LEARNING ADAPTIVE SEARCH GUIDANCE FOR KNOWLEDGE GRAPH QUESTION ANSWERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Knowledge graph question answering (KGQA) requires exploring multi-hop reasoning paths in a combinatorial space, where effective search guidance is crucial for both accuracy and efficiency. Recent MCTS-based KGQA systems often invoke large language models (LLMs) inside the search loop for expansion, simulation, or termination, incurring high cost and unreliability. We propose **TriFOR** (**Tri**-head **For**ecasting for Tree Reasoning), a PUCT-based tree search framework that removes in-loop LLM calls while retaining semantic guidance via a learned forecaster that jointly predicts (i) an action prior (policy), (ii) usefulness for successful search (value), and (iii) a stop probability (stop). TriFOR is pretrained with path supervision, and an optional variant further refines the forecaster by distilling from its own MCTS search trajectories. We also introduce **DeepHopQA**, a controlled diagnostic benchmark for long-hop KG search that addresses specific limitations of existing benchmarks, including limited long-hop coverage, missing path supervision, and incomplete subgraphs. Across various benchmarks, TriFOR achieves strong accuracy while substantially reducing inference cost.

1 INTRODUCTION

Large language models (LLMs; Achiam et al., 2023; Bai et al., 2023) have achieved strong performance across a wide range of language tasks and have been extended to question answering. However, in knowledge-intensive settings, LLMs remain prone to hallucinations (Zhuo et al., 2023; Zhang et al., 2025b). Retrieval-augmented generation (RAG; Lewis et al., 2020; Li et al., 2022) addresses this challenge by conditioning answer generation on retrieved external knowledge. Knowledge graphs (KGs; Vrandečić & Krötzsch, 2014) provide an explicit and structured representation of such knowledge through entity–relation triplets, offering a principled alternative to relying solely on knowledge implicitly encoded in model parameters. As a result, the combination of LLMs with KGs has emerged as a promising direction to improve the factual reliability.

Knowledge graph question answering (KGQA) aims to answer natural-language questions by retrieving and composing multi-hop reasoning paths over large knowledge graphs. Early retrieval-based approaches use representation learning and graph neural networks (GNNs; Scarselli et al., 2008) to reason over question-specific graph neighborhoods (Sun et al., 2018; Saxena et al., 2020; Zhang et al., 2022a; Mavromatis & Karypis, 2022). More recent systems combine retrieved graph evidence with LLMs or use LLMs to interactively traverse KGs (Mavromatis & Karypis, 2024; Baek et al., 2023; Sun et al., 2023; Luo et al., 2023). GNN-based retrieval can incur substantial subgraph computation, while LLM-guided graph search repeatedly invokes an LLM for traversal decisions (Mavromatis & Karypis, 2022; Sun et al., 2023). Motivated by the long-horizon and sparse-reward nature of multi-hop KG reasoning, recent work has reformulated KGQA as a tree search problem and applied Monte Carlo Tree Search (MCTS; Kocsis & Szepesvári, 2006). MCTS incrementally constructs reasoning paths while balancing exploration and exploitation, making it well-suited for discovering high-quality multi-hop evidence in large combinatorial spaces (Shen et al., 2025; Long et al., 2025; Song et al., 2025; Wang et al., 2025). However, existing MCTS-based KGQA systems typically invoke LLMs inside the search loop for expansion, evaluation, or termination, which substantially increases inference cost and produces unstable guidance in long-horizon reasoning where errors compound and self-evaluation becomes inconsistent (Lee et al., 2025).

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

Evaluating forecast-guided search requires supervision beyond final answers. Existing KGQA benchmarks contain few long-hop instances and often lack validated reasoning paths, while shortest paths can be semantically noisy proxies for evidence. We therefore treat long-hop KGQA evaluation as a controlled diagnostic setting: the goal is not to approximate natural question-answering circumstances, but to test whether a system can recover valid evidence paths in deep, branching graph environments.

