# KBQA-o1: Agentic Knowledge Base Question Answering with Monte Carlo Tree Search

**Haoran Luo** [1 2] **Haihong E** [1] **Yikai Guo** [3] **Qika Lin** [4] **Xiaobao Wu** [2] **Xinyu Mu** [1] **Wenhao Liu** [1] **Meina Song** [1] **Yifan Zhu** [1] **Luu Anh Tuan** [2]

## Abstract

Knowledge Base Question Answering (KBQA) aims to answer natural language questions with a large-scale structured knowledge base (KB). Despite advancements with large language models (LLMs), KBQA still faces challenges in weak KB awareness, imbalance between effectiveness and efficiency, and high reliance on annotated data. To address these challenges, we propose KBQA-o1, a novel agentic KBQA method with Monte Carlo Tree Search (MCTS). It introduces a ReAct-based agent process for stepwise logical form generation with KB environment exploration. Moreover, it employs MCTS, a heuristic search method driven by policy and reward models, to balance agentic exploration's performance and search space. With heuristic exploration, KBQA-o1 generates high-quality annotations for further improvement by incremental fine-tuning. Experimental results show that KBQA-o1 outperforms previous low-resource KBQA methods with limited annotated data, boosting Llama-3.1-8B model's GrailQA F1 performance to 78.5% compared to 48.5% of the previous sota method with GPT-3.5-turbo. Our code is publicly available at https://github.com/LHRLAB/KBQA-o1.

## 1. Introduction

Knowledge Base Question Answering (KBQA) leverages a large-scale structured knowledge base (KB), such as Freebase (Bollacker et al., 2008) or Wikidata (Vrandečić & Krötzsch, 2014), as a reference to answer questions in natural language, widely applied in fields such as search engines (Jang et al., 2017), medical consultations (Wu et al., 2024), and legal analysis (Cui et al., 2023). Typically, KBs are stored in graph databases and accessed using graph queries, such as SPARQL (Pérez et al., 2009), which support multi-hop and logical queries to acquire knowledge information. To answer natural language questions, language models are usually leveraged (Ye et al., 2022) to convert the questions into logical forms, such as S-expression (Gu et al., 2021), and then transform them into executable graph queries to obtain answers from KB, as shown in Figure 1.

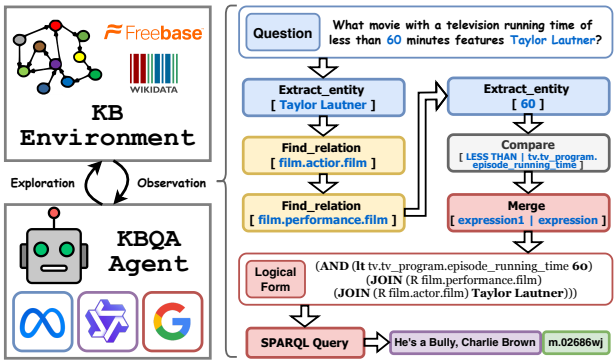



*Figure 1.* An example of KBQA task to answer a natural language question by exploring the KB environment with KBQA agent.

With the emergence of LLMs (OpenAI, 2024; Dubey et al., 2024), two main types of KBQA methods appear, as shown in Figure 2. On the one hand, end-to-end methods (Luo et al., 2024b;a) generate logical forms directly from natural language questions and utilize retrieval before or after generation for improvement. On the other hand, step-by-step methods (Huang et al., 2024; Sun et al., 2024) alternate between generation and retrieval for stepwise thinking on KB, performing in a Chain-of-Thought (CoT) (Wei et al., 2022) or Tree-of-Thoughts (ToT) (Yao et al., 2023a) manner.

However, three main challenges remain: **(1) Poor awareness of the KB environment in end-to-end methods.** Relying on direct logical form generation by language models, end-to-end KBQA methods (Li et al., 2023; Nie et al., 2024) struggle with limited logical form schemas and unseen entities and relations that existed in KB, making it difficult to fully capture the KB environment. **(2) Local optima**

[1]Beijing University of Posts and Telecommunications [2]Nanyang Technological University [3]Beijing Institute of Computer Technology and Application [4]National University of Singapore. Correspondence to: Haihong E <ehaihong@bupt.edu.cn>.

*Proceedings of the $42^{nd}$ International Conference on Machine Learning*, Vancouver, Canada. PMLR 267, 2025. 

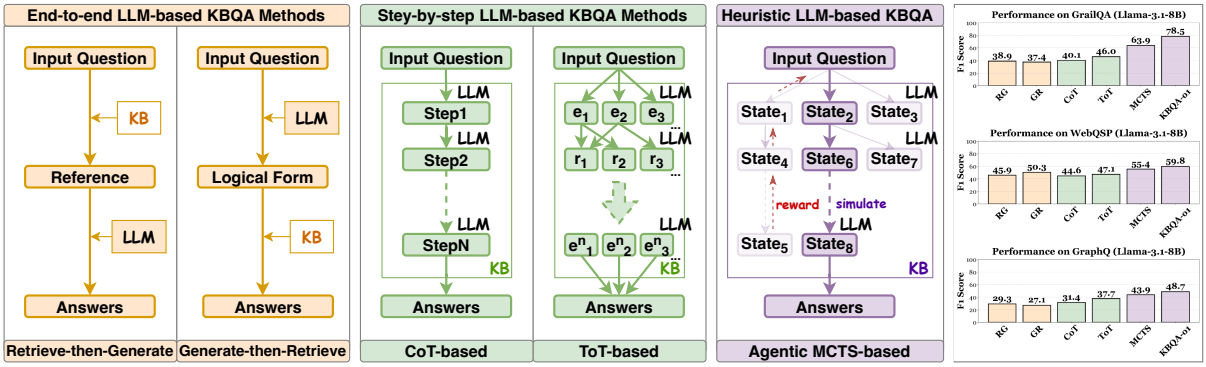



*Figure 2.* Comparison of the previous end-to-end KBQA framework, including retrieve-then-generate (RG) and generate-then-retrieve (GR) methods, step-by-step KBQA methods, including CoT-based and ToT-based methods, and our proposed heuristic KBQA method, which is agentic MCTS-based. With the same Llama-3.1-8B (Dubey et al., 2024) as the base model, both the MCTS-based agent process and the full KBQA-o1 after the incremental fine-tuning show improvements on all three KBQA datasets.

**or large search space in step-by-step methods.** Compared to end-to-end methods, CoT-based step-by-step methods (Huang et al., 2024) in the search process can lead to local optima due to intermediate bias. Although ToT-based methods (Sun et al., 2024) expand search options, tree-like searches still face challenges with large search spaces. **(3) Training depends on high-quality annotation.** Training open-source LLMs (Luo et al., 2024a) significantly contributes to the generation of logical forms for KBQA tasks, but this heavily relies on the quality of the annotated data. For large-scale KBs, producing substantial, high-quality annotated data by human labor is impractical.

To address these challenges, we propose KBQA-o1, a novel agentic KBQA approach with heuristic exploration of the KB environment, as shown in Figure 1. First, we design a ReAct-based (Yao et al., 2023b) agent process with atomic query tools to generate logical forms by fully interacting with the KB environment. Moreover, we employ Monte Carlo Tree Search (MCTS) (Świechowski et al., 2022), driven by policy and reward models, to optimize the agent process from local optima or large search space. Furthermore, to mitigate reliance on extensive human annotation, we first fine-tune the policy and reward models only using a small amount of sample-labeled data. For unlabeled questions, we employ MCTS exploration with the reward model filter instead of human labor to generate abundant, auto-annotated data, further enhancing the capabilities of both the policy and reward models by incremental fine-tuning.

We perform experiments on three KBQA datasets, GrailQA (Gu et al., 2021), WebQSP (Yih et al., 2016) and GraphQ (Su et al., 2016) in low-resource settings (Li et al., 2023) for application with limited annotated data. Experimental results demonstrate that KBQA-o1 outperforms existing low-resource KBQA methods and even approaches or surpasses the performance of fully supervised KBQA models, especially in more difficult cases like compositional and zero-shot. Ablation studies further validate the proposed MCTS-based agent process and incremental fine-tuning, both of which make KBQA-o1 outperform other forms of KBQA methods, as shown in Figure 2. In addition, KBQA-o1 supports multiple open-source LLMs, including Llama-3 (Dubey et al., 2024), Qwen2.5 (Yang et al., 2025) and Gemma-2 (Team et al., 2024), making it a plug-and-play and promising solution for diverse KBQA applications.

## 2. Related Work

**Knowledge Base Question Answering.** Before the rise of LLMs, KBQA methods could be divided into information-retrieval-based (IR-based) (Sun et al., 2018; 2019; Zhang et al., 2022) and semantic-parsing-based (SP-based) (Ye et al., 2022; Shu et al., 2022). In the era of LLMs, LLM-based KBQA methods can be divided into two categories: end-to-end and step-by-step. End-to-end methods take advantage of in-context learning (ICL) (Li et al., 2023; Nie et al., 2024) or fine-tuning (Luo et al., 2024b;a) to enable LLMs to generate queries. The step-by-step methods (Gu et al., 2023; Sun et al., 2024; Huang et al., 2024) follow a reasoning process on the graph to gradually find the answers. In this paper, we propose the first heuristic KBQA method.

**LLMs and LLM-powered Agent.** LLMs have shown significant advantages in generation and reasoning (Wang et al., 2024). Powered by CoT(Wei et al., 2022; Li et al., 2024), guiding LLMs to think step by step can further enhance reasoning capabilities. ReAct (Yao et al., 2023b) introduces a prompt-based agent, using tools to interact with the environment. Lightman et al. (2024) verifies CoT by rewarding the process and outcome. On the other hand, to expand the thought space in every step, ToT (Yao et al., 2023a) reasons in a tree-like manner. RAP (Hao et al., 2023) employs MCTS, a heuristic algorithm applied in AlphaGo.

*Table 1.* Description of proposed atomic query tools and corresponding arguments, where $\mathcal{M}_o = \{max, min\}$, and $\mathcal{M}_c = \{<, \leq, >, \geq\}$ are mode sets of Order and Compare. By using tools, the agent can obtain the target functions and equivalent logical forms accordingly.

| Atomic Query Tool | Arguments | Target Function | Equivalent Logical Form |
|---|---|---|---|
| `Extract_entity` [ *entity* ] | *entity* $\in \mathcal{E}$ | *expression* = START('*entity*') | *entity* |
| `Find_relation` [ *relation* ] | *relaiton* $\in \mathcal{R}$ | *expression* = JOIN('*relation*', *expression*) | (JOIN *relation* (*expression*)) |
| `Merge` [ *expression1* \| *expression* ] | *expression1*, *expression* | *expression* = AND(*expression1*, *expression*) | (AND (*expression1*) (*expression*)) |
| `Order` [ *mode* \| *relation* ] | *mode* $\in \mathcal{M}_o$, *relation* $\in \mathcal{R}$ | *expression* = ARG('*mode*', *expression*, '*relation*') | (*mode* (*expression*) *relation*) |
| `Compare` [ *mode* \| *relation* ] | *mode* $\in \mathcal{M}_c$, *relation* $\in \mathcal{R}$ | *expression* = CMP('*mode*', '*relation*', *expression*) | (*mode relation* (*expression*)) |
| `Time_constraint` [ *relation* \| *time* ] | *relation* $\in \mathcal{R}$, *time* $\in \mathcal{E}$ | *expression* = TC(*expression*, '*relation*', '*time*') | (TC (*expression*) *relation time*) |
| `Count` [ *expression* ] | *expression* | *expression* = COUNT(*expression*) | (COUNT (*expression*)) |
| `Finish` [ *expression* ] | *expression* | *expression* = STOP(*expression*) | (*expression*) |

## 3. Preliminaries

**Definition 1: Knowledge Base.** A knowledge base (KB) is a large-scale knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$, composed of an entity set $\mathcal{E}$, a relation set $\mathcal{R}$, and a triple set $\mathcal{T}$. Relations in the relation set $r \in \mathcal{R}$ are used to connect two entities. Each triple $(s, r, o) \in \mathcal{T}$ is formed by (entity, relation, entity), thus $\mathcal{T} = \{(s, r, o) | s \in \mathcal{E}, r \in \mathcal{R}, o \in \mathcal{E}\}$.

**Definition 2: Logical Form.** The logical form $F$ is a multi-hop expression that can convert equally to a graph query $q = \text{Convert}(F)$. Each logical form can be divided into a stepwise list of functions $F = [f_i]_{i=1}^{l}$ with $l$ steps.

**Problem Statement.** In the KBQA task, given a natural language question $\mathcal{Q}$ and a KB $\mathcal{G}$, the goal is to first convert $\mathcal{Q}$ into a logical form $F$, and then an executable graph query $q$. Once executed, the result $\mathcal{A} = \text{Exec}(q, \mathcal{G})$ is a set of entities in the KB $\mathcal{A} \subset \mathcal{E}$ that answer the question $\mathcal{Q}$.

## 4. Method: KBQA-o1

In this section, we introduce the three components of KBQA-o1: agent initialization, heuristic environment exploration using MCTS with policy and reward models to optimize the agent process, and incremental fine-tuning with auto-annotated data to improve low-resource performance.

### 4.1. Agent Initialization

KBQA-o1 follows a ReAct-based (Yao et al., 2023b) agent prompt, with KB environment, the agent state space and exploration space, targeting to generate logical forms.

**KB Environment $\mathcal{G}$.** We consider KB a critical environment that provides guides in generating the logical form at each step of the agent, for example, by providing the candidate relations connected to the current state of the logical form.

**The Agent State Space $\mathcal{H}$.** The agent state $\mathbf{h}_t \in \mathcal{H}$ is defined by its exploration history as $\mathbf{h}_t = (\mathbf{h}_0, \mathbf{e}_1, \ldots, \mathbf{e}_t)$.

