

PSR: Efficient Knowledge Graph Question Answering via Plan-Guided Schema-Level Reasoning

Mingyuan Li¹, Yucheng Li¹, Yuheng Bao¹, Yawei Sun², Gong Cheng¹

¹State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing, China

²Department of Computer Science and Technology, Tsinghua University, Beijing, China

Correspondence: gcheng@nju.edu.cn

Abstract

Retrieval-Augmented Generation (RAG) enhances Large Language Models (LLMs) in Knowledge Graph Question Answering (KGQA) by retrieving reasoning paths from external Knowledge Graphs (KGs). However, the practicality of existing methods is often limited by their reliance on iterative retrieval, where the number of required LLM calls increases with retrieval depth, leading to expensive interaction costs. To address this, we propose **Plan-guided Schema-level Reasoning (PSR)**, a training-free and efficient RAG framework for KGQA that requires only three LLM calls. First, PSR generates a retrieval plan from the input question (the first LLM call). It then performs Plan-Regularized Monte Carlo Tree Search to extract a compact set of schema-level paths from the KG, which are subsequently filtered by the LLM (the second LLM call). Finally, PSR grounds the chosen schema-level paths into instance-level reasoning paths to augment the LLM’s answer generation (the third LLM call). Experimental results on three datasets demonstrate the high efficiency and competitive accuracy of PSR.

1 Introduction

Retrieval-Augmented Generation (RAG) augments Large Language Models (LLMs) with external knowledge by retrieving information and conditioning generation on retrieved content (Fan et al., 2024; Li et al., 2025). For complex questions, retrieval is often multi-step (Saha et al., 2018; Sun et al., 2019). For example, consider the question: “*Lou Seal is the mascot for the team that last won the World Series when?*” Answering it requires retrieving multi-hop paths over a given knowledge graph (KG), e.g., `team_mascot` → `championships` → `date`, together with a max function that selects the latest date. This combination of multi-hop relation chaining and an aggregate function cannot be resolved in a single re-

Method	# LLM Calls
KBQA-o1 (Luo et al., 2025)	$1 + T(2RD + W)$
FastToG (Liang and Gu, 2025)	$(W + 1)D + 2$
FiDeLiS (Sui et al., 2025)	$D(W + 1) + 1$
ToG (Sun et al., 2024)	$2WD + D + 1$
PoG (Chen et al., 2024)	$1 + T(4D + p)$
DoG (Ma et al., 2025)	$D(a + 1) + 3(D - 1)$
ReKnoS (Wang et al., 2025)	$(2D + 1) + D_{\text{SR}}$
PSR (ours)	3

Table 1: LLM calls in existing methods (D : retrieval depth; W : beam width; T : search budget; R : rollouts per iteration; p : reflection triggers; D_{SR} : depth of super-relation induction; a : relation filtering calls).

trieval step (Sun et al., 2024; Chen et al., 2024).

Motivation. A common approach to multi-step retrieval on KGs treats the LLM as a step-wise agent: in each iteration, it selects relation paths, expands current paths by retrieving neighboring KG triples, and selects top-ranked paths for the next iteration of retrieval (Sun et al., 2024; Chen et al., 2024; Song et al., 2025; Wang et al., 2025). This paradigm faces two limitations on large KGs. First, complex questions challenge accurate entity-level search: as the retrieval depth increases, the number of reachable entities grows exponentially and introduces many irrelevant branches, which degrades retrieval accuracy. Second, because the LLM is invoked at each expansion step to guide step-wise retrieval (and even perform additional rollouts or reflections), the number of its calls increases substantially with retrieval depth, often incurring prohibitive inference latency and cost. In fact, as shown in Table 1, across representative methods, the LLM-call count grows with the retrieval depth D and typically also with other hyperparameters, rather than being a constant.

For practical use, one natural question would

be: *Can we solve multi-hop reasoning over KGs using a constant number of LLM calls that does not depend on retrieval depth, without compromising answer accuracy?*

Our Work. To answer this question, we propose *Plan-guided Schema-level Reasoning* (PSR), which adopts a *holistic plan-grounding* process to decouple LLM interaction from iterative retrieval. Instead of using the LLM to guide hop-by-hop path expansion, PSR uses the LLM-generated plan as a high-level semantic prior and performs training-free, fixed-budget schema-level search over a compact schema tree. The key distinction from prior work is that PSR does not rely on hop-by-hop LLM interaction or exhaustive schema-level path enumeration (Sun et al., 2024; Chen et al., 2024; Liu et al., 2025). This design keeps the number of LLM calls fixed at three, independent of retrieval depth.

Specifically, PSR implements this holistic plan-grounding process as a tri-stage framework, with one LLM call in each stage. Stage I invokes the LLM to generate high-level plans to guide path selection from the KG. Stage II performs Plan-Regularized Monte Carlo Tree Search (PR-MCTS), where the textual plan regularizes KG schema search to extract candidate schema-level paths from schema trees; a second LLM call then selects the most promising paths to be grounded in the KG. Stage III grounds the selected schema-level paths to retrieve KG triples and uses a third LLM call to generate the final answer from the retrieved triples. Experiments demonstrate that PSR achieves competitive accuracy with only three LLM calls for multi-hop KGQA.

The **main contributions** of this paper are summarized as follows:

- We propose **PSR**, a tri-stage framework that efficiently and effectively answers each multi-hop KG question with exactly three LLM calls, independent of retrieval depth.
- As a core component of PSR, we introduce **PR-MCTS** that enables LLM-free schema-level path retrieval by integrating a plan-alignment prior with branching control, yielding a small set of relevant schema-level paths for downstream LLM-based selection.

Our code is on GitHub.¹

¹<https://github.com/nju-websoft/PSR>

2 Related Work

In this section, we review existing KGQA methods that are technically related to PSR.

KGQA as Reasoning Path Retrieval. Our method generally follows the paradigm of formulating KGQA as reasoning path retrieval, i.e., first retrieving a set of paths starting from topic entities in the KG, then utilizing these paths to generate answers. Early works in this paradigm (Jiang et al., 2023; Sun et al., 2024) adopt an iterative strategy. They construct reasoning paths starting from topic entities step-by-step, utilizing LLMs to select the next relation types and entities to expand. While this approach effectively utilizes the reasoning capability of LLMs, it has the following limitations: 1) Performance is hindered by the scale of KGs, particularly for complex questions, because at each hop the retriever must consider many outgoing relations and entities, most of which are irrelevant to the question, so the correct multi-hop path is prone to be missed during pruning. 2) Since the LLM must decide for each expansion step, the number of LLM calls increases with retrieval depth.

Subsequent works have adopted various improvements to mitigate these limitations, which can be categorized into two directions:

Planning as Guidance. A series of works incorporate a planning phase before constructing reasoning paths. The plan outlines the potential structure of the paths, serving as guidance for subsequent construction. For instance, RoG (Luo et al., 2024) first generates the sequence of relations within a reasoning path, then filling in the entities; PoG (Chen et al., 2024) decomposes the question into sub-objectives, which are then addressed sequentially during the construction. FiDeLiS (Sui et al., 2025) generates a plan and then follows it to perform hop-by-hop beam search in the KG; at each hop, it invokes the LLM to select top candidate paths and uses deductive verification to decide whether to stop or continue extending the paths. Although these methods improve retrieval guidance, they still couple reasoning with retrieval depth through iterative LLM-in-the-loop expansion. In contrast, *PSR uses the LLM-generated plan only as a high-level semantic prior and performs schema-level search and ranking with PR-MCTS without hop-by-hop LLM interaction*. This design *decouples reasoning from retrieval depth*, keeping the number of LLM calls fixed regardless of retrieval depth.

Search Space Compression. Recognizing the dif-

difficulty of navigating massive KGs, a line of research focuses on compressing the search space, where common techniques include the extraction of question-relevant subgraphs (Mavromatis and Karypis, 2025; Long et al., 2025; Luo et al., 2024; Jiang et al., 2025), clustering of neighborhood entities (Liang and Gu, 2025) or relations (Wang et al., 2025). Most recently, ORT (Liu et al., 2025) introduces KG schema as a compressed search space, employing Depth-First Search (DFS) to enumerate schema-level paths that connect the topic entity type to the answer type in the schema graph, and then ground these schema-level paths to obtain concrete reasoning paths.

While ORT shares our insight that KG schema can serve as an effective compressed search space, its search still relies on exhaustive DFS enumeration followed by heuristic pruning. In contrast, PSR integrates planning guidance with schema-level compression: instead of exhaustively enumerating schema-level paths or relying on strict rule matching, *PSR introduces PR-MCTS to transform the LLM-generated high-level textual plan into an LLM-free and fixed-budget schema-level search heuristic*. This allows PSR to prune the schema tree more effectively while preserving semantic flexibility. Moreover, unlike prior MCTS-based KGQA methods (Song et al., 2025; Xiong et al., 2025), which typically require training a value model, *PR-MCTS is entirely training-free and scores schema-level paths by their semantic alignment with the textual plan, rather than requiring exact rule or pattern satisfaction*.

3 Method

3.1 Preliminaries

Task Formulation. We formulate KGQA as reasoning over a KG $\mathcal{G} = (\mathcal{V}, \mathcal{R}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{R}$ denote entities and relation types respectively, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{R} \times \mathcal{V}$ represents the set of observed factual triples. Given a question q and a set of topic entities $\mathcal{V}_q \subseteq \mathcal{V}$ (i.e., entities explicitly mentioned in q and identified by entity linking), the objective is to derive a final answer set $\mathcal{A} \subseteq \mathcal{V}$.

Reasoning Path Retrieval. Following existing LLM-based KGQA methods (Sun et al., 2024; Luo et al., 2024; Chen et al., 2024), we solve KGQA as a path retrieval problem. Specifically, we divide the process into two steps: First, a set of relevant *reasoning paths* $\mathcal{P}$ is retrieved. A k -hop reasoning path is defined as a

sequence $(e_0, r_1, e_1, \dots, r_k, e_k)$, where $e_0 \in \mathcal{V}_q$ and $(e_{i-1}, r_i, e_i) \in \mathcal{E}$ for all $1 \leq i \leq k$, serving as a structural grounding of the semantic constraints derived from q . Then, an answer set $\mathcal{A}$ is obtained by reasoning over the retrieved paths $\mathcal{P}$ given the question q .