We address these challenges with **TriFOR**, a forecaster-guided Predictor Upper Confidence Bounds for Trees (PUCT; Rosin, 2011) search framework for KGQA that eliminates in-loop LLM calls while retaining semantic guidance. Inspired by forecasting functions for reliable RAG (Band et al., 2024; Jang et al., 2025), **TriFOR** defines a learned forecaster directly over the KG space. Given a query and a partial reasoning path, a multi-head forecaster predicts (i) an action prior over outgoing relations, (ii) a continuation value estimating whether the path remains useful, and (iii) a stop probability indicating whether the current state is answer-ready. These signals jointly serve as the PUCT prior, state evaluator, and termination controller, enabling efficient long-horizon search without LLMs inside the tree. To support training and evaluation under challenging hop lengths, we further introduce **DeepHopQA**, a controlled diagnostic KGQA dataset for long-hop search. DeepHopQA pairs each question with an LLM-validated gold reasoning path and a question-specific subgraph, enabling path-level supervision and direct analysis of search behavior. **TriFOR** is pretrained offline using path supervision, with an optional **TriFOR**$^+$ refinement based on distillation from its own MCTS trajectories, and aggregates a small set of high-quality evidence paths at test time.

We evaluated our model on WebQSP (Yih et al., 2015), CWQ (Talmor & Berant, 2018), and DeepHopQA. Across benchmarks, **TriFOR** achieves strong multi-hop KGQA accuracy while substantially reducing inference cost, indicating that learned forecasting provides an effective interface for graph search. Our contributions can be summarized as follows:

- We propose **TriFOR**, a PUCT-based tree search framework for KGQA that guides expansion, evaluation, and stopping without in-loop LLM calls.
- We introduce DeepHopQA, a controlled long-hop KG search benchmark with LLM-validated gold reasoning paths for path-level supervision and diagnostic evaluation.
- We train **TriFOR** with offline path supervision and further study an optional **TriFOR**$^+$ refinement using distillation from its own MCTS trajectories.

2 PRELIMINARIES

2.1 PROBLEM SETUP

We consider the problem of answering a natural language question q using an LLM $\mathcal{M}$. In its simplest form, the model directly generates an answer $a \sim \mathcal{M}(q)$, based solely on its parametric knowledge. While such models show strong ability, it is well known that they often suffer from *hallucination* (Ji et al., 2023). A common approach to improve factual reliability is RAG, where the model retrieves external knowledge or documents d relevant to the question and conditions its generation on them:

$$a \sim \mathcal{M}(q, d), \qquad d \sim \mathrm{Retrieve}(q).$$

In this work, we focus on a particular instantiation of RAG where the external knowledge source is a KG. A KG represents factual knowledge in a structured form consisting of *entities* and *relations* (e.g., entities such as *Inception* and *Christopher Nolan*, connected by a relation *directed_by*). Retrieving information from a KG, therefore, corresponds to discovering a sequence of relational links that connect entities mentioned in the question to the answer. We study the KGQA setting where a question q is associated with one or more *seed entities* extracted from the question, and the goal is to retrieve relevant information from a KG in order to identify the correct answer entities.

2.2 QUESTION-SPECIFIC SUBGRAPH FOR KGQA

Let $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ denote a KG, where $\mathcal{E}$ is the set of entities, $\mathcal{R}$ the set of relations, and $\mathcal{T} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ the set of factual triplets. Given the large scale of the KG $\mathcal{G}$, reasoning over the entire graph for each query q can be computationally prohibitive. Instead, we assume that a question-dependent set of *seed entities* $\mathcal{S}_q \subseteq \mathcal{E}$ and question-specific local subgraph $\mathcal{G}_q = (\mathcal{E}_q, \mathcal{R}_q, \mathcal{T}_q)$, where $\mathcal{E}_q \subseteq \mathcal{E}$,

$\mathcal{R}_q \subseteq \mathcal{R}$, and $\mathcal{T}_q \subseteq \mathcal{E}_q \times \mathcal{R}_q \times \mathcal{E}_q$, are given. Within this local subgraph, the goal is to identify an answer entity set $\mathcal{Y}_q \subseteq \mathcal{E}$ by traversing relational paths that originate from the seed entities and terminate at entities that answer the question.

