustrate a layered learning process: the agent is optimized for reward (a), which drives improvements first in query syntax (c) and then in semantic accuracy (b). Simultaneously, the agent learns task efficiency, reducing the average number of turns required to find an answer (d).

#### 5.4.2. Case study: learned resilience and strategic decomposition.

To illustrate the learned policy, we analyzed behavior on the complex question: "Name the Han dynasty capital city with a twin town called Plovdiv.". A direct query fails. Our RL-Tuned agent correctly diagnosed this, pivoted to an exploratory query to find all cities twinned with Plovdiv, and then used a second verification query on the candidates to find the correct answer. This dynamic decomposition, a direct result of RL training, contrasts sharply with the baseline, which became trapped in syntax and logic errors.

## 6. Discussion and Conclusion

This work demonstrates that outcome-driven reinforcement learning (RL) enables compact language models to learn robust, multi-hop reasoning strategies over knowledge graphs. Our agent, trained using GRPO, significantly outperforms static and zero-shot baselines—closing the gap between symbolic structure and neural flexibility. Beyond raw accuracy, we observe emergent behaviors such as adaptive "compute scaling" and strategic query decomposition, showing that the agent learns to allocate effort based on task complexity.

These findings suggest that even small LLMs, when trained with structured feedback, can learn

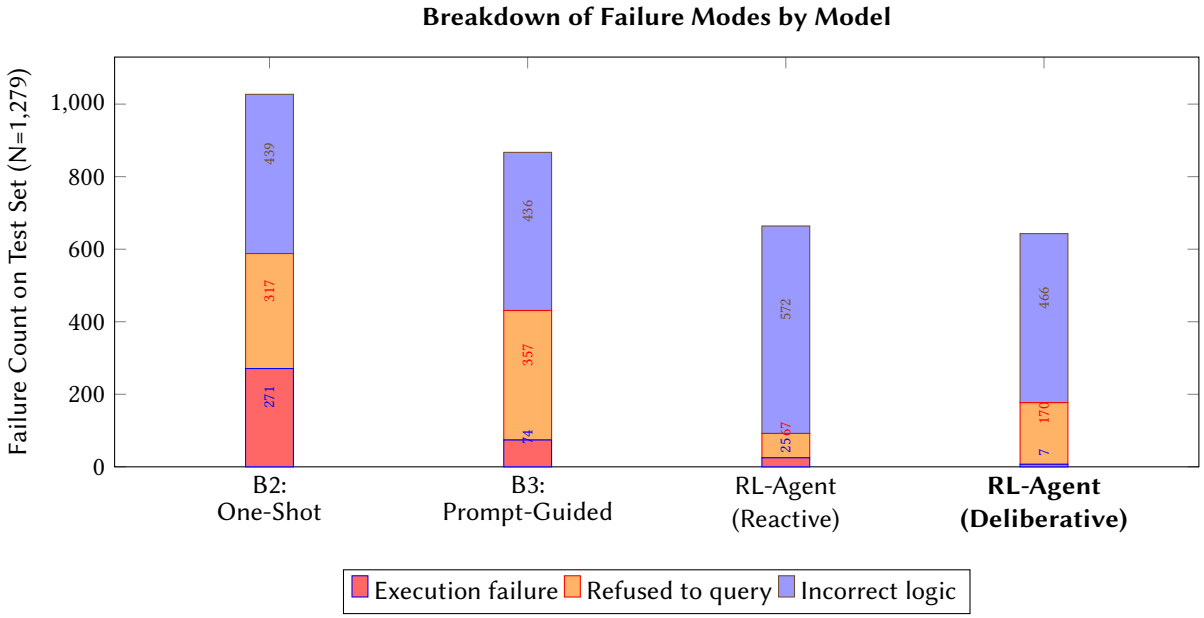



**Figure 4:** Absolute counts of failure modes on the test set. Reinforcement learning dramatically reduces fundamental errors like execution failures and refusal to query. This shifts the primary challenge for the trained agents from generating valid syntax to formulating correct logical plans.

to navigate symbolic environments through interaction. While preliminary, this work highlights a promising direction for combining language models and formal reasoning in a more adaptive and interpretable way.

Our evaluation was limited to a curated subset of LC-QuAD 2.0 with gold entity links and single-answer queries. We did not address open-domain entity linking, incomplete or noisy KGs, or more complex answer types such as lists or aggregations. Additionally, while our approach is lightweight in model size, training with reinforcement learning remains computationally demanding. We conducted experiments on a single H100 GPU, which limits our ability to assess scalability. Questions around energy efficiency, training cost, and feasibility for broader deployment remain open and deserve closer attention in future work.

Future work includes combining supervised fine-tuning with RL to reduce sample complexity; extending to end-to-end KGQA by integrating a learned entity linker; adapting the framework to other structured domains such as NL2SQL; and studying how the agent's policy complexity scales with model size and query difficulty.