3.2 Method Overview

To address multi-hop KGQA efficiently, we propose Plan-guided Schema-level Reasoning (PSR). As illustrated in Figure 1, PSR operates in three stages, requiring only three LLM calls. **Plan Generation** (Stage I) and **Schema-Level Path Ranking** (Stage II) perform plan-guided schema navigation to identify relevant path structures. **Path Grounding and Answer Generation** (Stage III) grounds these structures into instance-level reasoning paths, and accordingly generates the answer.

- **Stage I: Plan Generation.** We invoke the LLM to generate a plan set Π given q (**the 1st LLM call**). Each plan $\pi \in \Pi$ specifies a selected topic entity type c_{topic} , a sequence of LLM-generated relation labels $(r'_1, \dots, r'_L)$, and an LLM-predicted answer type label c'_{ans} . Note that labels are KG-agnostic texts.
- **Stage II: Schema-Level Path Ranking.** For each topic entity type c_{topic} , we construct a schema tree $\mathcal{T}$ rooted in this type by generalizing $\mathcal{G}$ from factual triples to type-level relations. We perform Plan-Regularized Monte Carlo Tree Search (PR-MCTS) to extract a schema-level candidate path set from $\mathcal{T}$. We rank these schema-level paths to obtain the top- K paths $\mathcal{P}_K^{\mathcal{T}}$, and invoke the LLM to select the most relevant paths $\mathcal{P}_s \subseteq \bigcup_{\pi \in \Pi} \mathcal{P}_K^{\mathcal{T}}$ (**the 2nd LLM call**).
- **Stage III: Path Grounding and Answer Generation.** We ground schema-level paths $\mathcal{P}_s$ back to instance-level reasoning paths $\mathcal{P}$ in $\mathcal{G}$; the set of all their constituent triples is denoted by $\mathcal{F}$. We invoke the LLM to generate the final answers $\mathcal{A}$ given q and $\mathcal{F}$ (**the 3rd LLM call**).

The entire process is outlined as an algorithm in Appendix A.4, with PR-MCTS detailed in Algorithm 2. The specific prompts used for these three LLM calls are detailed in Appendix A.10.

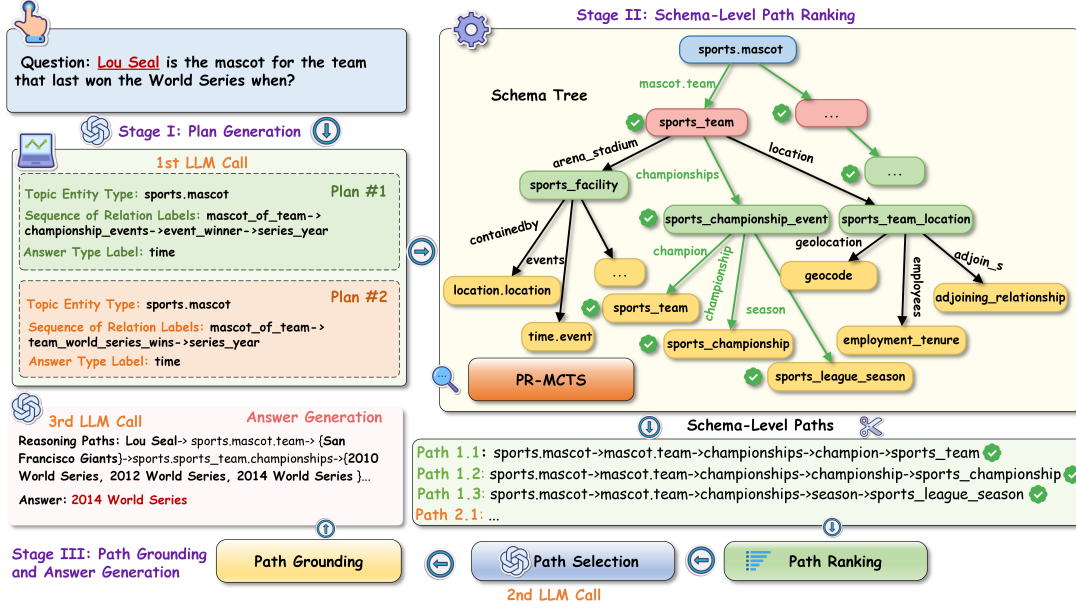


Figure 1: Overview of PSR. The 1st, 2nd, and 3rd LLM call denote plan generation, schema-level path selection, and answer generation, respectively. PR-MCTS, path ranking, and path grounding are LLM-free KG operations.

3.3 Stage I: Plan Generation

PSR performs plan generation under an entity type constraint derived from the topic entities. Given a question q and topic entities $\mathcal{V}_q$, we query $\mathcal{G}$ to extract the type set of each $e \in \mathcal{V}_q$. Let $\mathcal{C}$ denote the set of all entity types in $\mathcal{G}$, and let $\text{Types}(e) : \mathcal{V} \rightarrow 2^{\mathcal{C}}$ denote the types of entity e . To distinguish LLM-generated labels from identifiers in $\mathcal{G}$, we use $\mathcal{C}'$ to denote the set of LLM-generated entity type labels and $\mathcal{R}'$ to denote the set of LLM-generated relation labels. Let $\mathbf{x}_q := (q, \mathcal{V}_q, \{\text{Types}(e) : e \in \mathcal{V}_q\})$ denote the planning context. We invoke the LLM once to generate a plan set (**the 1st LLM call**):

$$\Pi \triangleq \text{LLM}(\mathbf{x}_q) = \bigcup_{e \in \mathcal{V}_q} \Pi(e) \quad (1)$$

where $\Pi(e)$ denotes the subset of plans generated for topic entity e . Each *plan* π is defined as:

$$\pi = (c_{\text{topic}}, (r'_1, \dots, r'_L), c'_{\text{ans}}) \quad (2)$$

where $c_{\text{topic}} \in \text{Types}(e) \subseteq \mathcal{C}$ for some $e \in \mathcal{V}_q$ is a topic entity type, each $r'_i \in \mathcal{R}'$ is an LLM-generated relation label and $(r'_1, \dots, r'_L)$ forms an LLM-generated relation label path of length L , and $c'_{\text{ans}} \in \mathcal{C}'$ is an LLM-generated answer type label. For the example in Section 1, the generated relation label path may be *team mascot* → *championships* → *date*, with a date-related answer type label.

3.4 Stage II: Schema-Level Path Ranking

In $\mathcal{G}$, the k -hop neighborhood of an entity expands exponentially with k . The schema graph $\mathcal{S}$ generalizes this structure into a compact form at the schema level, where nodes are entity types $\mathcal{C}$ and edges are relation types $\mathcal{R}$ in $\mathcal{G}$. We then unroll $\mathcal{S}$ into a schema tree $\mathcal{T}$ for schema-level search.

Schema Tree Construction. Guided by a plan π , PSR shifts the search space from $\mathcal{G}$ to its schema graph $\mathcal{S}$ and unrolls a directed schema tree $\mathcal{T}$ rooted at c_{topic} for subsequent schema-level search. We construct the *schema graph* as $\mathcal{S} = (\mathcal{C}, \mathcal{R}, \mathcal{E}_{\mathcal{S}})$. The type-level triples $\mathcal{E}_{\mathcal{S}}$ generalize instance-level factual triples in $\mathcal{G}$:

$$\mathcal{E}_{\mathcal{S}} = \{(c_h, r, c_t) \mid \exists (u, r, v) \in \mathcal{E} \text{ s.t. } c_h \in \text{Types}(u), c_t \in \text{Types}(v)\} \quad (3)$$

For a plan π , we unroll $\mathcal{S}$ into a directed schema tree $\mathcal{T}$ rooted at the plan-implied topic entity type c_{topic} , up to depth k , with edges from $\mathcal{E}_{\mathcal{S}}$.

Plan-Regularized Monte Carlo Tree Search. To search over the schema tree $\mathcal{T}$ without further LLM calls, we propose *Plan-Regularized Monte Carlo Tree Search* (PR-MCTS). Unlike MCTS-based KGQA methods (Song et al., 2025), which estimates returns from environment rollouts and expands over all available actions, PR-MCTS is tailored for an LLM-free and budget-constrained setting.

- *Plan as Semantic Prior.* The plan π pro-

vides a sequence of generated relation labels $r'_1, \dots, r'_L$ and an answer type label c'_{ans} . We inject this guidance into the search as a training-free soft semantic prior. This allows us to prioritize relation types that are semantically similar to $r'_1, \dots, r'_L$ without strictly requiring the LLM-generated labels to perfectly match the underlying KG schema and to use c'_{ans} for rollout-free evaluation of the terminal state.

- *Structure-Based Branching Control.* We employ an out-degree penalty as a structural regularizer to reduce branching in search.

To instantiate the four steps of a standard MCTS loop (*Selection, Expansion, Evaluation, and Back-propagation*), in the following, we formalize the search process on $\mathcal{T}$. A search state at depth d corresponds to a schema-tree node $c_d \in \mathcal{C}$. From a given state c_d , an action $a = (c_t, r) \in \text{Succ}(c_d)$, follows an edge in $\mathcal{T}$ to transition the state from c_d to c_t along r , where:

$$\text{Succ}(c_d) = \{(c_t, r) \mid (c_d, r, c_t) \in \mathcal{E}_S\} \quad (4)$$

Finally, a state satisfies the terminal condition if it reaches the depth limit k or if it has no outgoing transition, i.e., $\text{Succ}(c_d) = \emptyset$.

During PR-MCTS, each simulation expands a sequence of transitions of length at most k on the schema tree $\mathcal{T}$. For each visited state-action pair (c_d, a) with $a = (c_t, r)$, we maintain two statistics: the visit count $N(c_d, a)$ and the cumulative return $W(c_d, a)$. Specifically, let B denote the total number of simulations. For the b -th simulation, let $a_d^{(b)}$ be the action selected at depth d , and let $G_d^{(b)} \in \mathbb{R}$ be the scalar return assigned to that selection by the evaluation step. We define:

$$N(c_d, a) = \sum_{b=1}^B \mathbb{I}[a_d^{(b)} = a] \quad (5)$$

$$W(c_d, a) = \sum_{b=1}^B \mathbb{I}[a_d^{(b)} = a] \cdot G_d^{(b)} \quad (6)$$

where $\mathbb{I}[\cdot]$ is the indicator function. We define the mean action value as:

$$Q(c_d, a) = \frac{W(c_d, a)}{N(c_d, a)} \quad (7)$$

which is initialized to 0 when $N(c_d, a) = 0$. Let $N(c_d) = \sum_a N(c_d, a)$ denote the total visit count of state c_d .