- **Initial State ($\mathbf{h}_0$):** We create an initial prompt, consisting of the task description and the given question $\mathcal{Q}$, as the initial state $\mathbf{h}_0$, as shown in Appendix A.1.
- **State Update ($\mathbf{h}_t$):** At each step $t$, the state incorporates the latest exploration step, which is prompted as ReAct-based (Yao et al., 2023b) Thought-Action-Observation tuple $\mathbf{e}_t = (\mathbf{e}_t^{\text{tht}}, \mathbf{e}_t^{\text{act}}, \mathbf{e}_t^{\text{obs}})$ as shown in Appendix A.2, to update: $\mathbf{h}_t = \mathbf{h}_{t-1} + \mathbf{e}_t$.
- **State Representation ($F_{\mathbf{h}_t}$):** Besides prompt-based history, the cumulative Observations in the trajectory $(\mathbf{e}_1^{\text{obs}}, \ldots, \mathbf{e}_t^{\text{obs}})$ determine the function list, which is equivalent to the logical form $F_{\mathbf{h}_t} = [\mathbf{e}_i^{\text{obs}}]_{i=1}^{t}$.

**The Agent Exploration Space $\text{Exp}(\mathcal{H}, \mathcal{G})$.** The agent exploration $\mathbf{e}_t \in \text{Exp}(\mathbf{h}_{t-1}, \mathcal{G})$ is dynamically determined by the last state $\mathbf{h}_{t-1} \in \mathcal{H}$ and the KB environment $\mathcal{G}$. Each exploration comprises the following components:

- **Tool Selection ($\mathbf{e}_t^{\text{tht}}$):** Based on $\mathbf{h}_{t-1}$, the agent selects one of the eight atomic query tools as shown in Table 1.
- **Argument Determination ($\mathbf{e}_t^{\text{act}}$):** The agent identifies the appropriate arguments for the selected tool, leveraging the candidates provided by the KB environment. For example:
  - Calling `Extract_entity` requires specifying an *entity* name existed in $\mathcal{G}$ as the argument.
  - Calling `Find_relation` involves specifying a *relation* name that has a connection with $F_{\mathbf{h}_{t-1}}$.
- **Function Writen ($\mathbf{e}_t^{\text{obs}}$):** Based on the selected tool and arguments, the agent writes down the corresponding target function in Observation, referred to Table 1.

**The Agent Target $(\mathbf{h}_l, F_{\mathbf{h}_l}, \mathcal{A}_{\mathbf{h}_l})$.** The agent explores the exploration space until calling the `Finish` tool or when the length of the function list exceeds the maximum allowable length $l < L$. The ultimate target of the agent is to find a complete state $\mathbf{h}_l$, which forms a logical form $F_{\mathbf{h}_l}$ and the final answers executed $\mathcal{A}_{\mathbf{h}_l} = \text{Exec}(\text{Convert}(F_{\mathbf{h}_l}), \mathcal{G})$.

**Proposition 4.1.** *The agent's awareness of the environment makes it more effective in generating optimal logical forms compared to end-to-end methods.*

*Proof.* We provide quantitative experimental results in Section 5.4 and qualitative proofs in Appendix B.1. □

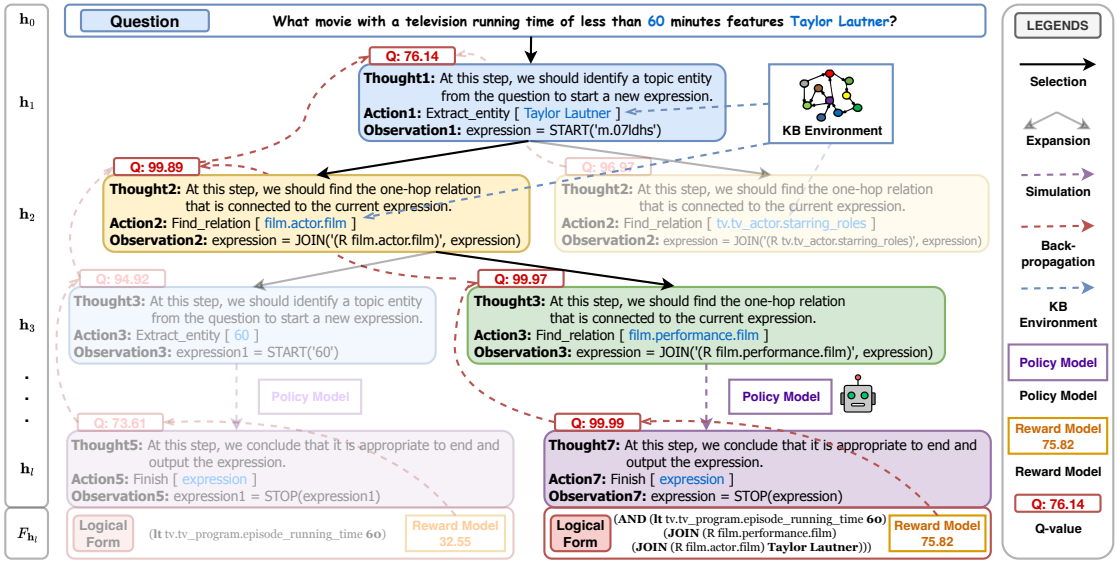



*Figure 3.* An example of the heuristic KB environment exploration with MCTS driven by policy and reward models.

### 4.2. Heuristic Environment Exploration

As shown in Figure 3, to address the issue of the step-by-step agent falling into local optima or large search spaces, we design an MCTS-based heuristic environment exploration method, driven by a policy model and a reward model.

#### 4.2.1. The Policy Model

The policy model aims to provide the agent with a forward-looking capability. We use the last state at each step $\mathbf{h}_{t-1}$ from the annotated training set $\mathcal{D}_a$ as input, and the steps from the current state to the conclusion $(\mathbf{e}_t, ..., \mathbf{e}_l)$ as output, forming SFT data for training the policy model $\pi_{\text{policy}}$:

$$\mathcal{L}_{\text{SFT}}(\pi_{\text{policy}}, \mathcal{D}_a) = -\mathbb{E}_{\mathcal{D}_a}\left[\sum_{t=1}^{l} \log \pi_{\text{policy}}\left(\sum_{i=t}^{l} \mathbf{e}_i \mid \mathbf{h}_{t-1}\right)\right]. \tag{1}$$

#### 4.2.2. The Reward Model

The reward model aims to assess the entire trajectory by scoring the final logical form. We use the question $\mathcal{Q}$ as input, and the logical form $F_{\mathbf{h}_l}$ from the annotated training set $\mathcal{D}_a$ as output, forming SFT data for training the reward model $\pi_{\text{reward}}$:

$$\mathcal{L}_{\text{SFT}}(\pi_{\text{reward}}, \mathcal{D}_a) = -\mathbb{E}_{\mathcal{D}_a}\left[\log \pi_{\text{reward}}\left(F_{\mathbf{h}_l} \mid \mathcal{Q}\right)\right], \tag{2}$$

Moreover, we employ a scoring method $R_\pi$ that uses the logits from the LLM $\pi$, which can be $\pi_{\text{policy}}$ or $\pi_{\text{reward}}$, to evaluate the likelihood of an output $y$, given an input $x$:

$$R_\pi(y \mid x) = \beta + \alpha \log \pi\left(y \mid x\right). \tag{3}$$

where $\beta$ is the defined full score, set as 100, and $\alpha$ is a positive temperature to control the disparity of scores.

#### 4.2.3. Monte Carlo Tree Search over KB

MCTS is a heuristic search algorithm in the form of a tree, where each node represents a state in the agent process. Starting from the initial state (the root node), the algorithm uses four stages of selection, expansion, simulation, and back-propagation to explore and enrich the search tree iteratively. MCTS conducts a total of $N$ search rollouts. In the $n$-th rollout ($n = 1, ..., N$), the agent process can be represented by the trajectory of agent states $\{[\mathbf{h}_t^{(n)}]_{t=1}^{l}\}_{n=1}^{N}$.

**Selection.** When a new MCTS rollout begins, the agent process starts from the root node and progressively searches down through the child nodes of the already explored tree until it reaches a leaf node. At each level, the UCT (Upper Confidence Bound applied to Trees) (Świechowski et al., 2022) algorithm is used to select the next child node:

$$\mathbf{e}_t \leftarrow \arg\max_{\mathbf{e} \in E(\mathbf{h}_{t-1}^{(n)})}\left[Q(\mathbf{h}_{t-1}^{(n)} + \mathbf{e}) + w\sqrt{\frac{\ln N(\mathbf{h}_{t-1}^{(n)})}{N(\mathbf{h}_{t-1}^{(n)} + \mathbf{e})}}\right], \tag{4}$$

where $N(.)$ is the visit counts of the agent state during the MCTS process, $E(.)$ is the candidate expansion of Thought-Action-Observation explorations, derived from Equation (7), and $Q(.)$ is the Q-value of the agent state, which will be updated by back-propagation. UCT balances the selection of high-scoring nodes with the exploration of unvisited ones. The variable $w$ controls the tendency towards exploration. A larger $w$ encourages exploration of nodes with fewer visits, while a smaller $w$ biases nodes with higher scores.

**Expansion.** Once the selection process reaches a leaf node but not in a terminal `Finish` state and is not beyond the maximum depth $L$, the policy model $\pi_{\text{policy}}$ generates

several possible next states by beam search:

$$\{\mathbf{e}_t^{(b)}\}_{b=1}^{B} \sim \pi_{\text{policy}}\left(\mathbf{e}_t \mid \mathbf{h}_{t-1}^{(n)}\right)_{\text{beam}}, \tag{5}$$

where $B$ is the beam size. To engage with KB environment, we utilize an unsupervised retrieval model, SimCSE (Gao et al., 2021), to match the generated explorations with the exploration options $\mathbf{e} \in \text{Exp}(\mathbf{h}_{t-1}^{(n)}, \mathcal{G})$ that are executable over KB $\mathcal{G}$ when connected with the last state $\mathbf{h}_{t-1}^{(n)}$:

$$\{\mathbf{e}_t^{(i)}\}_{i=1}^{k} \leftarrow \arg\max^{k}_{\mathbf{e}\in\text{Exp}(\mathbf{h}_{t-1}^{(n)},\mathcal{G})} \text{SimCSE}\left(\{\mathbf{e}_t^{(b)}\}_{b=1}^{B}, \mathbf{e}\right). \tag{6}$$

For example, if the ground truth is '`Find_relation` [ file.actor.film ]' the model might initially generate '[ film.actor ]'. Then, we select the most semantically related action options from $\text{Exp}(\mathbf{h}_{t-1}^{(n)}, \mathcal{G})$ such as '[ file.actor.film ]' and '[ tv.tv_actor.starring_roles ]' to filter the top $k$ explorations $\{\mathbf{e}_t^{(i)}\}_{i=1}^{k}$.

Then, the policy model $\pi_{\text{policy}}$ scores the $k$ candidates based on the previous state $\mathbf{h}_{t-1}^{(n)}$ and selects the top $d$ candidates as expanded options $E(\mathbf{h}_{t-1}^{(n)})$, which are added as child nodes to the leaf node, thus expanding the tree:

$$E(\mathbf{h}_{t-1}^{(n)}) = \{\mathbf{e}_t^{(i)}\}_{i=1}^{d} \leftarrow \arg\max^{d} R_{\pi_{\text{policy}}}\left(\{\mathbf{e}_t^{(i)}\}_{i=1}^{k} \mid \mathbf{h}_{t-1}^{(n)}\right). \tag{7}$$

**Simulation.** After the nodes are expanded, the policy model assigns scores to all newly added child nodes. The node with the highest prospective score is selected:

$$\mathbf{e}_t \leftarrow \arg\max_{\mathbf{e}\in E(\mathbf{h}_{t-1}^{(n)})} R_{\pi_{\text{policy}}}\left(\mathbf{e} \mid \mathbf{h}_{t-1}^{(n)}\right), \tag{8}$$

and the simulation continues to explore the process until the final `Finish` state, producing a complete logical form generation trajectory.

**Back-propagation.** Once the final state is reached, the reward model $\pi_{\text{reward}}$ evaluates the entire trajectory by assessing the corresponding logical form combined with the policy model's score from the last step to compute the overall Q-value of the final state:

$$Q(\mathbf{h}_l^{(n)}) \leftarrow \delta\, R_{\pi_{\text{policy}}}\left(\mathbf{e}_l \mid \mathbf{h}_{l-1}^{(n)}\right) + (1-\delta)\, R_{\pi_{\text{reward}}}\left(F_{\mathbf{h}_l^{(n)}} \mid \mathcal{Q}\right), \tag{9}$$

where $\delta$ is a ratio from $(0, 1)$ to balance the process score and overall score. The algorithm then back-propagates the score by updating the Q-values of all nodes along the trajectory, from the leaf back to the root:

$$Q(\mathbf{h}_t^{(n)}) \leftarrow \max_{j=1}^{n}\left(\frac{\sum_{i=l}^{t} Q(\mathbf{h}_i^{(j)})}{l-t+1}\right), \tag{10}$$

where the Q-values of parent nodes are updated to the maximum average Q-value from all child nodes along the trajectory. Meanwhile, the visit count of each node along the trajectory $N(\mathbf{h}_t^{(n)})$ adds 1, and then the next rollout begins.

#### 4.2.4. Final Trajectory Chosen

After MCTS completes its exploration for $N$ rollouts with parameter set $\theta$, we select the trajectory $\hat{\mathbf{h}}_l^{\mathcal{Q}} \in \{\mathbf{h}_l^{(n)}\}_{n=1}^{N}$ with the highest Q-value in every state in the expanded search tree as the optimal trajectory of question $\mathcal{Q}$:

$$(\hat{\mathbf{h}}_l^{\mathcal{Q}}, \hat{F}^{\mathcal{Q}}, \hat{\mathcal{A}}^{\mathcal{Q}}) = \text{MCTS}_{\theta}\left(\mathcal{Q}, \pi_{\text{policy}}, \pi_{\text{reward}}\right), \tag{11}$$

where $\hat{F}^{\mathcal{Q}}$ is corresponding logical form of $\hat{\mathbf{h}}_l^{\mathcal{Q}}$, and $\hat{\mathcal{A}}^{\mathcal{Q}} = \text{Exec}(\text{Convert}(\hat{F}^{\mathcal{Q}}), \mathcal{G})$ is the executed answers.