## Declaration on Generative AI

During the preparation of this work, the author(s) used GPT-4o in order to: (i) check grammar and spelling, (ii) assist with LaTeX formatting, and (iii) provide sparring and critical feedback on sections. After using this tool, the author(s) reviewed and edited the content as needed and take full responsibility for the publication's content.

## References

[1] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, C. Bizer, DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia, Semantic Web 6 (2015) 167–195. URL: https://journals.sagepub.com/doi/full/10.3233/SW-140134. doi:10.3233/SW-140134.

[2] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Communications of the ACM 57 (2014) 78–85. URL: https://dl.acm.org/doi/10.1145/2629489. doi:10.1145/2629489.

[3] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, A. Zimmermann, Knowledge Graphs, ACM Computing Surveys 54 (2022) 1–37. URL: https://dl.acm.org/doi/10.1145/3447772. doi:10.1145/3447772.

[4] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, J. Taylor, Industry-scale knowledge graphs: lessons and challenges, Communications of the ACM 62 (2019) 36–43. URL: https://dl.acm.org/doi/10.1145/3331166. doi:10.1145/3331166.

[5] Y. Zhang, H. Dai, Z. Kozareva, A. J. Smola, L. Song, Variational Reasoning for Question Answering with Knowledge Graph, 2017. URL: http://arxiv.org/abs/1709.04071. doi:10.48550/arXiv.1709.04071, arXiv:1709.04071 [cs].

[6] H. Sun, T. Bedrax-Weiss, W. Cohen, PullNet: Open Domain Question Answering with Iterative Retrieval on Knowledge Bases and Text, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Association for Computational Linguistics, Hong Kong, China, 2019, pp. 2380–2390. URL: https://www.aclweb.org/anthology/D19-1242. doi:10.18653/v1/D19-1242.

[7] H. Ren, H. Dai, B. Dai, X. Chen, M. Yasunaga, H. Sun, D. Schuurmans, J. Leskovec, D. Zhou, LEGO: Latent Execution-Guided Reasoning for Multi-Hop Question Answering on Knowledge Graphs, in: M. Meila, T. Zhang (Eds.), Proceedings of the 38th International Conference on Machine Learning, volume 139 of *Proceedings of Machine Learning Research*, PMLR, 2021, pp. 8959–8970. URL: https://proceedings.mlr.press/v139/ren21a.html.

[8] B. Y. Lin, X. Chen, J. Chen, X. Ren, KagNet: Knowledge-Aware Graph Networks for Commonsense Reasoning, 2019. URL: http://arxiv.org/abs/1909.02151. doi:10.48550/arXiv.1909.02151, arXiv:1909.02151 [cs].

[9] J. Berant, A. Chou, R. Frostig, P. Liang, Semantic Parsing on Freebase from Question-Answer Pairs, in: D. Yarowsky, T. Baldwin, A. Korhonen, K. Livescu, S. Bethard (Eds.), Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Seattle, Washington, USA, 2013, pp. 1533–1544. URL: https://aclanthology.org/D13-1160/.

[10] H. Sun, B. Dhingra, M. Zaheer, K. Mazaitis, R. Salakhutdinov, W. Cohen, Open Domain Question Answering Using Early Fusion of Knowledge Bases and Text, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Brussels, Belgium, 2018, pp. 4231–4242. URL: http://aclweb.org/anthology/D18-1455. doi:10.18653/v1/D18-1455.

[11] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, O. Yakhnenko, Translating Embeddings for Modeling Multi-relational Data., in: Neural Information Processing Systems, 2013.

[12] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, D. Zhou, Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, 2023. URL: http://arxiv.org/abs/2201.11903. doi:10.48550/arXiv.2201.11903, arXiv:2201.11903 [cs].

[13] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, Y. Cao, ReAct: Synergizing Reasoning and Acting in Language Models, 2023. URL: http://arxiv.org/abs/2210.03629. doi:10.48550/arXiv.2210.03629, arXiv:2210.03629 [cs].

[14] J. Jiang, K. Zhou, Z. Dong, K. Ye, W. X. Zhao, J.-R. Wen, StructGPT: A General Framework for Large Language Model to Reason over Structured Data, 2023. URL: http://arxiv.org/abs/2305.09645. doi:10.48550/arXiv.2305.09645, arXiv:2305.09645 [cs].

[15] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, X. Wu, Unifying Large Language Models and Knowledge Graphs: A Roadmap, IEEE Transactions on Knowledge and Data Engineering 36 (2024) 3580–3599. URL: http://arxiv.org/abs/2306.08302. doi:10.1109/TKDE.2024.3352100, arXiv:2306.08302 [cs].

[16] A. Chakraborty, Multi-hop Question Answering over Knowledge Graphs using Large Language Models, 2024. URL: http://arxiv.org/abs/2404.19234. doi:10.48550/arXiv.2404.19234,

arXiv:2404.19234 [cs].