1) *Selection.* Starting from the root c_{topic} , PR-MCTS recursively descends the schema tree by selecting actions with a plan-regularized extension of Upper Confidence bounds applied to Trees (UCT) (Coquelin and Munos, 2007; Browne et al., 2012). The plan π serves as a semantic prior during search: it biases exploration toward relation types that are lexically similar to π , while discouraging transitions into types with large out-degrees which would increase the branching factor of subsequent expansions. At each visited state c_d , we define:

$$\text{Score}(c_d, a) = Q(c_d, a) + U(c_d, a; \pi) \quad (8)$$

and select:

$$a^* = \arg \max_{a \in \text{Succ}(c_d)} \text{Score}(c_d, a) \quad (9)$$

The exploration term is defined as:

$$U(c_d, a; \pi) = \lambda \sqrt{\frac{\ln(1 + N(c_d))}{1 + N(c_d, a)}} + \Delta(c_d, a; \pi) \quad (10)$$

where $\lambda > 0$ is the exploration constant and the *plan-induced adjustment* is:

$$\Delta(c_d, a; \pi) = \alpha \mathcal{M}(a, \pi) - \beta \mathcal{H}(a) \quad (11)$$

where $\alpha \geq 0$ weights the plan-alignment prior, and $\beta \geq 0$ controls the strength of structural penalization; For an action $a = (c_t, r)$, we define the *structural regularizer* as:

$$\mathcal{H}(a) = \log(1 + \deg_{\text{out}}(c_t)) \quad (12)$$

where $\deg_{\text{out}}(c_t)$ is the out-degree of c_t in the schema tree $\mathcal{T}$. The *plan-alignment* term measures the lexical proximity between $t(r)$ —the textual form of the candidate relation type r and the plan-implied relation labels $\{r'_1, \dots, r'_L\}$:

$$\mathcal{M}(a, \pi) = \max_{r' \in \{r'_1, \dots, r'_L\}} \text{LexSim}(t(r), r') \quad (13)$$

where $\text{LexSim}(\cdot, \cdot) \in [0, 1]$ is a normalized sequence matching string similarity.

2) *Expansion.* When the selected action a^* reaches a state c_d with at least one unvisited action (i.e., $\exists a \in \text{Succ}(c_d)$ such that $N(c_d, a) = 0$), instead of expanding all possible branches, PR-MCTS bounds the per-iteration cost by considering a subset of at most m uniformly sampled unvisited actions from $\text{Succ}(c_d)$. From this sampled subset, the algorithm instantiates exactly one new child node corresponding to the action that minimizes the structural penalty $\mathcal{H}(a)$. Since $\mathcal{H}(a)$ increases with the

out-degree of the target node, this rule *reduces the branching factor of future expansions* and improves search efficiency under a fixed simulation budget.

3) *Evaluation*. We evaluate a terminal state $c_\ell \in \mathcal{C}$ using the plan-implied answer type label $c'_{\text{ans}} \in \mathcal{C}'$ as a prior. Let $t(c)$ denote the textual form of an entity type $c \in \mathcal{C}$. We assign the terminal value based on the cosine similarity between text embeddings:

$$V(c_\ell) = \text{sim}(\text{enc}(t(c_\ell)), \text{enc}(c'_{\text{ans}})) \quad (14)$$

where $\text{sim}(\cdot, \cdot)$ is computed from a frozen text encoder $\text{enc}(\cdot)$.

4) *Backpropagation*. After evaluating a terminal state c_ℓ , PR-MCTS backpropagates the terminal value $V(c_\ell)$ along the selected sequence of transitions and updates the statistics for each visited state-action pair (c_d, a) . Specifically, for each visited (c_d, a) , we increase its visit count $N(c_d, a)$ by 1 and update its cumulative return by $W(c_d, a) \leftarrow W(c_d, a) + V(c_\ell)$; its mean action value $Q(c_d, a)$ is then updated according to Eq. (7).

PR-MCTS repeats the above steps for a fixed number of simulations B and returns a candidate schema-level path set $\mathcal{P}_{\text{mcts}}^\mathcal{T}$ by collecting all unique root-to-terminal schema-level paths encountered during search, ranked by their visit frequencies across simulations.

Path Ranking. For $\mathcal{P}_{\text{mcts}}^\mathcal{T}$, we compute their similarity scores $\text{SimScore}(\cdot)$ with the plan and retain the top K to form $\mathcal{P}_K^\mathcal{T}$. Specifically, for the plan π , we obtain its textual form $\tilde{p} = [t(c_{\text{topic}}); r'_1; \dots; r'_L; c'_{\text{ans}}]$. Correspondingly, for each retrieved candidate schema-level path $p = (c_{\text{topic}}, r_1, c_1, \dots, r_j, c_j) \in \mathcal{P}_{\text{mcts}}^\mathcal{T}$, we linearize it into $\bar{p} = [t(c_{\text{topic}}); t(r_1); \dots; t(r_j); t(c_j)]$. Using a frozen text encoder $\text{enc}(\cdot)$, we define:

$$\text{SimScore}(p) = \cos(\text{enc}(\tilde{p}), \text{enc}(\bar{p})) \quad (15)$$

However, vector similarity may assign similar scores to paths differing by a single relation. To handle plan ambiguity and better enforce fine-grained relation constraints, we invoke the LLM to select the most promising paths from $\bigcup_{\pi \in \Pi} \mathcal{P}_K^\mathcal{T}$ (**the 2nd LLM call**):

$$\mathcal{P}_s = \text{Select}_{\text{LLM}} \left(q, \bigcup_{\pi \in \Pi} \mathcal{P}_K^\mathcal{T} \right) \quad (16)$$

3.5 Stage III: Path Grounding and Answer Generation

For each schema-level path in $\mathcal{P}_s$, PSR grounds it in $\mathcal{G}$ via KG retrieval. We define a retrieval

operator $\text{Retrieve}(\mathcal{G}, \mathcal{V}_q, \mathcal{P}_s)$ that executes each sequence of relation types specified in $\mathcal{P}_s$ on $\mathcal{G}$ starting from the topic entities $\mathcal{V}_q$ to retrieve the matched instance-level reasoning paths. We then aggregate their constituent triples to form the evidence set $\mathcal{F}$ as:

$$\mathcal{F} \triangleq \text{Retrieve}(\mathcal{G}, \mathcal{V}_q, \mathcal{P}_s) \subseteq \mathcal{E} \quad (17)$$

Finally, we provide $\mathcal{F}$ to the LLM (**the 3rd LLM call**) to generate an answer set $\mathcal{A}$ conditioned on the retrieved triples:

$$\mathcal{A} = \text{LLM}(q, \mathcal{F}) \quad (18)$$

where the LLM is prompted to generate the final answer by reasoning over the triples in $\mathcal{F}$.

4 Experiments

4.1 Experimental Setup

Datasets and Evaluation Metrics. We evaluate the proposed PSR across three widely-used KGQA benchmarks: WebQSP (Yih et al., 2015), ComplexWebQuestions (CWQ) (Talmor and Berant, 2018), and GrailQA (Gu et al., 2021). Detailed statistics for these datasets are provided in Appendix A.3. Following prior work (Sun et al., 2024; Chen et al., 2024; Sui et al., 2025; Wang et al., 2025), we use Hits@1 and F1 Score as the primary performance metrics. Furthermore, to evaluate the efficiency of PSR, we measure interaction cost using three metrics: average LLM calls per question, output token count, and latency.

Baselines. We provide implementation details of PSR in Appendix A.1. We compare PSR with representative baselines spanning three categories: 1) *LLM-based QA without KG*, including IO Prompt, which directly prompts the LLM to answer the question, and CoT (Wei et al., 2022), which augments prompting with explicit chain-of-thought reasoning; 2) *Fine-tuned reasoning path retrieval*, where LLMs are trained using the training sets to perform reasoning path retrieval, including RoG (Luo et al., 2024), EPERM (Long et al., 2025), KBQA-o1 (Luo et al., 2025), and KG-Agent (Jiang et al., 2025); 3) *Few-shot/zero-shot reasoning path retrieval*, where LLMs are prompted to perform reasoning path retrieval. Such methods include ToG (Sun et al., 2024), PoG (Chen et al., 2024), DoG (Ma et al., 2025), ORT (Liu et al., 2025), FastToG (Liang and Gu, 2025), FiDeLiS (Sui et al., 2025), and ReKnoS (Wang et al., 2025). We provide a theoretical analysis of the LLM interaction complexity for the baselines in Appendix A.2.

Method	WebQSP			CWQ		
	Hits@1	F1	Calls	Hits@1	F1	Calls
<i>LLM-based QA without KG</i>						
IO Prompt	67.40	59.71	1	47.50	37.93	1
CoT	75.60	65.37	1	48.90	44.76	1
<i>Fine-tuned reasoning path retrieval</i>						
RoG	85.70	69.81	2	62.60	56.17	2
EPERM	88.80	–	–	66.20	–	–
KBQA-o1	–	–	28.80	–	–	–
KG-Agent	83.30	–	–	72.20	–	–
<i>Few-shot/zero-shot reasoning path retrieval</i>						
<i>Backbone: GPT-4o-mini</i>						
ReKnoS	83.80	–	7.20	66.80	–	10.20
FastToG	65.80	–	–	45.00	–	–
<i>Backbone: GPT-4o</i>						
ORT	87.67	–	–	65.43	58.69	–
FiDeLiS	81.17	78.32	10.70	65.33	64.32	15.20
<i>Backbone: GPT-4</i>						
ToG	82.60	75.79	15.90	67.60	60.20	22.60
PoG	87.30	–	9	75.00	–	13.30
DoG	91.00	–	30.90	56.00	–	58.10
ReKnoS	84.90	–	7.20	68.20	–	10.20
PSR w/ GPT-4	89.17	80.17	3	72.83	65.60	3
PSR vs. ToG	+6.57	+4.38	-12.90	+5.23	+5.40	-19.60
PSR vs. PoG	+1.87	–	-6.00	-2.17	–	-10.30
PSR vs. DoG	-1.83	–	-27.90	+16.83	–	-55.10
PSR vs. ReKnoS	+4.27	–	-4.20	+4.63	–	-7.20

Table 2: Main experimental results. The results of base-lines are reported by their original papers, where – indicates unreported results.