2.3 Multi-hop Reasoning Paths and MDP Formulation

Answering a question over a knowledge graph typically requires *multi-hop reasoning*, where multiple relational facts must be composed to connect the seed entities to the correct answer entities. Such multi-hop reasoning can be represented as a *reasoning path* in the graph. A reasoning path is a sequence of entities and relations starting from a seed entity $v_0 \in \mathcal{S}_q$, $p = (v_0, r_1, v_1, \ldots, r_L, v_L)$, where each triplet $(v_{t-1}, r_t, v_t) \in \mathcal{T}_q$ corresponds to traversing an edge in the question-specific local subgraph $\mathcal{G}_q$. A path is considered a *gold reasoning path* if its terminal entity v_L belongs to the *gold* answer set, i.e., $v_L \in \mathcal{Y}_q$, and the path provides a semantically valid reasoning chain for answering q. We say that the path is *multi-hop* if $L > 1$.

The core challenge of KGQA is to efficiently identify feasible multi-hop reasoning paths among a vast combinatorial search space. We address this challenge by formulating multi-hop reasoning over $\mathcal{G}_q$ as a Markov decision process (MDP). MDP provides a standard framework for modeling sequential decision-making problems, which is finding reliable gold reasoning paths in our setting, and is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathbb{P}, \mathcal{R} \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathbb{P}$ the transition model, and $\mathcal{R}$ the reward function. Specifically, at step t, the state is defined as

$$s_t = (q, p_{1:t}), \qquad p_{1:t} = (v_0, r_1, v_1, \ldots, r_t, v_t),$$

which encodes the question, the current entity v_t, and the partial reasoning path $p_{1:t}$ (i.e., prefix) constructed so far. For completeness, we define $p_{1:0} = (v_0)$. From state s_t, the agent selects an action corresponding to an outgoing edge from the current entity v_t,

$$\mathcal{A}(s_t) = \{(r, v') \mid (v_t, r, v') \in \mathcal{T}_q\}.$$

Given an action $a_t = (r_{t+1}, v_{t+1})$, the transition to the next state is deterministic and given by $s_{t+1} = (q, p_{1:t} \oplus a_t)$, where $\oplus$ denotes concatenation. The reward indicates whether the current entity corresponds to a correct answer, $\mathcal{R}(s_t) = \mathbb{1}[v_t \in \mathcal{Y}_q]$. Under this formulation, KGQA reduces to learning a policy that induces high-probability trajectories reaching the answer entities.

2.4 PUCT-based Monte Carlo Tree Search

Given the MDP formulation of multi-hop reasoning, our goal is to obtain a policy that efficiently discovers reasoning paths leading to the answer entities. Due to sparse rewards and the large branching factor, we adopt an MCTS approach. MCTS incrementally builds a search tree by repeatedly selecting actions, expanding new states, evaluating them, and backpropagating the obtained returns. For each state–action pair (s, a), we maintain the visit count $n(s, a)$ and an empirical action-value estimate $Q(s, a)$. We also define $N(s) = \sum_{a \in \mathcal{A}(s)} n(s, a)$ as the total visit count of state s. To balance exploration and exploitation, we use the PUCT selection rule

$$a^\star = \arg\max_{a \in \mathcal{A}(s)} \left(Q(s, a) + c_{\text{PUCT}}\, P(a \mid s) \frac{\sqrt{\max\{1, N(s)\}}}{1 + n(s, a)} \right),$$

where $P(a \mid s)$ is a prior over actions and $c_{\text{PUCT}} > 0$ controls exploration. Repeated simulations refine the induced search policy, focusing computation on promising reasoning paths.

3 Method

Overview. We present our method in three parts. In **Part I**, we introduce **TriFOR** (Tri-head Forecasting for Tree Reasoning), a lightweight model that provides PUCT-MCTS with an action prior, a continuation value, and a stop signal. We describe its state representation and offline synthetic supervision used for pretraining. In **Part II**, we perform forecast-guided PUCT-MCTS inference without in-loop LLM calls, and fine-tune TriFOR by distilling on-policy targets from its own MCTS trajectories to mitigate the train–test distribution shift. In **Part III**, we introduce **DeepHopQA**, a controlled diagnostic benchmark for long-hop KG search with LLM-validated gold reasoning paths and question-specific branching subgraphs. Refer to Figure 1 to see the overview of TriFOR.