**Proposition 4.2.** *The MCTS-based heuristic method balances the effectiveness and size of the search space better than CoT-based and ToT-based step-by-step methods.*

*Proof.* We provide quantitative experimental results in Section 5.4 and qualitative proofs in Appendix B.2. □

### 4.3. Incremental Fine-Tuning

In addition to training on annotated data, KBQA-o1 employs MCTS with exploration incentives $\theta_{\text{exp}}$ for heuristic exploration on unannotated questions $\mathcal{Q} \in \mathcal{D}_n$:

$$\{(\hat{\mathbf{h}}_l^{\mathcal{Q}}, \hat{F}^{\mathcal{Q}}, \hat{\mathcal{A}}^{\mathcal{Q}})\}_{\mathcal{Q}\in\mathcal{D}_n} = \left\{\text{MCTS}_{\theta_{\text{exp}}}\left(\mathcal{Q}, \pi_{\text{policy}}, \pi_{\text{reward}}\right)\right\}_{\mathcal{Q}\in\mathcal{D}_n}. \tag{12}$$

Then, we discard the annotation by choosing if the answer set is not empty and the reward score of logical form does not exceed a threshold $\gamma^*$:

$$\hat{R}^{\mathcal{Q}} = R_{\pi_{\text{reward}}}\left(\hat{F}^{\mathcal{Q}} \mid \mathcal{Q}\right), \tag{13}$$

$$\mathcal{D}_i = \mathcal{D}_a \cup \left\{\left(\mathcal{Q}, \hat{F}^{\mathcal{Q}}, \hat{\mathcal{A}}^{\mathcal{Q}}\right) | \hat{\mathcal{A}}^{\mathcal{Q}} \neq \emptyset \wedge \hat{R}^{\mathcal{Q}} > \gamma^*\right\}_{\mathcal{Q}\in\mathcal{D}_n}. \tag{14}$$

Combined with the original annotated data $\mathcal{D}_a$, the incremental data $\mathcal{D}_i$ is then used for incremental fine-tuning of the policy and reward models:

$$\mathcal{L}_{\text{SFT}}(\pi_{\text{policy}}, \mathcal{D}_i) = -\mathbb{E}_{\mathcal{D}_i}\left[\sum_{t=1}^{l} \log \pi_{\text{policy}}\left(\sum_{i=t}^{l} \mathbf{e}_i \mid \mathbf{h}_{t-1}\right)\right], \tag{15}$$

$$\mathcal{L}_{\text{SFT}}(\pi_{\text{reward}}, \mathcal{D}_i) = -\mathbb{E}_{\mathcal{D}_i}\left[\log \pi_{\text{reward}}\left(F_{\mathbf{h}_l} \mid \mathcal{Q}\right)\right]. \tag{16}$$

Through incremental fine-tuning, the policy and reward models gain enhanced understanding of the environment and a preference for high-reward logical form trajectories. Finally, we perform testing $\mathcal{D}_t$ under efficiency-focused MCTS parameter settings $\theta_{\text{eff}}$, yielding final answers $\hat{\mathcal{A}}^{\mathcal{Q}}$:

$$\{(\hat{\mathbf{h}}_l^{\mathcal{Q}}, \hat{F}^{\mathcal{Q}}, \hat{\mathcal{A}}^{\mathcal{Q}})\}_{\mathcal{Q}\in\mathcal{D}_t} = \left\{\text{MCTS}_{\theta_{\text{eff}}}\left(\mathcal{Q}, \pi_{\text{policy}}, \pi_{\text{reward}}\right)\right\}_{\mathcal{Q}\in\mathcal{D}_t}. \tag{17}$$

**Proposition 4.3.** *There exists a reward threshold $\gamma^* < \beta$ such that incremental fine-tuning data, under the effect of the KB, can improve model performance.*

*Proof.* We provide quantitative experimental results in Section 5.5 and qualitative proofs in Appendix B.3. □

*Table 2.* 40-shot results on the local dev set of GrailQA. **Bold** numbers indicate the best low-resource performance.

| Method | LLM | I.I.D | | Compositional | | Zero-shot | | Overall | |
|---|---|---|---|---|---|---|---|---|---|
| | | EM | F1 | EM | F1 | EM | F1 | EM | F1 |
| *Full-Supervised KBQA Methods* | | | | | | | | | |
| RnG-KBQA (Ye et al., 2022) | | 86.7 | 89.0 | 61.7 | 68.9 | 68.8 | 74.7 | 69.5 | 76.9 |
| DecAF (Yu et al., 2023) | | 88.7 | 92.4 | 71.5 | 79.8 | 65.9 | 77.3 | 72.5 | 81.4 |
| TIARA (Shu et al., 2022) | | 88.4 | 91.2 | 66.4 | 74.8 | 73.3 | 80.7 | 75.3 | 81.9 |
| *Low-resource KBQA Methods* | | | | | | | | | |
| KB-BINDER (Li et al., 2023) | GPT-3.5-turbo | 40.0 | 43.3 | 33.9 | 36.6 | 40.1 | 44.0 | 38.7 | 42.2 |
| KB-Coder (Nie et al., 2024) | GPT-3.5-turbo | 40.6 | 45.5 | 34.5 | 38.6 | 42.2 | 47.3 | 40.1 | 44.9 |
| ARG-KBQA (Tian et al., 2024) | GPT-3.5-turbo | 46.6 | 51.5 | 36.4 | 41.8 | 46.6 | 52.1 | 43.8 | 48.5 |
| KBQA-o1 (Llama-3) | Llama-3.1-8B | 77.8 ±0.5 | 85.5 ±0.4 | 76.3 ±0.6 | 77.6 ±0.5 | 68.1 ±0.8 | 76.1 ±0.4 | 71.9 ±0.3 | 78.5 ±1.0 |
| | Llama-3.3-70B | **79.2** ±0.7 | **88.2** ±0.3 | 80.8 ±0.6 | 82.3 ±0.2 | 71.8 ±1.0 | 80.3 ±0.8 | 75.2 ±0.7 | 81.6 ±0.5 |
| KBQA-o1 (Qwen2.5) | Qwen2.5-7B | 76.1 ±0.3 | 84.4 ±0.5 | 75.7 ±0.7 | 77.0 ±0.7 | 67.3 ±0.9 | 75.7 ±0.2 | 70.8 ±0.5 | 77.9 ±0.8 |
| | Qwen2.5-14B | 78.0 ±0.4 | 85.9 ±0.6 | 77.4 ±0.8 | 78.4 ±0.5 | 68.6 ±0.1 | 79.0 ±0.7 | 72.5 ±0.6 | 79.2 ±0.2 |
| | Qwen2.5-32B | 78.9 ±0.5 | 86.2 ±0.7 | 79.3 ±0.6 | 80.4 ±0.8 | 69.6 ±0.3 | 79.7 ±0.7 | 73.3 ±0.8 | 80.3 ±1.3 |
| | Qwen2.5-72B | 79.1 ±0.4 | 87.4 ±0.5 | **81.5** ±0.7 | **83.0** ±0.6 | **72.1** ±0.4 | **81.9** ±0.5 | **75.8** ±0.4 | **82.1** ±0.2 |
| KBQA-o1 (Gemma-2) | Gemma-2-9B | 77.1 ±0.5 | 82.3 ±0.6 | 75.6 ±0.8 | 76.7 ±0.4 | 66.3 ±0.2 | 76.4 ±0.6 | 70.6 ±0.7 | 77.8 ±0.5 |
| | Gemma-2-27B | 78.3 ±0.3 | 82.6 ±0.4 | 76.0 ±0.6 | 77.3 ±0.5 | 69.5 ±1.1 | 79.6 ±0.7 | 72.8 ±0.5 | 79.7 ±0.3 |

*Table 3.* 100-shot results on the test set of WebQSP. **Bold** numbers indicate the best low-resource performance.

| Method | LLM | F1 |
|---|---|---|
| *Full-Supervised KBQA Methods* | | |
| RnG-KBQA (Ye et al., 2022) | | 75.6 |
| DecAF (Yu et al., 2023) | | 76.7 |
| TIARA (Shu et al., 2022) | | 78.7 |
| *Low-resource KBQA Methods* | | |
| KB-BINDER (Li et al., 2023) | GPT-3.5-turbo | 52.6 |
| KB-Coder (Nie et al., 2024) | GPT-3.5-turbo | 55.7 |
| ARG-KBQA (Tian et al., 2024) | GPT-3.5-turbo | 58.8 |
| KBQA-o1 (Llama-3) | Llama-3.1-8B | 59.8 ±1.2 |
| | Llama-3.3-70B | **67.0** ±0.4 |
| KBQA-o1 (Qwen2.5) | Qwen2.5-7B | 57.8 ±0.7 |
| | Qwen2.5-14B | 60.1 ±1.3 |
| | Qwen2.5-32B | 63.7 ±0.9 |
| | Qwen2.5-72B | 66.5 ±1.1 |
| KBQA-o1 (Gemma-2) | Gemma-2-9B | 58.9 ±0.6 |
| | Gemma-2-27B | 61.0 ±0.5 |

*Table 4.* 100-shot results on the test set of GraphQ. **Bold** numbers indicate the best low-resource performance.

| Method | LLM | F1 |
|---|---|---|
| *Full-Supervised KBQA Methods* | | |
| SPARQA (Sun et al., 2020) | | 21.5 |
| BERT+Ranking (Gu et al., 2021) | | 25.0 |
| ArcaneQA (Gu & Su, 2022) | | 31.8 |
| *Low-resource KBQA Methods* | | |
| KB-BINDER (Li et al., 2023) | GPT-3.5-turbo | 27.1 |
| KB-Coder (Nie et al., 2024) | GPT-3.5-turbo | 31.1 |
| ARG-KBQA (Tian et al., 2024) | GPT-3.5-turbo | - |
| KBQA-o1 (Llama-3) | Llama-3.1-8B | 48.7 ±0.8 |
| | Llama-3.3-70B | 50.5 ±0.2 |
| KBQA-o1 (Qwen2.5) | Qwen2.5-7B | 49.2 ±0.2 |
| | Qwen2.5-14B | 50.0 ±0.7 |
| | Qwen2.5-32B | 50.9 ±0.6 |
| | Qwen2.5-72B | **51.2** ±1.0 |
| KBQA-o1 (Gemma-2) | Gemma-2-9B | 49.8 ±0.7 |
| | Gemma-2-27B | 50.3 ±0.4 |

## 5. Experiments

This section presents the experimental setup, results, and analysis. We answer the following research questions (RQs): **RQ1**: Does KBQA-o1 outperform other KBQA methods? **RQ2**: Does the main component of KBQA-o1 work? **RQ3**: Does KBQA-o1 address the corresponding challenges compared to end-to-end and step-by-step KBQA methods? **RQ4**: How does incremental fine-tuning gradually improve low-resource KBQA performance?

### 5.1. Experimental Setup

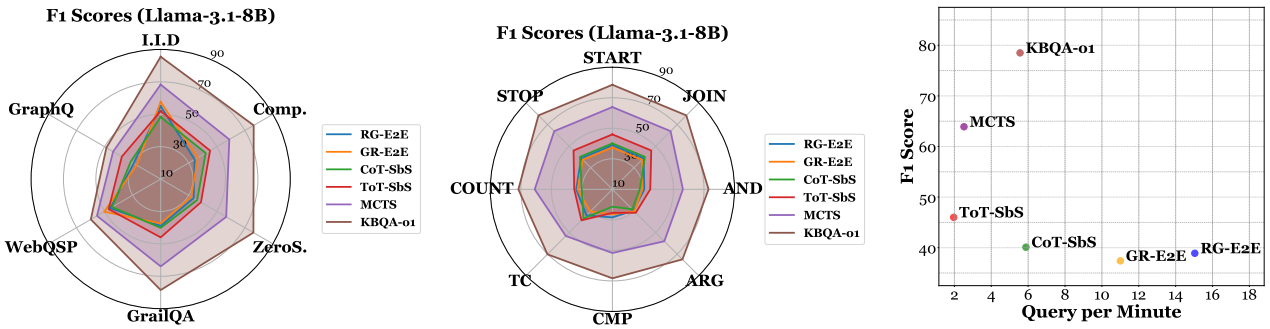
**Datasets.** All experiments are conducted on three standard KBQA datasets in low-resource setting (Li et al., 2023): GrailQA (Gu et al., 2021), WebQSP (Yih et al., 2016), and GraphQ (Su et al., 2016). All datasets are based on Freebase (Bollacker et al., 2008) KB.

**Baselines.** We mainly compare our method with KB-BINDER (Li et al., 2023), KB-Coder (Nie et al., 2024), and ARG-KBQA (Tian et al., 2024) with the same LLM of gpt-3.5-turbo-0613 from OpenAI in low-resource setting. Some results obtained by full-supervised training on the whole training dataset (Ye et al., 2022; Yu et al., 2023; Shu et al., 2022; Sun et al., 2020; Gu et al., 2021; Gu & Su, 2022) are also reported for reference.

**Evaluation Metrics.** In line with prior studies (Li et al., 2023; Nie et al., 2024; Tian et al., 2024), we use F1 score and Exact Match (EM) as the evaluation metric for GrailQA, while F1 score are reported for WebQSP and GraphQ.