4.2 Main Experimental Results

Table 2 reports Hits@1, F1 Score and the average number of LLM calls on WebQSP and CWQ. For space reasons, the GrailQA results are reported in Appendix A.7, where PSR remains competitive with clear efficiency advantages. On WebQSP, PSR achieves 89.17% Hits@1, significantly outperforming other iterative retrieval methods such as ToG (82.6%). While DoG achieves a slightly higher score (91.0%), it relies on an expensive debate mechanism and suffers a considerable performance drop on the more complex CWQ dataset (falling to 56.0%). In contrast, PSR achieves 72.83% Hits@1 on CWQ, surpassing ToG and ReKnoS by substantial margins and is competitive with the complex planning-based method PoG (75.0%), by using significantly fewer computational resources. The answer-set F1 results show a consistent trend with Hits@1, indicating that the performance gains of PSR are not limited to exact-match accuracy but

Method	WebQSP			CWQ		
	Hits@1	Calls	Tokens	Hits@1	Calls	Tokens
<i>Backbone: Llama-3.1-70B-Instruct</i>						
ToG	68.30	7.80	1104.80	37.10	11.60	1781.50
PoG	51.90	10.10	412.60	42.30	16.40	566.80
PSR	86.10	3	265.65	68.90	3	377.84
PSR vs. ToG	+17.80	-4.80	-839.15	+31.80	-8.60	-1403.66
PSR vs. PoG	+34.20	-7.10	-146.95	+26.60	-13.40	-188.96
<i>Backbone: Qwen-2.5-72B-Instruct</i>						
ToG	79.00	7.03	792.60	44.20	10.20	1245.10
PoG	77.90	9.90	357.20	68.30	19.70	630.90
PSR	87.38	3	286.30	70.67	3	428.13
PSR vs. ToG	+8.38	-4.03	-506.30	+26.47	-7.20	-816.97
PSR vs. PoG	+9.48	-6.90	-70.90	+2.37	-16.70	-202.77

Table 3: Experimental results under open-source LLM backbones.

also extend to answer-set quality. Regarding efficiency, existing methods exhibit high variance in computational cost, with LLM calls increasing linearly with retrieval depth (e.g., DoG requires up to 58.1 calls per question on CWQ). In contrast, PSR maintains a strictly constant budget of 3 LLM calls per question regardless of question complexity. Overall, PSR achieves state-of-the-art accuracy while maintaining a constant and low interaction cost. Appendix A.6 presents inference latency comparisons, and Appendix A.8 reveals that PSR achieves higher retrieval quality.

We further analyze failed cases of PSR on CWQ and group them into three major categories: Constraint Missing (57%), Schema Error (40%), and Budget Limit (3%). Detailed analysis and representative cases are provided in Appendix A.9.

4.3 Experimental Results under Open-Source LLM Backbones

Table 3 further compares PSR with representative baselines, ToG and PoG,² under two open-source LLM backbones, Llama-3.1-70B-Instruct and Qwen-2.5-72B-Instruct. Across backbones and datasets, PSR consistently achieves the best Hits@1 while using only 3 LLM calls per question. With Llama-3.1-70B-Instruct, PSR reaches 86.1 Hits@1 on WebQSP and 68.9 on CWQ, outperforming ToG (68.3/37.1) and PoG (51.9/42.3), while reducing the number of LLM calls from 7.8–10.1 to 3 on WebQSP and from 11.6–16.4 to 3 on CWQ. In addition, PSR substantially reduces token use compared

²We tried but failed to reproduce DoG and ORT.

Method Variant	WebQSP		CWQ	
	Hits@1	Δ	Hits@1	Δ
PSR	87.38	–	70.67	–
w/o PR-MCTS (using Beam Search)	85.71	-1.67	66.40	-4.27
w/o Path Ranking	86.21	-1.17	66.18	-4.49
w/o LLM-Based Path Selection	83.23	-4.15	65.69	-4.98

Table 4: Ablation study.

to other iterative methods, e.g., from 1104.8 to 265.6 tokens on WebQSP and from 1781.5 to 377.8 tokens on CWQ compared to ToG under Llama-3.1-70B-Instruct. With Qwen-2.5-72B-Instruct, PSR attains 87.38 Hits@1 on WebQSP and 70.67 on CWQ with 3 calls, exceeding ToG (79.0/44.2) and PoG (77.9/68.3) while keeping token use below baselines (e.g., 286.3 vs. 792.6 tokens on WebQSP compared to ToG). *These results indicate that the superior performance of PSR is robust to backbone changes, delivering consistent performance gains while retaining efficiency.*

Appendix A.5 reports results under lighter LLM backbones, showing that PSR retains strong accuracy and efficiency.

4.4 Ablation Study

To validate component efficacy, we conduct an ablation study with Qwen-2.5-72B-Instruct (Table 4). Replacing PR-MCTS with Beam Search (setting both Beam width and depth to 3) reduces Hits@1 by 1.67% (WebQSP) and 4.27% (CWQ), suggesting that allocating search budget across a wide range of schema-level paths (PR-MCTS) is beneficial compared to committing to a few top-ranked ones (Beam Search) under the same retrieval depth. Removing Path Ranking by Eq. (15) reduces performance by 1.17% and 4.49%, indicating the usefulness of this component. The largest degradation is observed when removing LLM-Based Path Selection by Eq. (16) (4.15% and 4.98%), implying that after graph search and embedding-based ranking, the remaining top-ranked schema-level paths can still differ in fine-grained relation semantics from the query. *Overall, each component contributes measurably: removing PR-MCTS, Path Ranking, or LLM-Based Path Selection consistently reduces Hits@1 on both WebQSP and CWQ, and the full PSR achieves the best performance only when all three components are jointly used.*

4.5 Hyper-Parameter Analysis

Figure 2 studies two hyper-parameters of PSR under Qwen-2.5-72B-Instruct on CWQ: m , the max-

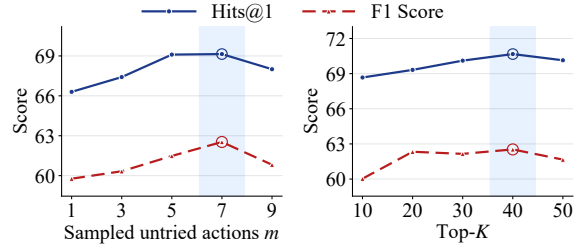


Figure 2: Hyper-parameter analysis.

Backbone	Method	EM	Hit@1	F1
DeepSeek-V4-Flash	CoT	2.10	15.40	7.73
	ToG	85.60	89.10	87.77
	PoG	92.40	93.10	93.67
	PSR	97.90	98.30	98.50
Qwen3.5-27B	CoT	2.10	19.80	10.32
	ToG	94.20	95.10	94.83
	PoG	89.50	90.80	90.86
	PSR	98.90	99.20	99.29

Table 5: Results on the proprietary KGQA dataset (%).

imum number of untried actions sampled from $\text{Succ}(c_d)$ in PR-MCTS, and K , the number of top-ranked schema-level paths kept for the second LLM call. As m increases, performance first improves and then declines: for $m \in \{1, 3, 5, 7, 9\}$, Hits@1/F1 rises from 66.30/59.77 at $m = 1$ to 70.67/62.53 at $m = 7$, then drops to 68.00/60.82 at $m = 9$. This suggests that overly large m disperses visit counts over more actions, reducing the reliability of $Q(c_d, a)$ and the concentration on relevant paths in $\mathcal{P}_{\text{mcts}}^T$. As K increases, performance improves and then saturates: for $K \in \{10, 20, 30, 40, 50\}$, it rises from 68.67/60.03 at $K = 10$ to 70.67/62.53 at $K = 40$, and remains stable for $K \in [30, 50]$ (Hits@1 70.11–70.67; F1 61.66–62.53), indicating that *PSR is relatively insensitive to K within this range*. Even at the worst setting of m , PSR still outperforms ToG on CWQ under the same backbone (66.30 vs. 44.2 Hits@1).

4.6 Evaluation on a Proprietary KGQA Dataset

We further evaluate PSR on a proprietary KGQA dataset provided by a company, constructed from a Traditional Chinese Medicine (TCM) KG covering, but not limited to, colorectal cancer. The dataset contains 1,000 automatically generated WebQSP-style single-topic-entity questions covering one- to three-hop reasoning. Each question is constructed from a reasoning path in the KG, and its gold an-

swer entities are retrieved by path execution rather than LLM generation.

As shown in Table 5, PSR consistently achieves the best performance under both LLM backbones. With DeepSeek-V4-Flash, PSR reaches 97.90% EM, 98.30% Hit@1, and 98.50% F1, while with Qwen3.5-72B it achieves 98.90%, 99.20%, and 99.29%, respectively. These results further demonstrate the effectiveness of PSR for multi-hop reasoning over a domain-specific KG.

5 Conclusion

In this paper, we propose PSR, a training-free RAG framework for KGQA that answers each question with exactly three LLM calls, independent of the retrieval depth. Instead of performing expensive hop-by-hop retrieval with the LLM, PSR employs plan-regularized MCTS on compact schema trees to efficiently identify relevant schema-level paths to ground. Experimental results demonstrate that PSR achieves state-of-the-art Hits@1 scores while maintaining consistent and low interaction costs. Ablation studies further validate that PR-MCTS, schema-level path ranking, and path selection are all essential for the final performance.

In future work, we will study how to reduce the sensitivity of schema-level search to noisy or missing type assignments when inducing the schema from a low-quality KG, and how to use grounding results to revise plan steps that fail as feedback signals to retrieve supporting KG triples. Our failure analysis further highlights plan completeness as a key direction for future improvement.