[17] A. Saxena, A. Tripathi, P. Talukdar, Improving Multi-hop Question Answering over Knowledge Graphs using Knowledge Base Embeddings, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 4498–4507. URL: https://www.aclweb.org/anthology/2020.acl-main.412. doi:10.18653/v1/2020.acl-main.412.

[18] Y. Gu, S. Kase, M. Vanni, B. Sadler, P. Liang, X. Yan, Y. Su, Beyond I.I.D.: Three Levels of Generalization for Question Answering on Knowledge Bases, in: Proceedings of the Web Conference 2021, 2021, pp. 3477–3488. URL: http://arxiv.org/abs/2011.07743. doi:10.1145/3442381.3449992, arXiv:2011.07743 [cs].

[19] W.-t. Yih, M.-W. Chang, X. He, J. Gao, Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, Beijing, China, 2015, pp. 1321–1331. URL: http://aclweb.org/anthology/P15-1128. doi:10.3115/v1/P15-1128.

[20] P. Liang, M. I. Jordan, D. Klein, Learning Dependency-Based Compositional Semantics, 2011. URL: http://arxiv.org/abs/1109.6841. doi:10.48550/arXiv.1109.6841, arXiv:1109.6841 [cs].

[21] W.-t. Yih, M. Richardson, C. Meek, M.-W. Chang, J. Suh, The Value of Semantic Parse Labeling for Knowledge Base Question Answering, in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Association for Computational Linguistics, Berlin, Germany, 2016, pp. 201–206. URL: http://aclweb.org/anthology/P16-2033. doi:10.18653/v1/P16-2033.

[22] E. Karpas, O. Abend, Y. Belinkov, B. Lenz, O. Lieber, N. Ratner, Y. Shoham, H. Bata, Y. Levine, K. Leyton-Brown, D. Muhlgay, N. Rozen, E. Schwartz, G. Shachaf, S. Shalev-Shwartz, A. Shashua, M. Tenenholtz, MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning, 2022. URL: http://arxiv.org/abs/2205.00445. doi:10.48550/arXiv.2205.00445, arXiv:2205.00445 [cs].

[23] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, L. M. Ni, H.-Y. Shum, J. Guo, Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph, 2024. URL: http://arxiv.org/abs/2307.07697. doi:10.48550/arXiv.2307.07697, arXiv:2307.07697 [cs].

[24] J. Singh, R. Magazine, Y. Pandya, A. Nambi, Agentic Reasoning and Tool Integration for LLMs via Reinforcement Learning, 2025. URL: https://www.microsoft.com/en-us/research/publication/agentic-reasoning-and-tool-integration-for-llms-via-reinforcement-learning/.

[25] C. Snell, J. Lee, K. Xu, A. Kumar, Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, 2024. URL: http://arxiv.org/abs/2408.03314. doi:10.48550/arXiv.2408.03314, arXiv:2408.03314 [cs].

[26] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-Consistency Improves Chain of Thought Reasoning in Language Models, 2023. URL: http://arxiv.org/abs/2203.11171. doi:10.48550/arXiv.2203.11171, arXiv:2203.11171 [cs].

[27] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, K. Narasimhan, Tree of Thoughts: Deliberate Problem Solving with Large Language Models, 2023. URL: http://arxiv.org/abs/2305.10601. doi:10.48550/arXiv.2305.10601, arXiv:2305.10601 [cs].

[28] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, D. Guo, DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models, 2024. URL: http://arxiv.org/abs/2402.03300. doi:10.48550/arXiv.2402.03300, arXiv:2402.03300 [cs].

[29] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, K. Cobbe, Let's Verify Step by Step, 2023. URL: http://arxiv.org/abs/2305.20050. doi:10.48550/arXiv.2305.20050, arXiv:2305.20050 [cs].

[30] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, A. Castro-Ros, M. Pellat, K. Robinson, D. Valter, S. Narang, G. Mishra, A. Yu, V. Zhao, Y. Huang, A. Dai, H. Yu, S. Petrov,

E. H. Chi, J. Dean, J. Devlin, A. Roberts, D. Zhou, Q. V. Le, J. Wei, Scaling Instruction-Finetuned Language Models, 2022. URL: http://arxiv.org/abs/2210.11416. doi:10.48550/arXiv.2210.11416, arXiv:2210.11416 [cs].

[31] P. Ma, X. Zhuang, C. Xu, X. Jiang, R. Chen, J. Guo, SQL-R1: Training Natural Language to SQL Reasoning Model By Reinforcement Learning, 2025. URL: http://arxiv.org/abs/2504.08600. doi:10.48550/arXiv.2504.08600, arXiv:2504.08600 [cs].

[32] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, QLoRA: Efficient Finetuning of Quantized LLMs, 2023. URL: http://arxiv.org/abs/2305.14314. doi:10.48550/arXiv.2305.14314, arXiv:2305.14314 [cs].