*Table 5.* Evaluation results of Llama-3.1-8B-based KBQA-o1 and its ablations. **Bold** numbers indicate the best performance.

| Method | I.I.D | | Compositional | | Zero-shot | | GrailQA | | WebQSP | GraphQ |
|---|---|---|---|---|---|---|---|---|---|---|
| | EM | F1 | EM | F1 | EM | F1 | EM | F1 | F1 | F1 |
| KBQA-o1 (Llama-3.1-8B) | **77.8** ±0.5 | **85.5** ±0.4 | **76.3** ±0.6 | **77.6** ±0.5 | **68.1** ±0.8 | **76.1** ±0.4 | **71.9** ±0.3 | **78.5** ±1.0 | **59.8** ±1.2 | **48.7** ±0.8 |
| w/o *agent prompt* | 65.2 ±1.1 | 72.8 ±0.8 | 61.9 ±1.3 | 63.2 ±1.2 | 45.7 ±1.6 | 49.8 ±1.4 | 51.2 ±2.3 | 56.4 ±3.1 | 52.3 ±1.2 | 37.7 ±2.2 |
| w/o *KB environment* | 43.5 ±1.4 | 51.3 ±1.3 | 38.7 ±1.7 | 41.5 ±1.5 | 29.6 ±1.9 | 35.4 ±1.6 | 34.9 ±2.0 | 43.0 ±1.6 | 49.3 ±0.7 | 25.1 ±1.4 |
| w/o *initial annotated sft* | 14.3 ±3.9 | 17.4 ±4.2 | 12.2 ±4.7 | 14.5 ±4.1 | 8.5 ±3.6 | 10.7 ±4.4 | 10.6 ±4.3 | 13.1 ±4.0 | 17.2 ±2.9 | 9.5 ±3.0 |
| w/o *MCTS* | 47.9 ±1.6 | 54.2 ±1.5 | 42.6 ±2.1 | 45.8 ±1.9 | 40.3 ±1.8 | 44.2 ±2.0 | 43.8 ±2.6 | 48.5 ±3.7 | 44.0 ±2.1 | 20.8 ±2.6 |
| w/o *incremental fine-tuning* | 60.7 ±1.0 | 68.3 ±1.2 | 56.2 ±1.5 | 58.9 ±1.4 | 53.8 ±1.7 | 56.7 ±1.5 | 57.1 ±1.0 | 63.9 ±1.6 | 55.4 ±2.4 | 43.9 ±1.7 |




(a) (b) (c)

*Figure 4.* Performance and efficiency comparison of Llama-3.1-8B-based KBQA-o1 with compared methods. (a) F1 scores comparison across datasets. (b) F1 scores across logical operators on GrailQA. (c) Trade-off between F1 scores and queries per minute on GrailQA.

**Implementation Details.** Following KB-BINDER (Li et al., 2023), we conduct 40-shot experiments for GrailQA, and 100-shot for WebQSP and GraphQ. During the MCTS exploration phase, we set $\theta_{\text{exp}}$ with $w = 50$, while in the prediction phase, we set $\theta_{\text{eff}}$ with $w = 10$. We select multiple open-source 7B-72B LLMs, including Llama-3 (Dubey et al., 2024), Qwen2.5 (Yang et al., 2025), and Gemma-2 (Team et al., 2024), to construct KBQA-o1. All experiments are done on 8 NVIDIA A40 GPUs (48GB), with results averaged from three randomly seeded experiments. Appendix G shows the optimal hyperparameter settings.

### 5.2. Main Result (RQ1)

As shown in Tables 2, 3, and 4, KBQA-o1 enables open-source LLM to outperform previous low-resource KBQA methods based on GPT-3.5-turbo, with limited labeled data. In more complex data sets such as GrailQA, KBQA-o1 improves the overall EM performance of the Llama-3.1-8B model by 28.1 percentage points and boosts F1 scores by 30.0 percentage points over the previous best methods. In compositional and zero-shot evaluations, KBQA-o1 even outperforms fully supervised KBQA methods, which demonstrates its strong capabilities for generalization, environment exploration, and handling complex logical questions. Moreover, KBQA-o1 is plug-and-play, allowing integration with various open-source 7B-72B LLMs, expected to improve further with future open-source LLM updates.

### 5.3. Ablation Study (RQ2)

As shown in Table 5, we conduct an ablation study on the Llama-3.1-8B-based KBQA-o1 methods. Five ablation settings are tested: removing the ReAct-based agent prompt, removing environment feedback, excluding the initial supervised fine-tuning (SFT) with a small amount of labeled data, removing MCTS optimization, and omitting the incremental fine-tuning stage, respectively. Comparisons of the evaluation results reveal that all modules contribute to overall performance, underscoring the importance of designed agent initialization, heuristic environment exploration, and incremental fine-tuning as key components of KBQA-o1.

### 5.4. Comparison Analysis (RQ3)

To explore how KBQA-o1 addresses the challenges of end-to-end and step-by-step methods mentioned in Section 1, we construct six variants. We design two end-to-end variants: one is a retrieve-then-generate method (RG-E2E) based on DECAF (Yu et al., 2023), and the other is a generate-then-retrieve method (GR-E2E), based on ChatKBQA (Luo et al., 2024a). Furthermore, we design two step-by-step variants: one is based on CoT (CoT-SbS) as QueryAgent (Huang et al., 2024), and the other is based on ToT (ToT-SbS) as ToG (Sun et al., 2024). Finally, we implement an MCTS-optimized variant without incremental fine-tuning and the full KBQA-o1 method after incremental fine-tuning.

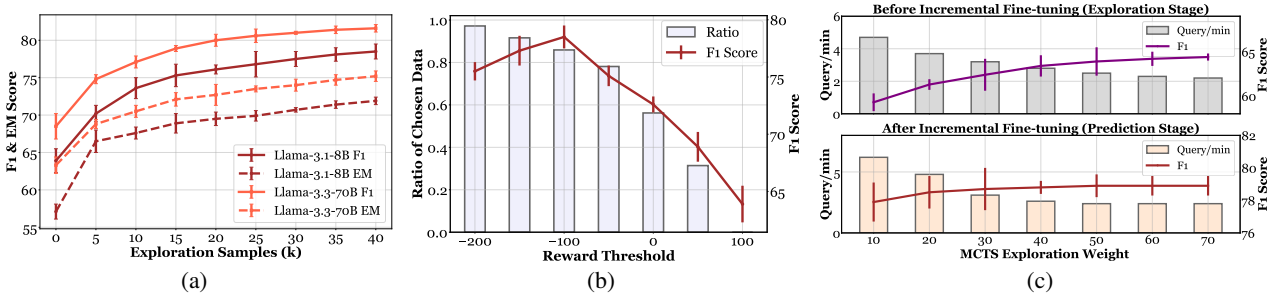



*Figure 5.* Impact of incremental fine-tuning tested on GrailQA: (a) Effect of exploration samples on F1 and EM scores. (b) Relationship between reward threshold, data ratio, and performance. (c) Influence of MCTS exploration weight on query efficiency and accuracy.

**Comparison with End-to-end Methods.** To investigate whether the agent process in KBQA-o1 is more capable of perceiving the KB environment than end-to-end methods, we compare CoT-SbS with RG-E2E and GR-E2E. Under the same prompt, CoT-SbS is equivalent to the agent process in KBQA-o1 without expansion. As shown in Figures 4(a) and 4(b), in the I.I.D and WebQSP evaluations with annotations in distribution, RG-E2E and GR-E2E perform better. However, in Compositional and Zero-shot evaluations, the agent process represented by CoT-SbS performs better. This is because, with preannotated logical forms, end-to-end generation rigidly adheres to a fixed logical form structure. In contrast, the step-by-step agent approach allows stepwise adjustments by KB environment awareness, enabling it to handle more complex out-of-distribution questions.

**Comparison with Step-by-step Methods.** To investigate whether the MCTS proposed in KBQA-o1 can further prevent the step-by-step agent process from getting stuck in local optima or dealing with a large search space, we compared MCTS with CoT-SbS and ToT-SbS. As shown in Figures 4(a) and 4(b), MCTS consistently outperform CoT-SbS and ToT-SbS in all evaluation tasks and logical operators. To analyze efficiency, Figure 4(c) shows that the efficiency of MCTS and KBQA-o1 falls between CoT-SbS and ToT-SbS. When exploring unlabeled data, the MCTS prioritizes correctness, while the full KBQA-o1, after incremental fine-tuning, adopts more efficient settings to balance performance and search time.

### 5.5. Analysis of Incremental Improvement (RQ4)

To explore the impact of incremental fine-tuning on improving the performance of KBQA-o1 in low-resource KBQA scenarios, we examine three key factors: the impacts of the exploration samples, the reward threshold, and the exploration weights on performance and efficiency.

**Impact of Exploration Samples.** As shown in Figure 5(a), we gradually increase the number of unlabeled exploration samples and observe a steady improvement in the F1 and EM scores after incremental fine-tuning. This suggests that newly explored and labeled samples contribute to automatically generating high-quality logical forms using both environmental and reward-based filtering, allowing KBQA to improve with minimal human annotation, making it highly promising for transferability and real-world applications.

**Impact of Reward Threshold $\gamma^*$.** As shown in Figure 5(b), we conduct experiments with reward thresholds $\gamma^*$ ranging from -200 to 100. As $\gamma^*$ increases, the proportion of post-labeled data decreases until no data are selected at the maximum threshold of 100. In GrailQA, setting $\gamma^*$ to -100 retained most labeled samples while filtering out lower-quality ones, resulting in the highest F1 score. This indicates that selecting an appropriate $\gamma^*$ helps preserve newly labeled high-quality samples while discarding erroneous samples, making incremental fine-tuning effective.

**Impact of Exploration Weight $w$.** Figure 5(c) presents a study on the selection of the exploration weight $w$ in MCTS before and after incremental fine-tuning. We test seven different values between 10 and 70. The results show that higher $w$ leads to slower generation, but improves accuracy. Based on this, we set $w \in \theta_{\text{exp}}$ at 50 before incremental fine-tuning to ensure accuracy. After incremental fine-tuning, we reduce $w \in \theta_{\text{eff}}$ to 10 to balance efficiency and effectiveness. Hence, we ensure that the MCTS with $\theta_{\text{exp}}$ performs a slow and careful search during exploration, leading to high-quality data for incremental fine-tuning. After that, we can implement a more efficient MCTS with $\theta_{\text{eff}}$ in applications, maintaining high performance while improving efficiency.

## 6. Conclusion

In this work, we propose KBQA-o1, an agentic KBQA method with Monte Carlo Tree Search (MCTS) for efficient exploration. By combining a ReAct-based agent process with incremental fine-tuning, it improves logical form generation and reduces reliance on annotated data. Experiments on three KBQA datasets show that KBQA-o1 outperforms previous low-resource methods and rivals fully supervised models, demonstrating its scalability and effectiveness.

## Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 62176026, Grant No. 62473271, and Grant No. 62406036). This work is also supported by the BUPT Excellent Ph.D. Students Foundation (No. CX2023133) and the Engineering Research Center of Information Networks, Ministry of Education, China.

## Impact Statement

This work introduces KBQA-o1, a novel agentic approach to knowledge base question answering (KBQA) using Monte Carlo Tree Search (MCTS). While effective, it still faces limitations in fine-grained policy design and scalability to larger domains. To address these challenges, we outline four future directions: (1) exploring reinforcement learning for continual learning, (2) enhancing logical reasoning capabilities, (3) adapting to specialized domains such as medicine and law, and (4) extending to multimodal, multilingual, and multi-agent settings, as detailed in Appendix K. This work poses no ethical concerns, as it relies solely on publicly available datasets and aims to advance KBQA technology with positive societal impacts.

## References

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pp. 1247–1250, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605581026. doi: 10.1145/1376616.1376746. URL https://doi.org/10.1145/1376616.1376746.

Cui, J., Li, Z., Yan, Y., Chen, B., and Yuan, L. Chatlaw: Open-source legal large language model with integrated external knowledge bases. *CoRR*, abs/2306.16092, 2023. URL https://doi.org/10.48550/arXiv.2306.16092.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Gao, T., Yao, X., and Chen, D. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6894–6910, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.552. URL https://aclanthology.org/2021.emnlp-main.552.

Gu, Y. and Su, Y. ArcaneQA: Dynamic program induction and contextualized encoding for knowledge base question answering. In Calzolari, N., Huang, C.-R., Kim, H., Pustejovsky, J., Wanner, L., Choi, K.-S., Ryu, P.-M., Chen, H.-H., Donatelli, L., Ji, H., Kurohashi, S., Paggio, P., Xue, N., Kim, S., Hahm, Y., He, Z., Lee, T. K., Santus, E., Bond, F., and Na, S.-H. (eds.), *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 1718–1731, Gyeongju, Republic of Korea, October 2022. International Committee on Computational Linguistics. URL https://aclanthology.org/2022.coling-1.148/.

Gu, Y., Kase, S., Vanni, M., Sadler, B., Liang, P., Yan, X., and Su, Y. Beyond i.i.d.: Three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, WWW '21, pp. 3477–3488, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383127. doi: 10.1145/3442381.3449992. URL https://doi.org/10.1145/3442381.3449992.

Gu, Y., Deng, X., and Su, Y. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4928–4949, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.270. URL https://aclanthology.org/2023.acl-long.270.

Hao, S., Gu, Y., Ma, H., Hong, J., Wang, Z., Wang, D., and Hu, Z. Reasoning with language model is planning with world model. In Bouamor, H., Pino, J., and Bali, K. (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 8154–8173, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.507. URL https://aclanthology.org/2023.emnlp-main.507.

Huang, X., Cheng, S., Huang, S., Shen, J., Xu, Y., Zhang, C., and Qu, Y. QueryAgent: A reliable and efficient reasoning framework with environmental feedback based self-correction. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5014–5035, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.274. URL https://aclanthology.org/2024.acl-long.274.

Jang, H., Oh, Y., Jin, S., Jung, H., Kong, H., Lee, D., Jeon, D., and Kim, W. Kbqa: constructing structured query

graph from keyword query for semantic search. In *Proceedings of the International Conference on Electronic Commerce*, ICEC '17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450353120. doi: 10.1145/3154943.3154955. URL https://doi.org/10.1145/3154943.3154955.

Li, T., Ma, X., Zhuang, A., Gu, Y., Su, Y., and Chen, W. Few-shot in-context learning on knowledge base question answering. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6966–6980, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.385. URL https://aclanthology.org/2023.acl-long.385.

Li, Z., Liu, H., Zhou, D., and Ma, T. Chain of thought empowers transformers to solve inherently serial problems. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3EWTEy9MTM.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Luo, H., E, H., Tang, Z., Peng, S., Guo, Y., Zhang, W., Ma, C., Dong, G., Song, M., Lin, W., Zhu, Y., and Luu, A. T. ChatKBQA: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Findings of the Association for Computational Linguistics ACL 2024*, pp. 2039–2056, Bangkok, Thailand and virtual meeting, August 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.122. URL https://aclanthology.org/2024.findings-acl.122.

Luo, L., Li, Y.-F., Haf, R., and Pan, S. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=ZGNWW7xZ6Q.

Nie, Z., Zhang, R., Wang, Z., and Liu, X. Code-style in-context learning for knowledge-based question answering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):18833–18841, Mar. 2024. doi: 10.1609/aaai.v38i17.29848. URL https://ojs.aaai.org/index.php/AAAI/article/view/29848.