Limitations

PSR has two main limitations. First, PSR relies on entity type information to induce the schema and guide schema-level search. When entity type assignments in the KG are missing or incorrect, some schema-level paths corresponding to valid instance-level reasoning paths may be mistakenly excluded during search. More broadly, in real-world KGs such as ad hoc extracted graphs or user-generated KGs such as Wikidata, noise, incompleteness, or overly flat schema patterns may weaken schema induction and reduce the coverage or discriminative power of schema-level search, thereby limiting the robustness and applicability of PSR beyond relatively clean benchmark settings. A possible direction to alleviate these issues is to reduce the reliance on exact type annotations during schema in-

duction, so that schema-level search can still cover relevant paths under incomplete, noisy, or weakly structured schema information. Second, PSR relies on relation labels predicted in the LLM-generated plan to guide schema-level search. When these predicted relations deviate substantially from the question semantics, the search process may spend its budget on irrelevant schema branches and fail to retrieve useful paths. A possible remedy is to introduce an iterative mechanism in which grounding failures are used as feedback to refine the initial plan, thereby correcting semantic deviations during retrieval.

Acknowledgments

This work was supported by the Noncommunicable Chronic Diseases-National Science and Technology Major Project (2025ZD0544900), the Fundamental Research Funds for the Central Universities (No. 2026300429), and the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (No. JYB2025XDXM118).

References

- Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfschagen, Stephen Tavenor, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. [A survey of monte carlo tree search methods](#). *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.
- Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024. Plan-on-graph: self-correcting adaptive planning of large language model on knowledge graphs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems, NIPS '24*, Red Hook, NY, USA. Curran Associates Inc.
- Pierre-Arnaud Coquelin and Rémi Munos. 2007. Bandit algorithms for tree search. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence, UAI'07*, page 67–74, Arlington, Virginia, USA. AUAI Press.
- Wenqi Fan, Yajuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. [A survey on rag meeting llms: Towards retrieval-augmented large language models](#). In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '24*, page 6491–6501, New York, NY, USA. Association for Computing Machinery.
- Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. [Beyond i.i.d.](#):

- Three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, WWW '21, page 3477–3488, New York, NY, USA. Association for Computing Machinery.
- Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. [StructGPT: A general framework for large language model to reason over structured data](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251, Singapore. Association for Computational Linguistics.
- Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, Yang Song, Chen Zhu, Hengshu Zhu, and Ji-Rong Wen. 2025. [KG-agent: An efficient autonomous agent framework for complex reasoning over knowledge graph](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9505–9523, Vienna, Austria. Association for Computational Linguistics.
- Vladimir I Levenshtein et al. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union.
- Yangning Li, Weizhi Zhang, Yuyao Yang, Wei-Chieh Huang, Yaozu Wu, Junyu Luo, Yuanchen Bei, Henry Peng Zou, Xiao Luo, Yusheng Zhao, Chunkit Chan, Yankai Chen, Zhongfen Deng, Yinghui Li, Haitao Zheng, Dongyuan Li, Renhe Jiang, Ming Zhang, Yangqiu Song, and Philip S. Yu. 2025. [A survey of RAG-reasoning systems in large language models](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 12120–12145, Suzhou, China. Association for Computational Linguistics.
- Xujian Liang and Zhaoquan Gu. 2025. [Fast think-on-graph: wider, deeper and faster reasoning of large language model on knowledge graph](#). In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press.
- Runxuan Liu, Bei Luo, Jiaqi Li, Baoxin Wang, Ming Liu, Dayong Wu, Shijin Wang, and Bing Qin. 2025. [Ontology-guided reverse thinking makes large language models stronger on knowledge graph question answering](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15269–15284, Vienna, Austria. Association for Computational Linguistics.
- Xiao Long, Liansheng Zhuang, Aodi Li, Minghong Yao, and Shafei Wang. 2025. [Eperm: an evidence path enhanced reasoning model for knowledge graph question and answering](#). In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press.
- Haoran Luo, Haihong E, Yikai Guo, Qika Lin, Xiaobao Wu, Xinyu Mu, Wenhao Liu, Meina Song, Yifan Zhu, and Anh Tuan Luu. 2025. [KBQA-o1: Agentic knowledge base question answering with monte carlo tree search](#). In *Forty-second International Conference on Machine Learning*.
- Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024. [Reasoning on graphs: Faithful and interpretable large language model reasoning](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Jie Ma, Zhitao Gao, Qi Chai, Wangchun Sun, Pinghui Wang, Hongbin Pei, Jing Tao, Lingyun Song, Jun Liu, Chen Zhang, and Lizhen Cui. 2025. [Debate on graph: a flexible and reliable reasoning framework for large language models](#). In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence and Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence and Fifteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'25/IAAI'25/EAAI'25. AAAI Press.
- Costas Mavromatis and George Karypis. 2025. [GNN-RAG: Graph neural retrieval for efficient large language model reasoning on knowledge graphs](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 16682–16699, Vienna, Austria. Association for Computational Linguistics.
- Amrita Saha, Vardaan Pahuja, Mitesh M. Khapra, Karthik Sankaranarayanan, and Sarath Chandar. 2018. Complex sequential question answering: towards learning to converse over linked question answer pairs with a knowledge graph. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press.
- Xiaozhuang Song, Shufei Zhang, and Tianshu Yu. 2025. [ReKG-MCTS: Reinforcing LLM reasoning on knowledge graphs via training-free Monte Carlo tree search](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 9288–9306, Vienna, Austria. Association for Computational Linguistics.
- Yuan Sui, Yufei He, Nian Liu, Xiaoxin He, Kun Wang, and Bryan Hooi. 2025. [FiDeLiS: Faithful reasoning in large language models for knowledge graph question answering](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8315–8330, Vienna, Austria. Association for Computational Linguistics.

- Haitian Sun, Tania Bedrax-Weiss, and William Cohen. 2019. [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)*, pages 2380–2390, Hong Kong, China. Association for Computational Linguistics.
- Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M. Ni, Heung-Yeung Shum, and Jian Guo. 2024. [Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Alon Talmor and Jonathan Berant. 2018. [The web as a knowledge-base for answering complex questions](#). In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.
- Song Wang, Junhong Lin, Xiaojie Guo, Julian Shun, Jundong Li, and Yada Zhu. 2025. [Reasoning of large language models over knowledge graphs with super-relations](#). In *The Thirteenth International Conference on Learning Representations*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS ’22*, Red Hook, NY, USA. Curran Associates Inc.
- Guanming Xiong, Haochen Li, and Wen Zhao. 2025. [Mcts-kbqa: Monte carlo tree search for knowledge base question answering](#). *Preprint*, arXiv:2502.13428.
- Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. [Semantic parsing via staged query graph generation: Question answering with knowledge base](#). In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China. Association for Computational Linguistics.

A Appendix

A.1 Implementation Details.

All experiments are conducted on a server with 512GB RAM and 8 NVIDIA RTX 4090 GPUs, using Python 3.10 and PyTorch 2.8, all LLM calls use a decoding temperature of 0.7. We set the retrieval depth to $k = 3$ (Notably, a schema-level depth of $k = 3$ easily accommodates 4-hop instance-level questions (e.g., in CWQ) because intermediate mediator nodes sharing the same type semantics are inherently compressed during schema induction) and run PR-MCTS with a per-plan simulation budget $B = 500$ and retain the top- $K = 40$ candidate schema-level paths for LLM selection. During expansion, we sample at most $m = 7$ untried actions to control branching. In the UCB rule, we set the exploration constant $\lambda = 1.0$, the plan-alignment weight $\alpha = 2.0$, and the structural penalty weight $\beta = 0.5$. For $\text{LexSim}(\cdot, \cdot)$ in Eq. (13), we employ normalized Levenshtein similarity (Levenshtein et al., 1966). The two similarities are used at different stages. Eq. (13) is repeatedly used during PR-MCTS to compare $t(r)$, the textual form of a candidate relation type r , with the plan-implied relation labels. Lexical similarity preserves domain prefixes and fine-grained relation constraints, avoiding semantic over-smoothing during intermediate traversal. In contrast, Eq. (15) is applied only after PR-MCTS returns $\mathcal{P}_{\text{mcts}}^T$, where cosine similarity between the normalized embeddings of $\tilde{p}$ and $\bar{p}$ captures paraphrases and the entity types included in the plan and candidate schema-level path. For the mapping function $t(\cdot)$, we convert internal KG identifiers into human-readable strings while retaining domain prefixes to preserve hierarchical semantics (e.g., replacing separators with spaces, so `film.film.production_companies` becomes “film film production companies”). Furthermore, generic answer types (e.g., `type.datetime`) are mapped to descriptive phrases (e.g., “date and time representation”), ensuring the semantic encoder robustly aligns them with the LLM’s predictions during terminal evaluation in Eq. (14).

A.2 LLM Interaction Complexity of the Baselines

We provide additional details for each baseline and summarize its LLM-interaction complexity in Table 6. When a baseline includes fine-tuned LLM components, we count their model invocations as LLM calls.

- **RoG** (Luo et al., 2024) invokes an LLM to generate a relation-path plan, grounds the plan in the KG via retrieval, and then invokes the LLM once to generate the final answer from the retrieved evidence. Therefore, RoG uses 2 LLM calls.
- **EPERM** (Long et al., 2025) uses fine-tuned LLM components as scoring functions inside an iterative subgraph retriever. With beam width W and retrieval depth D , the retriever performs relation scoring for each beam state at each hop, resulting in DW model invocations in the retrieval stage. Together with two additional stage-level calls for weighted-plan generation and answer prediction, EPERM uses $DW + 2$ LLM calls.
- **KBQA-o1** (Luo et al., 2025) uses MCTS where the LLM is used for both policy scoring and reward scoring. Over T iterations, with R rollouts per iteration and rollout depth D , and with W candidate next-step explorations (action/next-state options) scored and reranked by the policy LLM at each expansion, KBQA-o1 uses $1 + T(2RD + W)$ LLM calls.
- **KG-Agent** (Jiang et al., 2025) runs an LLM-driven agent loop. The LLM is invoked once per iteration to choose tools or actions until termination. With multi-hop depth D and p additional verification or self-correction iterations, KG-Agent uses $D + p$ LLM calls.
- **FastToG** (Liang and Gu, 2025) performs community-by-community reasoning with width W and maximum iteration D . In the initial phase, it invokes the LLM once to extract the query entities, once to select W head communities from the candidate communities produced by local community detection and coarse pruning, and once to run reasoning over the resulting W community chains. In the reasoning phase, for each iteration, it invokes the LLM W times to select one next community for each chain from the candidate communities, and once to run reasoning over the updated W chains; thus it uses $W + 1$ calls per iteration. Therefore, in the worst case of D iterations, FastToG uses $(W + 1)D + 2$ LLM calls.

- **ORT** (Liu et al., 2025) invokes an LLM for ontology-label induction over extracted aim/-condition items, which uses L_{ont} calls. It then invokes the LLM once to predict the target answer label. After that, it enumerates label-level paths on the ontology graph via an LLM-free reverse-search up to depth D . It finally invokes the LLM once for semantic pruning and once for answer generation. Therefore, ORT uses $L_{\text{ont}} + 3$ LLM calls.
- **FiDeLiS** (Sui et al., 2025) validates W path candidates per hop. At each of the D hops, it invokes the LLM W times for verification and once for selection. Including one additional final call for answer generation, FiDeLiS uses $D(W + 1) + 1$ LLM calls.
- **ToG** (Sun et al., 2024) performs hop-by-hop beam search with width W and hop limit D . At each hop, it makes $2W$ LLM calls to select the next relation and the next entity for the W partial paths, and makes one additional LLM call to decide whether to stop or continue. After termination, it makes one final LLM call to generate the answer conditioned on the retrieved KG triples. Thus, in the worst case of D hops, ToG uses $2WD + D + 1$ LLM calls.
- **PoG** (Chen et al., 2024) performs T search iterations. In each iteration, it performs prompting across D hops and uses $4D$ calls, and it additionally triggers reflection for p times. Including one initial call to generate a reasoning plan, PoG uses $1 + T(4D + p)$ LLM calls.
- **DoG** (Ma et al., 2025) uses hop-by-hop traversal with debate. At each of the D hops, it performs relation filtering and selection with $a + 1$ calls. In addition, for each of the $D - 1$ transitions it performs 3 debate rounds. Therefore, DoG uses $D(a + 1) + 3(D - 1)$ LLM calls.
- **ReKnoS** (Wang et al., 2025) combines a base traversal stage and a super-relation induction stage. The base traversal stage uses $2D + 1$. The super-relation induction stage uses D_{SR} calls, where the LLM is invoked iteratively to induce high-level relational abstractions (super-relations) that compress multi-hop paths into single reasoning steps. Therefore, ReKnoS uses $(2D + 1) + D_{\text{SR}}$ LLM calls.

Method	# LLM Calls
<i>Fine-tuned/trained reasoning path retrieval</i>	
RoG (Luo et al., 2024)	2
EPERM (Long et al., 2025)	$DW + 2$
KBQA-o1 (Luo et al., 2025)	$1 + T(2RD + W)$
KG-Agent (Jiang et al., 2025)	$D + p$
<i>Few-shot/zero-shot reasoning path retrieval</i>	
FastToG (Liang and Gu, 2025)	$(W + 1)D + 2$
ORT (Liu et al., 2025)	$L_{\text{ont}} + 3$
FiDeLiS (Sui et al., 2025)	$D(W + 1) + 1$
ToG (Sun et al., 2024)	$2WD + D + 1$
PoG (Chen et al., 2024)	$1 + T(4D + p)$
DoG (Ma et al., 2025)	$D(a + 1) + 3(D - 1)$
ReKnoS (Wang et al., 2025)	$(2D + 1) + D_{\text{SR}}$
PSR (ours)	3

Table 6: LLM calls in existing methods (D : retrieval depth; W : beam width; T : search budget; R : roll-outs per iteration; p : reflection triggers; D_{SR} : depth of super-relation induction; L_{ont} : represents the LLM calls needed for ontology graph induction from the KG; a : relation filtering calls).

A.3 Dataset Statistics

We evaluate PSR on three Freebase KGQA benchmarks, WebQSP (Yih et al., 2015), CWQ (Talmor and Berant, 2018), and GrailQA (Gu et al., 2021). We follow the evaluation split adopted in recent work (Sun et al., 2024; Chen et al., 2024; Wang et al., 2025) and report results on the test split. Table 7 presents their hop distributions.

WebQSP contains 1,639 test questions and its maximum reasoning hop is 2. Under this split, 65.49% of WebQSP test questions are 1-hop and 34.51% are 2-hop. CWQ contains 3,519 test questions and its maximum reasoning hop is 4. Under this split, 40.91% of CWQ test questions are 1-hop, 38.34% are 2-hop, and 20.75% require ≥ 3 -hop reasoning. GrailQA contains 1,000 test questions and its maximum reasoning hop is 3. Under this split, 74.60% of GrailQA test questions are 1-hop, 24.50% are 2-hop, and only 0.90% require ≥ 3 -hop reasoning.

Table 8 reports their answer-set size distributions. WebQSP has a higher proportion of multi-answer questions: 51.2% of test questions have a single answer, 27.4% have 2–4 answers, 8.3% have 5–9 answers, and 12.1% have ≥ 10 answers. CWQ is dominated by single-answer questions: 70.6% of test questions have a single answer, 19.4% have 2–4 answers, 6.0% have 5–9 answers, and 4.0%

Dataset	License	#Test	Max-hop	1-hop	2-hop	≥ 3 -hop
WebQSP	CC BY 4.0	1,639	2	65.49%	34.51%	0.00%
CWQ	Apache-2.0	3,519	4	40.91%	38.34%	20.75%
GrailQA	CC BY-SA 4.0	1,000	3	74.60%	24.50%	0.90%

Table 7: Dataset licenses and hop distributions of test questions in WebQSP, CWQ and GrailQA.

Dataset	1	2–4	5–9	≥ 10
WebQSP	51.20%	27.40%	8.30%	12.10%
CWQ	70.60%	19.40%	6.00%	4.00%
GrailQA	68.0%	18.00%	5.80%	8.20%

Table 8: Answer-set size distributions of test questions in WebQSP, CWQ, and GrailQA. The statistics are derived from FiDeLiS (Sui et al., 2025) and ToG (Sun et al., 2024).

have ≥ 10 answers. GrailQA shows a distribution closer to CWQ than to WebQSP, with 68.0% of test questions having a single answer, 18.0% having 2–4 answers, 5.8% having 5–9 answers, and 8.2% having ≥ 10 answers.

A.4 Algorithmic Description of PSR

Algorithm 1 presents the overall procedure of PSR, and Algorithm 2 details PR-MCTS.

In addition, we provide a time complexity analysis of PR-MCTS. Let $A = \max_{c_d} |\text{Succ}(c_d)|$ denote the maximum number of actions at a state. For each plan, PR-MCTS performs B simulations, each containing at most k transitions. At each depth, Selection evaluates at most A actions, while Expansion considers at most m sampled unvisited actions. Therefore, the time complexity is $O(Bk(A + m))$ per plan and $O(|\Pi|Bk(A + m))$ per question.

A.5 Experimental Results under Lighter LLM Backbones

To further evaluate whether PSR remains effective under constrained computational budgets, we additionally replace the original large backbone with a substantially lighter open-source LLM. Specifically, we use Qwen-2.5-14B-Instruct, while keeping the datasets, prompts, retrieval settings, and evaluation protocol the same as in Section 4 unless otherwise specified. We compare PSR against ToG and PoG using the same metrics as in the main paper, including Hits@1, the average number of LLM calls, and token consumption.

As shown in Table 9, PSR consistently maintains the best overall performance under the lighter

Method	WebQSP			CWQ		
	Hits@1	Calls	Tokens	Hits@1	Calls	Tokens
<i>Backbone: Qwen-2.5-14B-Instruct</i>						
ToG	60.31	7.99	1143.61	35.32	10.82	1743.62
PoG	77.60	8.06	311.94	55.71	17.24	512.09
PSR	79.32	3	245.33	59.87	3	347.42
PSR vs. ToG	+19.01	-4.99	-945.28	+24.55	-7.82	-1396.20
PSR vs. PoG	+1.72	-5.06	-113.61	+4.16	-14.24	-164.67

Table 9: Results under a lighter LLM backbone.

Qwen-2.5-14B-Instruct backbone while preserving its exact 3-call interaction pattern. Compared with ToG, PSR improves Hits@1 from 60.31 to 79.32 on WebQSP and from 35.32 to 59.87 on CWQ, while also substantially reducing both LLM calls and token consumption. Compared with PoG, PSR still achieves higher accuracy on both datasets, with a notably smaller number of LLM calls and lower token usage. These results indicate that the advantage of PSR is not tied to very large backbones, and can generalize to lighter open-source LLMs under more constrained computational settings.

A.6 Inference Latency Analysis

We further evaluate the inference latency using Qwen-2.5-72B-Instruct, the open-source LLM adopted in our main experiments. For the KG backend, all methods are evaluated on Freebase, and all graph retrieval and grounding operations are executed through SPARQL queries under the same implementation setting. Note that FiDeLiS originally runs on a RoG-constructed subgraph of Freebase, while ToG, PoG, ReKnoS, and PSR are evaluated on the real Freebase backend. To make the comparison transparent, we report FiDeLiS under two settings: FiDeLiS (Subgraph), which follows its original subgraph-based setting, and FiDeLiS, where we adapt FiDeLiS to the real Freebase backend for a setting more consistent with the other baselines. We report the average end-to-end latency per question, including SPARQL-based graph retrieval, schema-level search, LLM inference, and final answer generation.

Algorithm 1 PSR**Input:** Knowledge graph $\mathcal{G}$, question q , and topic entities $\mathcal{V}_q$ **Output:** Answer set $\mathcal{A}$

- 1: Generate a plan set $\Pi \leftarrow \text{LLM}(q, \mathcal{V}_q, \{\text{Types}(e) : e \in \mathcal{V}_q\})$ ▷ Eqs. (1)–(2)
- 2: Construct the schema graph $\mathcal{S} = (\mathcal{C}, \mathcal{R}, \mathcal{E}_{\mathcal{S}})$ from $\mathcal{G}$ ▷ Eq. (3)
- 3: **for all** plan $\pi \in \Pi$ with topic entity type c_{topic} **do**
- 4: Unroll $\mathcal{S}$ into a schema tree $\mathcal{T}$ rooted at c_{topic} ▷ Eq. (4)
- 5: Extract schema-level paths $\mathcal{P}_{\text{mcts}}^{\mathcal{T}} \leftarrow \text{PR-MCTS}(\mathcal{T}, \pi)$ with simulation budget B ▷ Eqs. (5)–(14)
- 6: $\mathcal{P}_K^{\mathcal{T}} \leftarrow$ Top- K schema-level paths in $\mathcal{P}_{\text{mcts}}^{\mathcal{T}}$ ▷ Eq. (15)
- 7: **end for**
- 8: Selected schema-level paths $\mathcal{P}_s \leftarrow \text{Select}_{\text{LLM}}(q, \bigcup_{\pi \in \Pi} \mathcal{P}_K^{\mathcal{T}})$ ▷ Eq. (16)
- 9: Retrieve triples $\mathcal{F} \leftarrow \text{Retrieve}(\mathcal{G}, \mathcal{V}_q, \mathcal{P}_s)$ ▷ Eq. (17)
- 10: Generate answers $\mathcal{A} \leftarrow \text{LLM}(q, \mathcal{F})$ ▷ Eq. (18)
- 11: **return** $\mathcal{A}$

Method	WebQSP (s)	CWQ (s)
ToG	58.67	106.77
PoG	47.97	103.97
FiDeLiS (Subgraph)	34.80	59.80
FiDeLiS	80.20	98.20
ReKnoS	68.78	59.72
PSR	21.37	37.32
PSR vs. ToG	-37.30	-69.45
PSR vs. PoG	-26.60	-66.65
PSR vs. FiDeLiS (Subgraph)	-13.43	-22.48
PSR vs. FiDeLiS	-58.83	-60.88
PSR vs. ReKnoS	-47.41	-22.40

Table 10: Average end-to-end inference latency per question on WebQSP and CWQ.