OpenAI. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

Pérez, J., Arenas, M., and Gutierrez, C. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3), September 2009. ISSN 0362-5915. doi: 10.1145/1567274.1567278. URL https://doi.org/10.1145/1567274.1567278.

Sastry, S. and Sastry, S. Lyapunov stability theory. *Nonlinear systems: analysis, stability, and control*, pp. 182–234, 1999.

Shu, Y., Yu, Z., Li, Y., Karlsson, B., Ma, T., Qu, Y., and Lin, C.-Y. TIARA: Multi-grained retrieval for robust question answering over large knowledge base. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 8108–8121, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.555. URL https://aclanthology.org/2022.emnlp-main.555.

Su, Y., Sun, H., Sadler, B., Srivatsa, M., Gür, I., Yan, Z., and Yan, X. On generating characteristic-rich question sets for QA evaluation. In Su, J., Duh, K., and Carreras, X. (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 562–572, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1054. URL https://aclanthology.org/D16-1054/.

Sun, H., Dhingra, B., Zaheer, M., Mazaitis, K., Salakhutdinov, R., and Cohen, W. 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*, pp. 4231–4242, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1455. URL https://aclanthology.org/D18-1455.

Sun, H., Bedrax-Weiss, T., and Cohen, W. 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)*, pp. 2380–2390, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1242. URL https://aclanthology.org/D19-1242.

Sun, J., Xu, C., Tang, L., Wang, S., Lin, C., Gong, Y., Ni, L., Shum, H.-Y., and Guo, J. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=nnVO1PvbTv.

Sun, Y., Zhang, L., Cheng, G., and Qu, Y. Sparqa: Skeleton-based semantic parsing for complex questions over knowl-

edge bases. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8952–8959, Apr. 2020. doi: 10.1609/aaai.v34i05.6426. URL https://ojs.aaai.org/index.php/AAAI/article/view/6426.

Świechowski, M., Godlewski, K., Sawicki, B., and Mańdziuk, J. Monte carlo tree search: a review of recent modifications and applications. *Artif. Intell. Rev.*, 56(3):2497–2562, July 2022. ISSN 0269-2821. doi: 10.1007/s10462-022-10228-y. URL https://doi.org/10.1007/s10462-022-10228-y.

Team, G., Riviere, M., Pathak, S., Sessa, P. G., Hardin, C., Bhupatiraju, S., Hussenot, L., Mesnard, T., Shahriari, B., Ramé, A., et al. Gemma 2: Improving open language models at a practical size, 2024. URL https://arxiv.org/abs/2408.00118.

Tian, Y., Song, D., Wu, Z., Zhou, C., Wang, H., Yang, J., Xu, J., Cao, R., and Wang, H. Augmenting reasoning capabilities of LLMs with graph structures in knowledge base question answering. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 11967–11977, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.699. URL https://aclanthology.org/2024.findings-emnlp.699/.

Vrandečić, D. and Krötzsch, M. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, sep 2014. ISSN 0001-0782. doi: 10.1145/2629489. URL https://doi.org/10.1145/2629489.

Wang, J., Sun, K., Luo, L., Wei, W., Hu, Y., Liew, A. W.-C., Pan, S., and Yin, B. Large language models-guided dynamic adaptation for temporal knowledge graph reasoning. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 8384–8410. Curran Associates, Inc., 2024.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022.

Wu, J., Zhu, J., Qi, Y., Chen, J., Xu, M., Menolascina, F., and Grau, V. Medical graph rag: Towards safe medical large language model via graph retrieval-augmented generation, 2024. URL https://arxiv.org/abs/2408.04187.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822. Curran Associates, Inc., 2023a.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Ye, X., Yavuz, S., Hashimoto, K., Zhou, Y., and Xiong, C. RNG-KBQA: Generation augmented iterative ranking for knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6032–6043, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.417. URL https://aclanthology.org/2022.acl-long.417.

Yih, W.-t., Richardson, M., Meek, C., Chang, M.-W., and Suh, J. 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)*, pp. 201–206, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-2033. URL https://aclanthology.org/P16-2033.

Yu, D., Zhang, S., Ng, P., Zhu, H., Li, A. H., Wang, J., Hu, Y., Wang, W. Y., Wang, Z., and Xiang, B. DecAF: Joint decoding of answers and logical forms for question answering over knowledge bases. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=XHc5zRPxqV9.

Zhang, J., Zhang, X., Yu, J., Tang, J., Tang, J., Li, C., and Chen, H. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5773–5784, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.396. URL https://aclanthology.org/2022.acl-long.396.

# Appendix

## A. Prompts Used in KBQA-o1

### A.1. Initial Prompt

Figure 6 illustrates the initial prompt used in the KBQA-o1 agent process. This prompt defines the structure and steps for solving KBQA tasks through interleaving Thought, Action, and Observation stages. The prompt guides the agent to perform specific actions, such as entity extraction, relation finding, expression merging, ordering, numerical comparisons, and adding time constraints. These predefined actions are essential for generating logical forms step by step from natural language questions. The diagram highlights the structured input format (<input> for the question and <output> for logical expressions) and the scratchpad used for intermediate reasoning steps during the task.

I want you to be a good knowledge graph query generator, solving a knowledge base question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be eight types:
**(1) Extract_entity [ entity name ]**, where we identify a topic entity from the question to start a new expression.
**(2) Find_relation [ relation name ]**, where we find the one-hop relation that is connected to the current expression.
**(3) Merge [ expression1 | expression2 ]**, where we merge these two expressions.
**(4) Order [ mode | relation ]**, where we perform a sorting operation and impose a constraint to output either the maximum or minimum value.
**(5) Compare [ mode | relation ]**, where we perform a numerical comparison to determine the range.
**(6) Time_constraint [ relation | time ]**, where we add a time constraint.
**(7) Count [ expression ]**, where we perform a counting operation to determine the number of answers.
**(8) Finish [ expression ]**, where we conclude that it is appropriate to end and output the expression.

**<input>**
**Question:** {question}
**<output>**
{scratchpad}

*Figure 6.* Initial prompt of agent process.

### A.2. Example Agent Process

Figure 7 provides an example of the complete agent process in KBQA-o1 for the query. The figure demonstrates how the agent iteratively constructs the logical form using the Thought-Action-Observation prompt. Each step involves reasoning about the current context (Thought), performing a specific operation (Action), and observing the resulting logical expression (Observation). The process begins with extracting the topic entity followed by finding relevant relations and applying numerical and time constraints. The agent merges expressions to form a complete logical form, which is then executed to retrieve the answer from the knowledge base. The final logical output form is shown at the bottom, showcasing the agent's ability to generate structured and executable queries systematically.

**Input Question:** What movie with a television running time of less than 60 minutes features Taylor Lautner?

**Thought1:** At this step, we should identify a topic entity from the question to start a new expression.
**Action1:** Extract_entity [ Taylor Lautner ]
**Observation1:** expression = START('m.07ldhs')
**Thought2:** At this step, we should find the one-hop relation that is connected to the current expression.
**Action2:** Find_relation [ film.actor.film ]
**Observation2:** expression = JOIN('(R film.actor.film)', expression)
**Thought3:** At this step, we should find the one-hop relation that is connected to the current expression.
**Action3:** Find_relation [ film.performance.film ]
**Observation3:** expression = JOIN('(R film.performance.film)', expression)
**Thought4:** At this step, we should identify a topic entity from the question to start a new expression.
**Action4:** Extract_entity [ 60 ]
**Observation4:** expression1 = START('60')
**Thought5:** At this step, we should perform a numerical comparison to determine the range.
**Action5:** Compare [ LESS THAN | tv.tv_program.episode_running_time ]
**Observation5:** expression1 = CMP('lt', 'tv.tv_program.episode_running_time', expression1)
**Thought6:** At this step, we should merge these two expressions.
**Action6:** Merge [ expression1 | expression ]
**Observation6:** expression = AND(expression1, expression)
**Thought7:** At this step, we conclude that it is appropriate to end and output the expression.
**Action7:** Finish [ expression ]
**Observation7:** expression = STOP(expression)

**Equivalent Logical Form:**
(AND (lt tv.tv_program.episode_running_time 60) (JOIN (R film.performance.film) (JOIN (R film.actor.film) Taylor Lautner)))

*Figure 7.* An example of a complete agent process.

## B. Proof

### B.1. Proof of Proposition 4.1

**Proposition 4.1.** *The agent's awareness of the environment makes it more effective in generating optimal logical forms compared to end-to-end methods.*

*Proof.* According to the Lyapunov Stability Second Theorem (Sastry & Sastry, 1999):

For a dynamic system $\dot{x} = f(x)$, where $x$ is the state and $f(x)$ is the nonlinear function describing state evolution. Define a non-negative scalar function $V(x)$ such that: $V(x) > 0$ for all $x \neq 0$ and $V(0) = 0$ (positive definiteness). Its time derivative along the system trajectory satisfies $\dot{V}(x) < 0$ and is bounded (negative definiteness). Then $V(x)$ is called a Lyapunov function candidate, and the system (from Lyapunov's perspective) is asymptotically stable.

For a discrete dynamic system $x_{t+1} = f(x_t)$, if a Lyapunov function $V(x)$ can be define, and the Lyapunov increment $\Delta V(x_t) = V(x_{t+1}) - V(x_t)$, then $V(x_t) > 0$ and $\Delta V(x_t) < 0$ with bounded implies asymptotic stability.

Our KBQA-o1 system is modeled as a discrete dynamic system:

$$h_{t+1} = f(h_t, \mathcal{G}), \tag{18}$$

where $h_t \in \mathcal{H}$ is the system state at step $t$; $\mathcal{G}$ is environment feedback. The state transition probability is defined as: $P(h_{t+1} \mid h_t, \mathcal{G})$, which represents the probability of transitioning to the next state $h_{t+1}$, given the current state $h_t$ and the environment feedback $\mathcal{G}$. The goal is for the system state $h_t$ to converge asymptotically to the target state $h^*$, to satisfy $P(h^* \mid h_t, \mathcal{G}) = 1$.

We define a positive definite Lyapunov function $V(h_t)$ as:

$$V(h_t) = -\log P(h_t \mid h^*, \mathcal{G}), \tag{19}$$

where $P(h_t \mid h^*, \mathcal{G})$ is the posterior probability of state $h_t$ relative to the target state $h^*$; $V(h_t)$ quantifies the deviation of the current state $h_t$ from the target state $h^*$.

The posterior probability satisfies $0 < P(h_t \mid h^*, \mathcal{G}) \leq 1$. Thus, $V(h_t) = -\log P(h_t \mid h^*, \mathcal{G}) \geq 0$. When $h_t = h^*$, $P(h_t \mid h^*, \mathcal{G}) = 1 \quad \Longrightarrow \quad V(h_t) = 0$. Thus, $V(h_t)$ is positive definite.

The change in the Lyapunov function between steps is defined as:

$$\Delta V(h_t) = V(h_{t+1}) - V(h_t). \tag{20}$$

Using the definition of $V(h_t)$, we get $V(h_{t+1})$:

$$V(h_{t+1}) = -\log P(h_{t+1} \mid h^*, \mathcal{G}), \tag{21}$$

and the posterior probability recursion:

$$P(h_{t+1} \mid h^*, \mathcal{G}) = P(h_t \mid h^*, \mathcal{G}) \cdot P(h_{t+1} \mid h_t, \mathcal{G}), \tag{22}$$

we have:

$$V(h_{t+1}) = -\log P(h_t \mid h^*, \mathcal{G}) - \log P(h_{t+1} \mid h_t, \mathcal{G}). \tag{23}$$

Thus:

$$\Delta V(h_t) = V(h_{t+1}) - V(h_t) = -\log P(h_{t+1} \mid h_t, \mathcal{G}). \tag{24}$$

The probability $P(h_{t+1} \mid h_t, \mathcal{G})$ can be expressed as:

$$P(h_{t+1} \mid h_t, \mathcal{G}) = P(\text{Exp}(h_t, \mathcal{G}) \mid h_t, \mathcal{G}) \cdot P(h_{t+1} \mid h_t, \text{Exp}(h_t, \mathcal{G})), \tag{25}$$

where $P(\text{Exp}(h_t, \mathcal{G}) \mid h_t, \mathcal{G})$ is the probability that the environment generates an exploration space $\text{Exp}(h_t, \mathcal{G})$, containing valid actions for transitioning to $h_{t+1}$. And $P(h_{t+1} \mid h_t, \text{Exp}(h_t, \mathcal{G}))$ is the probability that the system transitions from $h_t$ to $h_{t+1}$, given the exploration space.

In KBQA-o1, our environment can provide all possible explorations under the prior state, and the correct exploration step is guaranteed to be included. Thus, $P(\text{Exp}(h_t, \mathcal{G}) \mid h_t, \mathcal{G}) = 1$. Therefore, $P(h_{t+1} \mid h_t, \mathcal{G}) = P(h_{t+1} \mid h_t, \text{Exp}(h_t, \mathcal{G}))$, and given a set of explorations that contains the correct one, the probability of selecting the correct exploration satisfies $0 < P(h_{t+1} \mid h_t, \text{Exp}(h_t, \mathcal{G})) < 1$. Consequently, $0 < P(h_{t+1} \mid h_t, \mathcal{G}) < 1$.

Since $0 < P(h_{t+1} \mid h_t, \mathcal{G}) < 1$, we conclude $\Delta V(h_t) < 0$ and is bounded. Thus, the Lyapunov function change is strictly negative definite.

Based on the Lyapunov Stability Second Theorem, $V(h_t) = -\log P(h_t \mid h^*, \mathcal{G})$ satisfies positive definiteness: $V(h_t) \geq 0$, and $V(h_t) = 0$ only when $h_t = h^*$, and negative definiteness: $\Delta V(h_t) < 0$, ensuring $V(h_t)$ strictly decreases. Thus, the Agent system is asymptotically stable in the Lyapunov sense, and the state $h_t$ converges to $h^*$.

However, for end-to-end methods, although both pre-retrieval and post-retrieval approaches can incorporate environmental information, pre-retrieval may result in an exploration space that does not include the target exploration, leading to $P(\text{Exp}(h_t, \mathcal{G}) \mid h_t, \mathcal{G}) = 0$. Post-retrieval, on the other hand, may generate a logical form framework that is incorrect, resulting in $P(h_{t+1} \mid h_t, \text{Exp}(h_t, \mathcal{G})) = 0$. In such cases, this leads to $P(h_{t+1} \mid h_t, \mathcal{G}) = P(\text{Exp}(h_t, \mathcal{G}) \mid h_t, \mathcal{G}) \cdot P(h_{t+1} \mid h_t, \text{Exp}(h_t, \mathcal{G})) = 0$, causing $\Delta V(h_t)$ isn't bounded, leading to instability.

Therefore, we conclude that the agent's awareness of the environment makes it more effective in generating optimal logical forms compared to end-to-end methods.

□

### B.2. Proof of Proposition 4.2

**Proposition 4.2.** *The MCTS-based heuristic method balances the effectiveness and size of the search space better than CoT-based and ToT-based step-by-step methods.*

*Proof.* Let $\mathcal{F}$ represent the search space of logical forms, with a size of $|\mathcal{F}| = k^L$, where $k$ is the number of choices per step, and $L$ is the maximum search depth. Let $\mathcal{F}_{\text{optimal}} \subseteq \mathcal{F}$ denote the set of high-quality solutions whose scores exceed a threshold $\tau$. For any search method $X$, if its generated candidate set is $\mathcal{F}_X \subseteq \mathcal{F}$, the coverage rate is defined as:

$$C_X = \frac{|\mathcal{F}_X \cap \mathcal{F}_{\text{optimal}}|}{|\mathcal{F}_{\text{optimal}}|}. \tag{26}$$

We first compare CoT (Chain-of-Thought) and our proposed MCTS-based method. CoT follows a single path $\mathbf{h}_l$, producing a single candidate $F_{\text{CoT}}$, with its candidate set given by $\mathcal{F}_{\text{CoT}} = \{F_{\text{CoT}}\}$. If $F_{\text{CoT}} \in \mathcal{F}_{\text{optimal}}$, the coverage rate is

$$C_{\text{CoT}} = \frac{1}{|\mathcal{F}_{\text{optimal}}|}, \tag{27}$$

otherwise, $C_{\text{CoT}} = 0$. MCTS, on the other hand, expands multiple different paths during multiple rollouts. If $N$ rollouts are performed in total, the resulting candidate set is

$$\mathcal{F}_{\text{MCTS}} = \bigcup_{n=1}^{N} \{\mathbf{h}_l^{(n)}\}, \tag{28}$$

where $\mathbf{h}_l^{(n)}$ is the n-th path generated. Under effective strategy and reward guidance, most paths will fall into $\mathcal{F}_{\text{optimal}}$, and thus

$$C_{\text{MCTS}} = \frac{|\mathcal{F}_{\text{MCTS}} \cap \mathcal{F}_{\text{optimal}}|}{|\mathcal{F}_{\text{optimal}}|} \gg C_{\text{CoT}}. \tag{29}$$

This demonstrates that MCTS significantly outperforms CoT in terms of coverage of high-quality solutions.

Next, we compare MCTS and ToT (Tree of Thought) based on BFS/DFS. ToT explores all possible paths in $\mathcal{F}$, so its max candidate set is $\mathcal{F}_{\text{ToT}} = \mathcal{F}$. The coverage rate for ToT is

$$C_{\text{ToT}} = \frac{|\mathcal{F} \cap \mathcal{F}_{\text{optimal}}|}{|\mathcal{F}_{\text{optimal}}|} = 1, \tag{30}$$

but this requires a time complexity of

$$T_{\text{ToT}} = O(k^L). \tag{31}$$

MCTS, however, only explores part of the tree. Assuming each rollout expands $\omega k$ candidates ($\omega \in (0, 1)$), and N rollouts are performed, the total search size is $N \cdot \omega k \cdot L$, with a time complexity of

$$T_{\text{MCTS}} = O(N \cdot \omega k \cdot L). \tag{32}$$

As long as $N \cdot \omega k \cdot L \ll k^L$, we have

$$T_{\text{MCTS}} \ll T_{\text{ToT}}. \tag{33}$$

Moreover, if the strategy and reward models effectively focus the search on high-potential nodes,

$$|\mathcal{F}_{\text{MCTS}} \cap \mathcal{F}_{\text{optimal}}| \approx |\mathcal{F}_{\text{optimal}}|, \tag{34}$$

leading to

$$C_{\text{MCTS}} \approx C_{\text{ToT}} = 1. \tag{35}$$

Combining these results, we see that compared to CoT, MCTS significantly improves the coverage rate by expanding multiple paths. Compared to ToT, MCTS achieves a near-equal coverage rate with substantially lower time complexity. Thus, MCTS balances effectiveness (coverage rate) and efficiency (search space size) better than both CoT and ToT. □

### B.3. Proof of Proposition 4.3

**Proposition 4.3.** *There exists a reward threshold $\gamma^* < \beta$ such that incremental fine-tuning data, under the joint effect of the KB and the reward model, can significantly improve model performance.*

*Proof.* First, we define the performance improvement of incremental fine-tuning, $\Delta E(\gamma^*)$. The model's initial performance on the annotated dataset $\mathcal{D}_a$ is denoted as $E_{\text{base}}$, and its performance after incremental fine-tuning is $E_{\text{new}} = E_{\text{base}} + \Delta E(\gamma^*).Here, \Delta E(\gamma^*)$ represents the performance gain contributed by incremental data, which depends on the quantity and quality of the incremental data.

The incremental data originates from the unannotated dataset $\mathcal{D}_n$, where the logical forms $\hat{F}^{\mathcal{Q}}$ and answers $\hat{\mathcal{A}}^{\mathcal{Q}}$ are generated through a filtering mechanism. The filtering criteria are as follows: 1) The reward score of the logical form $R_{\pi_{\text{reward}}}(F) > \gamma^*$; 2) The answer set is non-empty, $\hat{\mathcal{A}}^{\mathcal{Q}} \neq \emptyset$.

The logical forms that satisfy the criteria form the incremental dataset $\mathcal{D}_i(\gamma^*)$. The size and quality of the incremental data are both dependent on the reward threshold $\gamma^*$. The performance improvement can be expressed as:

$$\Delta E(\gamma^*) = \lambda \cdot N_{\text{high}}(\gamma^*) \cdot Q_{\text{avg}}(\mathcal{D}_i(\gamma^*)), \tag{36}$$

where $N_{\text{high}}(\gamma^*)$ is the quantity of incremental data; $Q_{\text{avg}}(\mathcal{D}_i(\gamma^*))$ is the average quality of incremental data; $\lambda > 0$ is a scaling factor representing the contribution of data quality to the performance gain.

Next, we analyze the properties of $N_{\text{high}}(\gamma^*)$ and $Q_{\text{avg}}(\mathcal{D}_i(\gamma^*))$, and subsequently derive the behavior of $\Delta E(\gamma^*)$.

The quantity of incremental data $N_{\text{high}}(\gamma^*)$ depends on the number of logical forms in the unannotated dataset that satisfy $R_{\pi_{\text{reward}}}(F) > \gamma^*$. Let $N_n = |\mathcal{D}n|$ be the size of the unannotated dataset, and let the reward scores of logical forms follow a probability density function $P(R)$. Then:

$$N_{\text{high}}(\gamma^*) = N_n \cdot P_{\text{high}}(\gamma^*), \tag{37}$$

where $P_{\text{high}}(\gamma^*)$ represents the probability that the reward score exceeds $\gamma^*$:

$$P_{\text{high}}(\gamma^*) = \int_{\gamma^*}^{\beta} P(R)\, dR. \tag{38}$$

Clearly, as $\gamma^*$ increases, $N_{\text{high}}(\gamma^*)$ decreases. When $\gamma^* \to -\infty$, $N_{\text{high}}(\gamma^*) \to N_n; when \gamma^* \to \beta, N_{\text{high}}(\gamma^*) \to 0$.

The quality of the incremental data $Q_{\text{avg}}(\mathcal{D}i(\gamma^*))$ is the average reward score of logical forms with $R\pi_{\text{reward}}(F) > \gamma^*$, defined as:

$$Q_{\text{avg}}(\mathcal{D}_i(\gamma^*)) = \frac{\int_{\gamma^*}^{\beta} R \cdot P(R)\, dR}{\int_{\gamma^*}^{\beta} P(R)\, dR}. \tag{39}$$

The numerator $\int_{\gamma^*}^{\beta} R \cdot P(R)\, dR$ is the total quality weighted by reward scores, while the denominator $\int_{\gamma^*}^{\beta} P(R)\, dR = P_{\text{high}}(\gamma^*)$ is the total probability. When $\gamma^* \to -\infty$, all logical forms are included in the incremental data, and the average quality is the expected reward score:

$$Q_{\text{avg}}(\mathcal{D}_i(\gamma^*)) \to \mathbb{E}[R] = \int_{-\infty}^{\beta} R \cdot P(R)\, dR. \tag{40}$$

When $\gamma^* \to \beta$, only the logical forms with the highest reward scores are retained, and the average quality approaches 1.

Now, we analyze the behavior of the performance improvement $\Delta E(\gamma^*)$. Substituting $N_{\text{high}}(\gamma^*)$ and $Q_{\text{avg}}(\mathcal{D}i(\gamma^*))$ into $\Delta E(\gamma^*)$, we obtain:

$$\Delta E(\gamma^*) = \lambda \cdot N_n \cdot \left( \int_{\gamma^*}^{\beta} P(R)\, dR \right) \cdot \frac{\int_{\gamma^*}^{\beta} R \cdot P(R)\, dR}{\int_{\gamma^*}^{\beta} P(R)\, dR}. \tag{41}$$

Simplifying this, we get:

$$\Delta E(\gamma^*) = \lambda \cdot N_n \cdot \int_{\gamma^*}^{\beta} R \cdot P(R)\, dR. \tag{42}$$

To analyze the critical points, we take the derivative of $\Delta E(\gamma^*)$ with respect to $\gamma^*$:

$$\frac{\partial \Delta E(\gamma^*)}{\partial \gamma^*} = -\lambda \cdot N_n \cdot \gamma^* \cdot P(\gamma^*). \tag{43}$$

It is clear that when $\gamma^* \to -\infty$, $\frac{\partial \Delta E(\gamma^*)}{\partial \gamma^*} > 0$, indicating that $\Delta E(\gamma^*)$ increases with $\gamma^*$; when $\gamma^* \to \beta$, $\frac{\partial \Delta E(\gamma^*)}{\partial \gamma^*} < 0$, indicating that $\Delta E(\gamma^*)$ decreases with $\gamma^*$. Thus, $\Delta E(\gamma^*)$ is a unimodal function, and there exists a critical point $\gamma^*_{\text{opt}} \in (-\infty, \beta)$ at which $\Delta E(\gamma^*)$ achieves its maximum.

In conclusion, we have proven that: There exists a reward threshold $\gamma^* < \beta$ such that the trade-off between the quantity and quality of incremental data is optimized. At this threshold, the performance improvement $\Delta E(\gamma^*)$ reaches its maximum. This establishes the existence of $\gamma^*$ and its role in optimizing the effect of incremental fine-tuning.

□

## C. MCTS Algorithm Details

Figure 8 and Algorithm 1 illustrate the Monte Carlo Tree Search (MCTS) process in KBQA-o1. The figure highlights the four stages of MCTS: Selection, where nodes are chosen using the Upper Confidence Bound for Trees (UCT) to balance exploration and exploitation; Expansion, where candidate actions are generated by the policy model, filtered for relevance to the knowledge base, and added as child nodes; Simulation, where the most promising path is explored to produce a complete logical form and compute rewards; and Back-propagation, where rewards are propagated back to update Q-values and visit counts. The pseudocode formalizes this process, iteratively performing rollouts that follow the four stages. It ensures efficient exploration by selecting nodes with high potential, expanding with semantically relevant actions, simulating logical forms, and updating scores through back-propagation. This approach enables KBQA-o1 to navigate large search spaces effectively and generate high-quality logical forms.

**Complexity Analysis.** The time complexity of the MCTS process in KBQA-o1 can be analyzed based on its four stages: Selection, Expansion, Simulation, and Back-propagation. In the Selection stage, the algorithm traverses the search tree up to depth $L$ , selecting the best node using the Upper Confidence Bound for Trees (UCT), leading to a complexity of $O(k \cdot L)$ per rollout, where k is the number of possible actions per step. The Expansion stage generates B beam candidates, which are filtered based on knowledge base similarity, adding $O(\omega k)$ complexity. The Simulation stage explores paths up to depth L , contributing $O(k \cdot L)$ . Finally, the Back-propagation stage updates the rewards along the path, requiring $O(L)$ .

Summing up these steps, the complexity for a single rollout is $O(k \cdot L)$ , and with $N$ rollouts, the total complexity of MCTS is $O(N \cdot k \cdot L)$ , which scales linearly with the number of rollouts and search depth.

Compared to exhaustive search methods like Tree-of-Thoughts (ToT), which have a complexity of $O(k^L)$ , MCTS is significantly more efficient. Instead of exploring all possible paths, MCTS selectively explores high-potential nodes based on the policy model and UCT, reducing redundant computations. This selective approach allows MCTS to scale effectively, making it suitable for large KBs. Additionally, MCTS dynamically balances exploration and exploitation, adapting its search strategy based on reward feedback. This adaptability, combined with its lower computational cost, enables MCTS to generate high-quality logical forms while maintaining efficiency.