As shown in Table 10, PSR consistently achieves the lowest inference latency on both datasets. On WebQSP, PSR reduces the average latency from 58.67s, 47.97s, and 68.78s to 21.37s, compared with ToG, PoG, and ReKnoS, respectively. Similarly, compared to the real-backend version of FiDeLiS (80.20s), PSR yields a massive reduction of 58.83s. Even when compared to the highly restricted FiDeLiS (Subgraph) setting (34.80s), PSR is still significantly faster. On CWQ, PSR reduces the latency from 106.77s, 103.97s, and 59.72s to 37.32s (and from 98.20s for FiDeLiS). In terms of relative speedup, PSR is approximately $2.75\times$ faster than ToG, $2.24\times$ faster than PoG, and $3.22\times$ faster than ReKnoS on WebQSP, and approximately $2.86\times$, $2.79\times$, and $1.60\times$ faster on CWQ, respectively. This result is consistent with the constant 3-call interaction pattern of PSR, which avoids the sequential $O(W)$ or $O(D)$ style LLM invocation overhead required by iterative retrieval methods.

Dataset	Total Time	PR-MCTS Search	Text Encoding
WebQSP	21.37s	1.20s (6%)	0.85s (4%)
CWQ	37.32s	1.80s (5%)	1.10s (3%)

Table 11: Runtime breakdown of the non-LLM components in PSR.

To further quantify the non-LLM overhead of PSR, Table 11 reports the runtime of PR-MCTS search and text encoding within the end-to-end inference latency.

PR-MCTS search and text encoding together account for only 2.05s (10%) and 2.90s (8%) of the total latency on WebQSP and CWQ, respectively. These results show that PSR does not trade fewer LLM calls for heavy non-LLM computation.

A.7 Evaluation on GrailQA

We further evaluate PSR on GrailQA, a more recent and more challenging benchmark for knowledge base question answering. Following prior LLM-based KGQA work such as ToG (Sun et al., 2024) and PoG (Chen et al., 2024), we use a subset of 1,000 questions to maintain a fair and consistent comparison setting.

Table 12 presents the results on GrailQA. The results show that PSR remains highly competitive on this more challenging benchmark. Compared with ToG, PSR achieves higher Hits@1 (83.74 vs. 81.40) and F1 (72.41 vs. 66.60), while requiring substantially fewer LLM calls (3 vs. 11.10) and much fewer tokens (188.60 vs. 774.60). Compared with PoG, PSR attains comparable Hits@1 (83.74 vs. 84.70), while further reducing the number of LLM calls (3 vs. 6.50) and slightly lowering token

Algorithm 2 PR-MCTS

Input: Schema tree $\mathcal{T}$ rooted at c_{topic} , plan π , simulation budget B , depth limit k , and maximum number m of sampled unvisited actions

Output: Candidate schema-level path set $\mathcal{P}_{\text{mcts}}^{\mathcal{T}}$

```
1: Initialize  $N(c_d, a) \leftarrow 0$ ,  $W(c_d, a) \leftarrow 0$ , and  $Q(c_d, a) \leftarrow 0$ 
2: for  $b = 1$  to  $B$  do
3:   Set  $c_0 \leftarrow c_{\text{topic}}$  and  $d \leftarrow 0$ 
4:   Initialize an empty selected sequence of transitions
5:   while  $d < k \wedge \text{Succ}(c_d) \neq \emptyset$  do
6:     if  $N(c_d, a) > 0$  for all  $a \in \text{Succ}(c_d)$  then
7:       Select  $a_d^{(b)} \leftarrow \arg \max_{a \in \text{Succ}(c_d)} \text{Score}(c_d, a)$  ▷ Selection; Eqs. (8)–(13)
8:     else
9:       Uniformly sample at most  $m$  unvisited actions from  $\text{Succ}(c_d)$ 
10:      Select the sampled action  $a_d^{(b)}$  that minimizes  $\mathcal{H}(a)$  ▷ Expansion;
11:    end if
12:    Add  $(c_d, a_d^{(b)})$  to the selected sequence of transitions
13:    Let the selected action be  $a_d^{(b)} = (c_{d+1}, r_{d+1})$ 
14:    Move to the next state  $c_{d+1}$  and set  $d \leftarrow d + 1$ 
15:  end while
16:  Set  $\ell \leftarrow d$  and evaluate the terminal state  $c_\ell$  by computing  $V(c_\ell)$  ▷ Evaluation; Eq. (14)
17:  for all  $(c_d, a_d^{(b)})$  in the selected sequence do ▷ Backpropagation
18:    Increase the visit count:  $N(c_d, a_d^{(b)}) \leftarrow N(c_d, a_d^{(b)}) + 1$ 
19:    Update the cumulative return:  $W(c_d, a_d^{(b)}) \leftarrow W(c_d, a_d^{(b)}) + V(c_\ell)$ 
20:    Update the mean action value  $Q(c_d, a_d^{(b)})$  according to Eq. (7)
21:  end for
22:  Record the corresponding root-to-terminal schema-level path
23: end for
24: Form  $\mathcal{P}_{\text{mcts}}^{\mathcal{T}}$  from the unique recorded paths, ranked by visit frequency
25: return  $\mathcal{P}_{\text{mcts}}^{\mathcal{T}}$ 
```

consumption (188.60 vs. 202.80). Overall, these results suggest that PSR maintains strong answer quality and efficiency on GrailQA.

A.8 Retrieval Quality Analysis

To further evaluate whether the retrieved paths provide sufficient evidence before answer generation, we report Gold Answer Recall (GAR). GAR measures the average fraction of gold answer entities covered by the grounded paths fed into the final LLM call for answer generation. This metric directly evaluates whether the retrieval and grounding stages identify answer-supporting KG evidence before the LLM produces the final answer.

As shown in Table 13, PSR achieves the highest GAR on all three benchmarks. Compared with ToG, PSR improves GAR by 25.30 points on WebQSP, 29.20 points on CWQ, and 9.44 points on GrailQA. Compared with PoG, PSR also improves

GAR by 9.02, 5.20, and 3.52 points, respectively. These results show that PSR not only improves final answer accuracy, but also retrieves grounded evidence that covers a larger fraction of gold answers before the final LLM call. This further supports the effectiveness of PSR’s plan-guided schema-level path retrieval and path grounding.

To further isolate the contribution of the third LLM call, we keep the selected schema-level paths and grounded triples unchanged, and replace only final answer generation with deterministic answer extraction. As shown in Table 14, the final LLM improves Hits@1 by 14.16 points on WebQSP and 9.00 points on CWQ, indicating that it helps compose constraints and normalize answers from the retrieved evidence.

Method	Overall	I.I.D.	Compositional	Zero-shot	F1	Calls	Tokens
<i>LLM-based QA without KG</i>							
IO Prompt	29.60	23.10	19.80	31.20	–	1	–
CoT	28.10	22.30	21.20	32.10	–	1	–
<i>Fine-tuned reasoning path retrieval</i>							
RoG	57.80	–	–	–	56.20	2	–
KG-Agent	77.50	–	–	–	65.30	–	–
<i>Few-shot/zero-shot reasoning path retrieval</i>							
<i>Backbone: GPT-4</i>							
DoG	80.00	–	–	–	–	–	–
ReKnoS	83.60	86.10	68.40	87.50	70.20	10.20	–
ToG	81.40	79.40	63.70	86.50	66.60	11.10	774.60
PoG	84.70	87.90	69.70	88.60	–	6.50	202.80
PSR w/ GPT-4	83.74	86.20	68.60	87.70	72.41	3	188.60
<i>PSR vs. ToG</i>	+2.34	+6.80	+4.90	+1.20	+5.81	-8.10	-586.00
<i>PSR vs. PoG</i>	-0.96	-1.70	-1.10	-0.90	–	-3.50	-14.20

Table 12: Main experimental results on GrailQA.

Method	WebQSP	CWQ	GrailQA
ToG	62.30	52.00	77.16
PoG	78.58	76.00	83.08
PSR	87.60	81.20	86.60

Table 13: Retrieval quality before answer generation, measured by Gold Answer Recall (%).

Method	WebQSP	CWQ
PSR w/ deterministic extraction	73.22	61.67
Full PSR w/ final LLM	87.38	70.67

Table 14: Diagnostic ablation of Stage III in Hits@1.

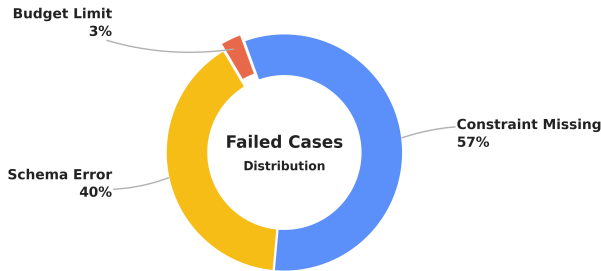


Figure 3: Failed cases analysis.

A.9 Analysis of Failed Cases

To better understand the limitations of PSR, we manually analyzed 100 failed cases from CWQ and identified three dominant error types, as shown in Figure 3, constraint missing (57%), schema error (40%), and budget limit (3%).