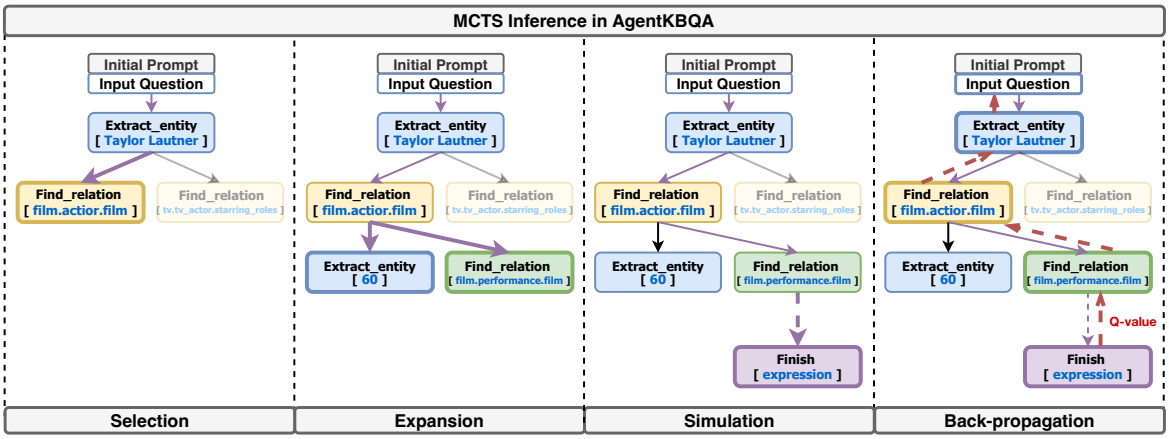



*Figure 8.* Four stages of the MCTS process in KBQA-o1.

**Algorithm 1** MCTS in KBQA-o1

**Require:** Initial state $\mathbf{h}_0$, knowledge base $\mathcal{G}$, policy model $\pi_{\text{policy}}$, reward model $\pi_{\text{reward}}$, beam size $B$, number of top candidates $d$, depth limit $L$, number of rollouts $N$, exploration weight $w$.

**for** $n \leftarrow 1$ to $N$ **do**
  $t \leftarrow 0$
  **while** $N(\mathbf{h}_t) > 0$ **do**
    Select by UCT: $\mathbf{e}_t \leftarrow \arg\max_{\mathbf{e} \in E(\mathbf{h}_{t-1}^{(n)})} \left[ Q(\mathbf{h}_{t-1}^{(n)} + \mathbf{e}) + w \sqrt{\frac{\ln N(\mathbf{h}_{t-1}^{(n)})}{N(\mathbf{h}_{t-1}^{(n)} + \mathbf{e})}} \right]$ ▷ Selection
    Update $\mathbf{h}_t^{(n)} \leftarrow \mathbf{h}_{t-1}^{(n)} + \mathbf{e}_t$
    $t \leftarrow t + 1$
  **end while**
  **while** $\mathbf{h}_t$ is not a terminal state $\wedge\ t \leq L$ **do**
    Beam search by policy model: $\{\mathbf{e}_t^{(b)}\}_{b=1}^{B} \sim \pi_{\text{policy}} \left( \mathbf{e}_t \mid \mathbf{h}_{t-1}^{(n)} \right)_{\text{beam}}$ ▷ Expansion
    Semantic selection from KB: $\{\mathbf{e}_t^{(i)}\}_{i=1}^{k} \leftarrow \arg\max^{k}_{\mathbf{e} \in \text{Exp}(\mathbf{h}_{t-1}^{(n)}, \mathcal{G})} \text{SimCSE}\left( \{\mathbf{e}_t^{(b)}\}_{b=1}^{B}, \mathbf{e} \right)$
    Policy model rerank: $E(\mathbf{h}_{t-1}^{(n)}) = \{\mathbf{e}_t^{(i)}\}_{i=1}^{d} \leftarrow \arg\max^{d} R_{\pi_{\text{policy}}} \left( \{\mathbf{e}_t^{(i)}\}_{i=1}^{k} \mid \mathbf{h}_{t-1}^{(n)} \right)$
    Simulate along the best: $\mathbf{e}_t \leftarrow \arg\max_{\mathbf{e} \in E(\mathbf{h}_{t-1}^{(n)})} R_{\pi_{\text{policy}}} \left( \mathbf{e} \mid \mathbf{h}_{t-1}^{(n)} \right)$ ▷ Simulation
    Update $\mathbf{h}_t^{(n)} \leftarrow \mathbf{h}_{t-1}^{(n)} + \mathbf{e}_t$
    $t \leftarrow t + 1$
  **end while**
  $l \leftarrow t$
  Calculate reward: $Q(\mathbf{h}_l^{(n)}) \leftarrow \delta\ R_{\pi_{\text{policy}}} \left( \mathbf{e}_l \mid \mathbf{h}_{l-1}^{(n)} \right) + (1 - \delta)\ R_{\pi_{\text{reward}}} \left( F_{\mathbf{h}_l^{(n)}} \mid \mathcal{Q} \right)$
  **for** $t \leftarrow l, \ldots, 0$ **do**
    Update $Q(\mathbf{h}_t^{(n)}) \leftarrow \max_{j=1}^{n} \left( \frac{\sum_{i=l}^{t} Q(\mathbf{h}_i^{(j)})}{l - t + 1} \right)$ ▷ Back-propagation
    Add visit count: $N(\mathbf{h}_t^{(n)}) \leftarrow N(\mathbf{h}_t^{(n)}) + 1$
  **end for**
**end for**

## D. Dataset Details

Table 6 provides an overview of the dataset statistics used in the KBQA-o1 experiments across different settings: I.I.D (Independent and Identically Distributed), Compositional, Zero-shot, and three datasets (GrailQA, WebQSP, and GraphQ). The table includes the following rows: **#Train** is the number of training examples used for each dataset. **#Exploration** is the number of exploration queries generated during the heuristic exploration phase for each dataset. The numbers indicate the extensive exploration. **#Test** is the number of testing examples for evaluation in each setting.

*Table 6.* Dataset statistics of KBQA-o1.

| | I.I.D | Compositional | Zero-shot | GrailQA | WebQSP | GraphQ |
|---|---|---|---|---|---|---|
| **#Train** | - | - | - | 40 | 100 | 100 |
| **#Exploration** | - | - | - | 43851 | 2929 | 2332 |
| **#Test** | 1564 | 1487 | 3645 | 6696 | 1566 | 2319 |

**GrailQA dataset** (Gu et al., 2021) is a large-scale dataset specifically designed to evaluate KBQA models across three key generalization levels: i.i.d., compositional, and zero-shot. It contains 64,331 questions, of which 6,696 in the local dev set with three levels are used for testing in the KBQA-o1 experiments. This dataset requires models to understand a wide range of logical forms and multi-hop reasoning over knowledge bases. Its exploration phase involves a significant number of queries (43,851), emphasizing the importance of comprehensive environment exploration. The dataset is particularly challenging in its compositional and zero-shot settings, where models need to handle unseen combinations of entities and relations, testing their ability to generalize beyond simple patterns.

**WebQSP dataset** (Yih et al., 2016) is an enriched version of the original WebQuestions dataset, providing semantic parses for each of its 4,737 questions. In KBQA-o1 experiments, 1,566 questions are used for testing, and 2,929 queries are generated during the exploration phase. This dataset focuses on evaluating the semantic understanding of KBQA models, requiring them to map natural language questions to precise logical forms. The inclusion of semantic parses enhances the dataset's utility for training and evaluating models, enabling fine-grained analysis of their ability to disambiguate and retrieve information from the knowledge base. The relatively smaller size of WebQSP compared to GrailQA makes it a valuable benchmark for assessing model efficiency in low-resource settings.

**GraphQ dataset** (Su et al., 2016) is a dataset aimed at testing KBQA models on complex reasoning tasks involving graph-structured data. It contains over 5,000 questions, with 2,319 used for testing in the KBQA-o1 experiments. The exploration phase generates 2,332 queries, reflecting the intricate nature of reasoning required by the dataset. GraphQ challenges models to navigate multi-hop relationships and resolve complex dependencies between entities, pushing them to understand the underlying structure of the knowledge graph. Its focus on characteristic-rich queries ensures a thorough evaluation of a model's reasoning capabilities, making it an essential dataset for advancing KBQA methods.

## E. Atomic Query Tool Details

As shown in Table 1, KBQA-o1 introduces eight atomic query tools designed to facilitate logical form generation for KBQA. These tools are tailored to systematically convert natural language queries into logical forms by leveraging the structural properties of KB. Each tool serves a specific function, ensuring precise interactions with the KB to retrieve or manipulate information. Below is an explanation of each tool:

**Extract_entity** is used to identify and extract a specific entity from the knowledge base. It takes an entity as an argument and initializes the logical form with this entity. The target function is represented as START('entity'), and the corresponding logical form is simply the entity itself. For instance, extracting "Taylor Lautner" initializes the logical form with the entity identifier for the actor.

**Find_relation** identifies a relation connected to the current logical form. It takes a relation as an argument and appends it to the existing logical form using the JOIN operation. The resulting logical form is (JOIN relation (expression)). For example, finding the film.actor.film relation connects an actor to their associated films.

**Merge** combines two logical expressions into a single conjunctive expression. It takes two arguments, expression1 and expression, and returns AND(expression1, expression) as the target function. The equivalent logical form is (AND

(expression1) (expression)). For example, merging two conditions like "actor is Taylor Lautner" and "runtime $<$ 60 minutes" forms a combined condition.

**Order** determines the extreme value of a property, such as the maximum or minimum. It takes a mode (e.g., max or min) and a relation as arguments. The target function is ARG('mode', expression, 'relation'), and the logical form is (mode (expression) relation). For example, finding the longest film associated with an actor uses this tool.

**Compare** evaluates a property against a specified value using a comparison operator (e.g., $<$, $<=$, $>$, $>=$). It takes a mode and a relation as arguments. The target function is CMP('mode', 'relation', expression), and the corresponding logical form is (mode relation (expression)). For instance, checking if a film's runtime is less than 60 minutes employs this tool.

**Time_constraint** imposes a time-based constraint on the logical form. It takes a relation and a time as arguments, applying a temporal filter to the expression. The target function is TC(expression, 'relation', 'time'), and the logical form is (TC (expression) relation time). For example, filtering films released before a specific year uses this tool.

**Count** calculates the number of entities that satisfy a given condition. It takes an expression as an argument, returning the target function COUNT(expression) and the logical form (COUNT (expression)). For instance, counting the number of films an actor has appeared in utilizes this tool.

**Finish** signals the termination of the logical form generation process. It takes an expression as its argument and finalizes the logical form as STOP(expression). The corresponding logical form is simply (expression). This tool ensures the logical form is complete and ready for execution against the KB.

## F. Baseline Details

The six full-resource baselines provide a comprehensive evaluation framework for KBQA. These methods leverage fully annotated datasets and advanced reasoning mechanisms to achieve high performance on various KBQA tasks. Below is a summary of each baseline:

**RnG-KBQA** (Ye et al., 2022) is a retrieve-and-generate framework that first retrieves relevant knowledge from the knowledge base (KB) and then generates executable logical forms for answering questions. It focuses on leveraging retrieval to enhance logical form generation accuracy.

**DecAF** (Yu et al., 2023) employs multi-granular retrieval strategies to ensure robust KBQA performance. By focusing on progressively refining retrieved knowledge, DecAF addresses the complexity of large-scale KBs and supports more accurate logical reasoning.

**TIARA** (Shu et al., 2022) is a multi-stage retrieval method designed for large-scale KBs. It enhances robustness by retrieving candidate knowledge in multiple stages and integrating it into logical form generation, improving its ability to handle complex queries.

**SPARQA** (Sun et al., 2020) uses a skeleton-based semantic parsing approach to generate logical forms for complex questions. It simplifies question processing by extracting a structural skeleton, which is then converted into a complete logical form.

**BERT+Ranking** (Gu et al., 2021) integrates a BERT-based encoder with a ranking mechanism to select the most relevant answers from candidate entities. It enhances the precision of entity linking and relation matching in complex KBQA scenarios.

**ArcaneQA** (Gu & Su, 2022) combines dynamic program generation with contextualized encoding to address complex reasoning tasks. Its innovative design enables it to process multi-hop reasoning queries with high accuracy.

The three low-resource baselines are designed to address the challenges of KBQA in scenarios. Unlike fully supervised methods, these approaches focus on GPT API for in-context-learning. Below is a detailed summary of each method:

**KB-BINDER** (Li et al., 2023) leverages GPT-3.5-turbo to perform KBQA with minimal supervision. It adopts a structured in-context learning approach, where logical form templates guide the generation process. This method effectively balances efficiency and accuracy in low-resource settings.

**KB-Coder** (Nie et al., 2024) employs code-style in-context learning to improve logical form generation. By treating logical form generation as a code-writing task, this method enhances reasoning consistency and adaptability, even with limited training data.

**ARG-KBQA** (Tian et al., 2024) enhances knowledge retrieval efficiency in low-resource KBQA. It optimizes argument selection for logical forms, ensuring that queries retrieve the most relevant knowledge while minimizing errors caused by insufficient training data.

The four baselines in Section 5.4 represent different approaches to KBQA, covering both end-to-end and step-by-step methods. These methods are designed to evaluate different strategies, including retrieval-based logical form generation, sequential reasoning, and tree-based expansion. The comparison with KBQA-o1 highlights the strengths and limitations of each approach, providing insight into the effectiveness of heuristic exploration with MCTS. Below is a summary of each baseline:

**RG-E2E**: Based on the DecAF (Yu et al., 2023), RG-E2E follows a retrieve-then-generate paradigm, where relevant knowledge is first retrieved from the KB before generating the logical form. While effective in structured environments, this approach struggles with unseen entity-relation combinations due to its reliance on pre-retrieved data.

**GR-E2E**: Adapted from the ChatKBQA (Luo et al., 2024a), GR-E2E first generates a preliminary logical form and then refines it by retrieving supporting evidence from the KB. While more flexible than RG-E2E, it faces challenges in ensuring that the generated logical form aligns correctly with KB constraints.

**CoT-SbS**: Inspired by QueryAgent (Huang et al., 2024), CoT-SbS applies Chain-of-Thought (CoT) reasoning to KBQA, where logical forms are constructed incrementally through multiple reasoning steps. Although it improves interpretability, it is prone to local optima and reasoning errors, especially in complex multi-hop queries.

**ToT-SbS**: Based on the Think-on-Graph (Sun et al., 2024), ToT-SbS extends step-by-step reasoning into a tree-like structure, expanding multiple reasoning paths simultaneously. While it mitigates local optima, it suffers from large search spaces, making it computationally expensive compared to heuristic search methods.

## G. Hyperparameter Settings

Table 7 presents the hyperparameter configurations for the KBQA-o1 across three datasets: GrailQA, WebQSP, and GraphQ. These parameters are categorized into four stages: Initial Few-shot SFT, MCTS Exploration Stage, Incremental Fine-tuning, and MCTS Prediction Stage, each designed to optimize the KBQA framework's performance for different tasks.

In the Initial Few-shot SFT stage, the policy and reward models are fine-tuned using a small labeled dataset to initialize the framework. Both models adopt the DoRA architecture, optimized for reasoning tasks. The batch size is set to 4, ensuring stability in training with manageable memory usage. A fixed learning rate of 5e-5 controls weight updates, providing a balance between convergence speed and training stability. The number of epochs varies by dataset, with 100 for the policy model in GrailQA and 50 for WebQSP and GraphQ. The reward model undergoes more training, with 300 epochs for GrailQA and 100 for WebQSP and GraphQ, to ensure the robustness of the reward function.

The MCTS Exploration Stage employs Monte Carlo Tree Search to explore logical forms by simulating reasoning paths within the KB. Each rollout performs 6 iterations of exploration. A beam size of 2 limits the number of candidate paths considered. TopK and TopD parameters further refine candidate selection, with GrailQA and GraphQ using TopK=10, while WebQSP uses TopK=3. Similarly, GrailQA and WebQSP have TopD=3, whereas GraphQ sets TopD=5. An exploration weight of 50 balances exploration of new paths with exploitation of known high-quality paths. The reward ratio, fixed at 0.5, ensures equal contribution from the policy and reward models. Dataset-specific reward thresholds (-100 for GrailQA, 30 for WebQSP, and -50 for GraphQ) filter out low-quality paths, tailoring the exploration process to the characteristics of each dataset.

The Incremental Fine-tuning stage further refines the policy and reward models based on insights gained during the exploration stage. The models retain the DoRA architecture, with a batch size of 4 and a learning rate of 5e-5. Both models are fine-tuned for 10 epochs across all datasets, ensuring improved generalization.

In the MCTS Prediction Stage, the refined models are used to generate final logical forms for KBQA tasks. 6 rollouts are performed, consistent with the exploration stage, but the beam size is reduced to 1 to focus on the most promising candidate paths. TopK and TopD parameters mirror those in the exploration stage to maintain consistency in candidate selection. The exploration weight is reduced to 10, prioritizing exploitation of high-quality paths while allowing limited exploration during prediction. The reward ratio remains fixed at 0.5 across all datasets, maintaining a consistent balance between contributions from the policy and reward models.

*Table 7.* HyperParameter Settings for GrailQA, WebQSP, and GraphQ

| Hyperparameter Name | GrailQA | WebQSP | GraphQ |
|---|---|---|---|
| *Initial Few-shot SFT* | | | |
| SFT1 Type for Policy Model | DoRA | DoRA | DoRA |
| SFT1 Batch Size for Policy Model | 4 | 4 | 4 |
| SFT1 Learning Rate for Policy Model | 5e-5 | 5e-5 | 5e-5 |
| SFT1 Epoch for Policy Model | 100 | 50 | 50 |
| SFT1 Type for Reward Model | DoRA | DoRA | DoRA |
| SFT1 Batch Size for Reward Model | 4 | 4 | 4 |
| SFT1 Learning Rate for Reward Model | 5e-5 | 5e-5 | 5e-5 |
| SFT1 Epoch for Reward Model | 300 | 100 | 100 |
| *MCTS Exploration Stage* | | | |
| MCTS1 Rollout $N$ | 6 | 6 | 6 |
| MCTS1 Beam Size $B$ | 2 | 2 | 2 |
| MCTS1 TopK $k$ | 10 | 3 | 10 |
| MCTS1 TopD $d$ | 3 | 3 | 5 |
| MCTS1 Exploration Weight $w$ | 50 | 50 | 50 |
| MCTS1 Reward Ratio $\delta$ | 0.5 | 0.5 | 0.5 |
| MCTS1 Reward Threshold $\gamma^*$ | -100 | 30 | -50 |
| *Incremental Fine-tuning* | | | |
| SFT2 Type for Policy Model | DoRA | DoRA | DoRA |
| SFT2 Batch Size for Policy Model | 4 | 4 | 4 |
| SFT2 Learning Rate for Policy Model | 5e-5 | 5e-5 | 5e-5 |
| SFT2 Epoch for Policy Model | 10 | 10 | 10 |
| SFT2 Type for Reward Model | DoRA | DoRA | DoRA |
| SFT2 Batch Size for Reward Model | 4 | 4 | 4 |
| SFT2 Learning Rate for Reward Model | 5e-5 | 5e-5 | 5e-5 |
| SFT2 Epoch for Reward Model | 20 | 20 | 20 |
| *MCTS Prediction Stage* | | | |
| MCTS2 Rollout $N$ | 6 | 6 | 6 |
| MCTS2 Beam Size $B$ | 1 | 1 | 1 |
| MCTS2 TopK $k$ | 10 | 3 | 10 |
| MCTS2 TopD $d$ | 3 | 3 | 5 |
| MCTS2 Exploration Weight $w$ | 10 | 10 | 10 |
| MCTS2 Reward Ratio $\delta$ | 0.5 | 0.5 | 0.5 |

## H. Open-source LLMs used in KBQA-o1

The KBQA-o1 framework is designed with a plug-and-play architecture, allowing seamless integration of different open-source LLMs based on task requirements. This flexibility allows easy substitution or upgrading of LLMs, ensuring that the system remains state-of-the-art as newer models become available.

**Llama-3** (Dubey et al., 2024), developed by Meta AI, is an advanced large language model known for its strong reasoning abilities and generalization across diverse tasks. It excels in processing multi-hop reasoning and compositional queries, making it ideal for KBQA tasks that require logical inference over structured data, ensuring robustness in complex question-answering scenarios.

**Qwen2.5** (Yang et al., 2025), released by Alibaba, is particularly effective in entity linking and relation extraction, two crucial components of KBQA. Trained on both structured and unstructured data, Qwen2.5 ensures high-precision entity disambiguation and improves the accuracy of knowledge retrieval. Its multilingual capabilities make it promising to further enhance KBQA-o1's adaptability across different datasets and domains.

**Gemma-2** (Team et al., 2024), developed by Google, is optimized for retrieval-based reasoning and structured query generation. It specializes in efficiently retrieving relevant knowledge and generating accurate logical forms for complex queries. Its lightweight yet powerful design ensures high-speed inference, making it well-suited for low-latency applications in KBQA.

# I. Case Study

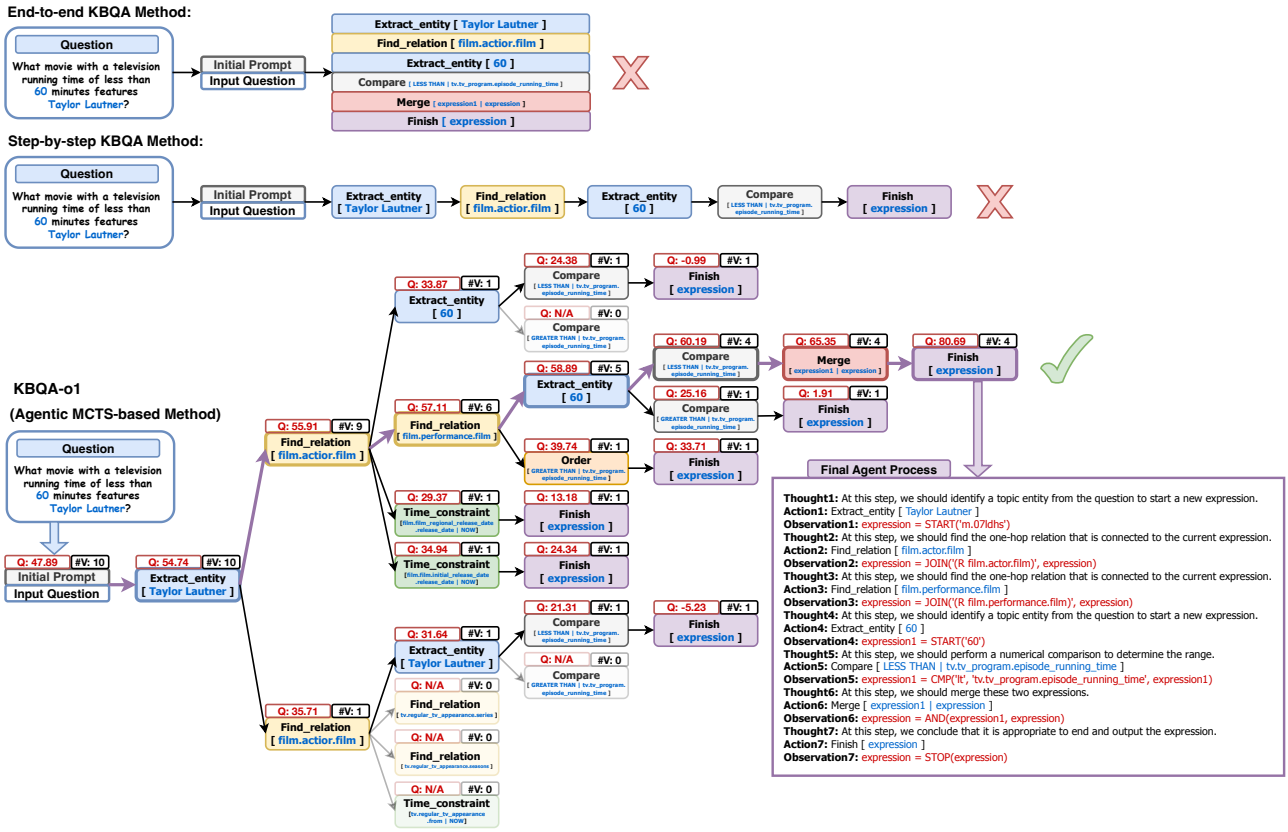



*Figure 9.* Case study of KBQA-o1's MCTS inference compared with end-to-end and step-by-step methods.

To visually illustrate the reasoning process of KBQA-o1, Figure 9 compares the proposed MCTS-based method with end-to-end and step-by-step KBQA approaches. In this example, the question "What movie with a television running time of less than 60 minutes features Taylor Lautner?" is posed. The question is first processed in the initial prompt, forming the agent's starting state. Through multiple rounds of MCTS exploration, the framework builds a search tree, where the Q-value represents the score updated through backpropagation, and #V denotes the number of times each state has been visited. These values guide the agent in selecting the optimal next node using the Upper Confidence Bound for Trees (UCT).

The end-to-end KBQA method prematurely terminates logical form generation due to incomplete reasoning and a lack of exploration. The step-by-step KBQA method incrementally constructs logical forms but also fails, as it cannot fully navigate the reasoning constraints, leading to an incorrect result. In contrast, the KBQA-o1 method effectively explores and evaluates multiple reasoning paths, refining its decisions based on reward feedback and generating the correct logical form. The final agent process demonstrates how observations at each step are transformed into graph query statements, which are executed to obtain the correct answer. This process highlights KBQA-o1's ability to systematically decompose the query, balance exploration and exploitation, and handle multi-step reasoning effectively, outperforming other methods.

# J. Error Analysis

We analyze the errors made by KBQA-o1 on the GrailQA dataset and categorize them into three main types. These errors reflect the challenges in exploring logical paths, retrieving accurate results, and ranking the most appropriate paths in the context of complex KBQA tasks.

**Executable path not discovered (29.8%)** In this category, KBQA-o1 fails to explore any executable logical path in the knowledge base, often due to the complexity of GrailQA's compositional and zero-shot queries. These errors arise from insufficient exploration of multi-hop reasoning paths and limited representation of rare logical forms in the training data.

Improvements in exploration depth and diversity of logical forms in the training process could help mitigate this issue.

**Correct path not discovered (54.7%)** This is the most common error type, where KBQA-o1 explores logical paths but fails to discover the correct one. These errors highlight challenges in precise entity and relation retrieval, particularly in zero-shot settings where entities or relations are unseen during training. Enhancing the semantic understanding and retrieval mechanisms would be crucial for reducing this type of error.

**Correct path not selected as the best (15.5%)** In some cases, KBQA-o1 successfully explores the correct logical path but fails to select it as the optimal solution. For example, it may assign a higher Q-value to an incorrect logical form over the correct one. These errors often stem from suboptimal reward evaluation during MCTS backpropagation or inconsistencies in the reward model's fine-tuning. Addressing these issues through better reward modeling and fine-tuning could significantly enhance path selection accuracy.

## K. Future Directions

The KBQA-o1 offers several promising directions for future research and development. These directions aim to enhance the model's capabilities, extend its application domains, and address current limitations.

**Exploring Reinforcement Learning Techniques Like DPO for Continual Learning.** One promising direction is leveraging advanced reinforcement learning techniques, such as Direct Policy Optimization (DPO), to enable continual learning for KBQA-o1. By refining policy and reward models through reinforcement learning, the framework could dynamically adapt to new datasets and tasks. A key challenge in this direction is constructing high-quality positive and negative sample pairs to guide the learning process effectively. Future work could explore automated methods for generating these samples or employing human-in-the-loop systems for quality assurance.

**Expanding the Set of Logical Operators for Enriched QA Patterns.** To broaden the range of questions the system can handle, future research could focus on introducing and supporting more diverse logical operators. This would allow KBQA-o1 to answer more complex queries, particularly those requiring advanced reasoning patterns such as conditional logic, aggregation, and temporal constraints. By integrating these operators, the framework could offer more comprehensive and flexible question-answering capabilities, covering a wider array of user needs.

**Specialized Applications in Domains like Medicine and Law.** KBQA-o1 has significant potential for domain-specific applications, particularly in fields such as medicine and law. These areas require precise reasoning and domain-specific knowledge to handle intricate and high-stakes queries. Adapting the framework to these domains would involve integrating specialized knowledge bases and fine-tuning the models to align with domain-specific terminologies and reasoning patterns. Such advancements could pave the way for impactful real-world applications, including clinical decision support and legal research.

**Extending to Multimodal, Multilingual, and Multi-Agent Systems.** Future iterations of KBQA-o1 could extend its capabilities to multimodal and multilingual settings, enabling the framework to process and reason over diverse input formats such as images, audio, and text in multiple languages. Additionally, incorporating multi-agent systems could enhance the framework's ability to handle collaborative and interactive tasks, where multiple agents contribute to exploring and resolving complex queries. These advancements would significantly expand the framework's usability across various scenarios.