The largest portion of failures comes from missing constraints, as illustrated in Table 15, where the

high-level plan captures the main topic entities and core relational semantics but fails to preserve key operators such as comparison, superlative, or aggregation constraints. The second major source is schema error, as illustrated in Table 16, where the generated or selected schema-level path is semantically plausible but structurally inconsistent with the correct Freebase schema composition, leading downstream grounding to instantiate an incorrect path. Only a small portion of failures is caused by budget limit, as illustrated in Table 17, where the high-level plan is largely correct but decisive evidence cannot be fully explored under limited search budgets on large fan-out nodes. These cases illustrate the grounding gap in PSR, where a schema-level path remains plausible but fails to instantiate into factual triples during Stage III.

In the following, we provide one representative failed case for each category to illustrate how these errors arise and propagate across the three stages of PSR.

A.10 Prompts Used in PSR

PSR uses exactly three LLM calls per question, corresponding to Stage I (§3.3), Stage II (§3.4), and Stage III (§3.5). For each call, we use a fixed prompt template and only substitute the inputs required by that stage. Specifically, the first Prompt (Figure 4) takes the question q , topic entities $\mathcal{V}_q$, and their type set $\{\text{Types}(e) : e \in \mathcal{V}_q\}$ as inputs and outputs a plan set Π . The second Prompt (Figure 5) takes the question q and the candidate schema-level paths in $\mathcal{P}_K^T$ as inputs and outputs

Question	Which globe region with the UTC−05:00 time zone does the Dominican Republic call home?
Answer	Gold: Greater Antilles; Pred: Atlantic Time Zone
Stage I	The best generated plan already reveals the core issue: the <i>Dominican Republic</i> branch is reduced to time-zone retrieval, rather than preserving the target answer type <i>globe region</i>. Plan-1: Dominican Republic→location.country.time_zones →time.time_zone Topic Entity Type-1: location.country Answer Entity Type-1: time.time_zone ...
Stage II	The selected paths continue this bias and all terminate in time.time_zone, so the search is centered on time zones rather than regions. Path-1: location.location→location.location.time_zones Path-2: location.country→location.country.administrative_divisions→location.location.time_zones Path-3: location.country→location.country.first_level_divisions →location.location.time_zones ...
Stage III	The grounded evidence only supports time-zone entities, so the system finally outputs Atlantic Time Zone rather than the correct region Greater Antilles. Dominican Republic→location.location.time_zones → {Atlantic Time Zone, UTC−04:00, ...}
Diagnosis	This case belongs to constraint missing . The system preserves the time-zone condition but fails to preserve the target-type constraint that the answer should be a geographic region. Consequently, the later stages consistently follow an incorrect answer type.

Table 15: Case study of CONSTRAINT MISSING.

the selected schema-level path set $\mathcal{P}_s$. The third Prompt (Figure 6) takes the question q and the retrieved triple set $\mathcal{F}$ as inputs and outputs the final answer set $\mathcal{A}$. In all three prompts, the output format is explicitly constrained to avoid free-form responses.

Question	On what radio station in New York City would you find an organizational leader named Gary Knell?
Answer	Gold: WNYC; Pred: Letter from America
Stage I	The generated plan actually targets the correct answer type <code>broadcast.radio_station</code>, indicating that the initial semantic abstraction is largely reasonable. Plan-1: Gary Knell→business.board_member.organization→ organization.organization.radio_stations →broadcast.radio_station Topic Entity Type-1: business.board_member Answer Entity Type-1: broadcast.radio_station
Stage II	However, the selected paths drift away from the intended station schema and instead terminate in <code>radio.radio_program</code>, which changes the target entity type from <code>broadcast.radio_station</code> to <code>radio.radio_program</code>. Path-1: people.person→people.person.nationality→ radio.radio_subject.programs_with_this_subject Path-2: location.citytown→location.citytown.postal_codes →location.postal_code.country → radio.radio_subject.programs_with_this_subject Path-3: location.location→location.location.events→ radio.radio_subject.programs_with_this_subject ...
Stage III	The grounded evidence consequently supports a radio-program entity rather than a radio-station entity, which leads PSR to output Letter from America. Gary Knell→people.person.nationality→ radio.radio_subject.programs_with_this_subject→{Letter from America, ...}
Diagnosis	This case belongs to schema error. Although the generated plan correctly targets <code>broadcast.radio_station</code>, the path-selection stage shifts to semantically related but structurally incorrect program-oriented paths. As a result, the grounding stage retrieves a <code>radio.radio_program</code> instead of the required radio station, producing Letter from America rather than WNYC.

Table 16: Case study of SCHEMA ERROR.

Question	What Hollywood studio produced the American Gladiators TV movie?
Answer	Gold: Universal Studios Hollywood; Pred: Trans World International
Stage I	The generated plans target the correct answer type, suggesting that the high-level semantic abstraction is largely correct. Plan-1: Hollywood→film.film_location.films_filmed_here →film.film.production_companies → film.production_company Topic Entity Type-1: film.film_location Answer Entity Type-1: film.production_company
Stage II	Although the target answer type remains correct, the selected paths do not retain the intended <code>tv_movies</code> → <code>production_companies</code> branch for <i>American Gladiators</i>. Instead, the search drifts toward producer-related alternatives under a large candidate space. Path-1: film.film_location → film.film_location.featured_in_films → film.film.production_companies Path-2: broadcast.content→broadcast.content.producer→ film.production_company.films →film.film.production_companies Path-3: broadcast.content→broadcast.content.producer→ organization.organization.parent ...
Stage III	The grounded evidence consequently follows the producer-related branch and yields Trans World International, while the correct TV-movie production-company branch is not reached. American Gladiators→broadcast.content.producer→{Trans World International, ...}
Diagnosis	This case belongs to budget limit. The generated plans already contain the correct TV-movie production-company branch, and the target answer type is also correct. However, under a large search space (e.g., 973 candidate paths for <i>Hollywood</i> and 610 candidate paths for <i>American Gladiators</i>), the correct branch is not retained during path selection. As a result, grounding follows a producer-related alternative and returns Trans World International instead of Universal Studios Hollywood.

Table 17: Case study of BUDGET LIMIT.

Prompt1: Plan Generation

Given a question and some Topic Entities, each Topic Entity has a list of types. You must generate possible relation paths for each Topic Entity.

Here are the RULES you must obey:

1. Output a JSON dict: keys = topic entity surface names.
2. For each topic entity output a list of **path objects**.
3. Each path object MUST contain:
 - "start_type": a list of types **chosen ONLY** from the given Topic Entity Types
 - "path": a relation chain "Entity -> r1 -> r2 -> ..."
 - "answer_types": plausible types of the final answer entity
 - "confidence": float 0.0-1.0, how confident you are this path answers the question
 - "reasoning": brief explanation of why this path semantically fits the question
4. For each entity output at least 2 path objects, ranked by confidence (highest first).
5. NEVER invent types. Use only the provided clean types.

Example 1

Question: Who is the director of the movie featured Miley Cyrus and was produced by Tobin Armbrust?

Topic Entities: ["Miley Cyrus", "Tobin Armbrust"]

Topic Entities Types: [["people.person", "film.actor"], ["people.person", "film.producer"]]

Output: {

 Miley Cyrus: [

 {

 start_type: ["film.actor"],
 "path": "Miley Cyrus -> film.actor.film -> film.performance.film -> film.film.directed_by",
 "answer_types": ["film.director", "people.person"],
 "confidence": 0.95,
 "reasoning": "...."

 },

 Tobin Armbrust: [

 {

 start_type: ["film.producer"],
 "path": "Tobin Armbrust -> film.producer.film -> film.film.directed_by",
 "answer_types": ["film.director", "people.person"],
 "confidence": 0.85,
 "reasoning": "...."

 },

} [... Examples 2, 3...]

Now begin for the real input:

Figure 4: Prompt1: Plan Generation (the 1st LLM Call).

Prompt2: Path Selection

You are a Freebase schema reasoning expert.

QUESTION: {question}

TASK: For each entity below, select the TOP 3 best schema paths that can answer the question.

SELECTION CRITERIA (in order of importance):

1. Semantic alignment: Does the path's meaning match what the question asks?
2. Answer type match: Does the end_type match the expected answer type?
3. Path coherence: Are the predicate transitions logically connected?
4. Numeric score: Use as tiebreaker only.

CRITICAL: If multiple entities are given, their paths should lead to a COMMON answer (intersection constraint).

ENTITIES AND CANDIDATE PATHS:

```
{
  "0": {
    "entity": "Miley Cyrus",
    "paths": [
      {"id": 0, "path": "film.actor.film -> directed_by", "score": 0.95},
      {"id": 1, "path": "film.actor.film -> producer", "score": 0.60},
      ...
    ]
  },
  "1": {
    "entity": "Tobin Armbrust",
    "paths": [
      {"id": 0, "path": "film.producer.film -> directed_by", "score": 0.85},
      ...
    ]
  }
}
```

OUTPUT FORMAT - Return ONLY a JSON object:

```
{
  "0": ["best_id", "second_id", "third_id"],
  "1": ["best_id", "second_id", "third_id"],
  ...
}
```

Output ONLY the JSON, no explanation.

Figure 5: Prompt2: Path Selection (the 2nd LLM Call).

Prompt3: Answer Generation

You are a knowledge graph question answering expert.

QUESTION:

Who is the director of the movie featured Miley Cyrus and was produced by Tobin Armbrust?

FACT CHAINS FROM KNOWLEDGE GRAPH:

{...}

REASONING INSTRUCTIONS:

1. Analyze what the question is asking for (person? place? date? organization? count?).
2. Trace through the fact chains to find entities that satisfy ALL constraints.
3. If multiple candidate answers exist, rank them by likelihood based on how strongly they satisfy constraints.
4. The ranked answers should reflect: (a) number of supporting fact chains, (b) structural consistency, (c) semantic alignment with the question.
5. Ignore empty results {}, unnamed entities (m.xxx, g.xxx), and irrelevant chains.

OUTPUT REQUIREMENT:

- Output ONLY the final answer(s)
- If multiple answers, output them in descending order of likelihood
- Separate multiple answers with commas
- DO NOT provide explanation
- DO NOT output "I don't know"
- DO NOT output empty response

ANSWER:

Figure 6: Prompt3: Answer Generation (the 3rd LLM Call).