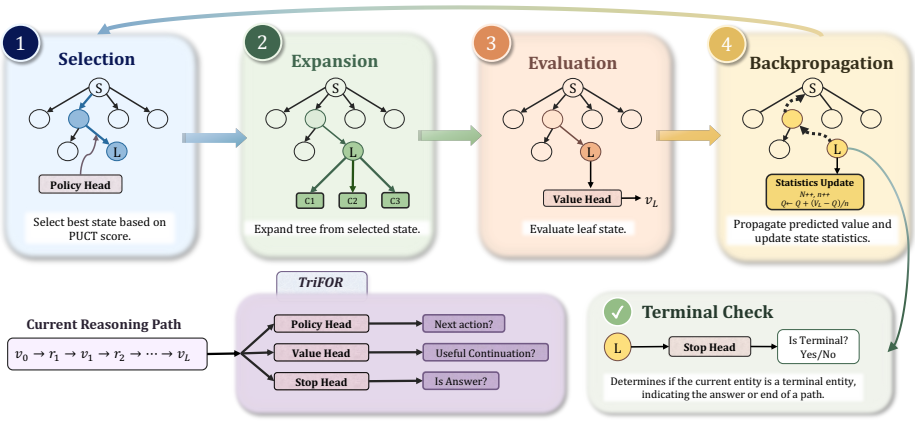

Figure 1: Overview of **TriFOR**, a forecast-guided PUCT-MCTS framework for KGQA. **TriFOR** uses a lightweight tri-head forecaster to guide search and termination without in-loop LLM calls.

3.1 Part I: TriFOR

The MCTS based on standard PUCT selection requires modeling an action prior $P(a \mid s)$ to guide expansion and an action-value estimation function $Q(s, a)$ to evaluate partial trajectories. This aligns well with KGQA, where states are partial reasoning paths and actions are outgoing KG edges. In the context of knowledge graph reasoning, however, finding gold multi-hop reasoning paths requires deciding when to stop the search precisely, motivating the need to model an explicit stopping signal.

We explicitly parameterize these three signals with a lightweight forecaster to drive the search without in-loop LLM calls. In the following, we introduce TriFOR, a tri-head forecasting model that provides action priors, value estimates, and stopping probabilities for each search state.

3.1.1 State and Action Representation

To efficiently select future actions from the current state, we design textual state and action representations that preserve partial reasoning path structure while remaining compatible with lightweight encoders. At each search step t, the state s_t consists of the question q and a partial reasoning path prefix $p_{1:t} = (v_0, r_1, v_1, \ldots, r_t, v_t)$. We linearize the path prefix into a sequence of textual hops as:

$$x_0 = \texttt{[START]}\,\texttt{[ENT]}\,v_0, \tag{1}$$

$$x_i = \texttt{[ENT]}\,v_{i-1}\,\texttt{[REL]}\,\tilde{r}_i\,\texttt{[ENT]}\,v_i \;\; (1 \le i \le t), \tag{2}$$

where each relation r_i is canonicalized into textual form $\tilde{r}_i$. The markers `[START]`, `[ENT]`, and `[REL]` are special tokens indicating the path structure and are treated as ordinary input tokens by the encoder. After linearizing the path prefix, we encode the question and hop texts using a lightweight pretrained language model (PLM). Specifically, the question is encoded once as $h_q = \mathrm{PLM}(q)$, and each hop text x_i is independently encoded using the `[CLS]` representation as $h_i = \mathrm{PLM}(x_i)_{\texttt{[CLS]}}$. To model dependencies across hops, we apply a learnable hop-level Transformer to $h_{1:t}$ for $t \ge 1$:

$$\tilde{h}_{1:t} = \texttt{Transformer}_{\text{hop}}(h_{1:t} + \texttt{PE}_{1:t}), \tag{3}$$

where $\texttt{PE}_{1:t}$ denotes positional embeddings. The root uses the seed representation h_0. We aggregate the sequence using $Z_t = \texttt{CA}(\tilde{h}_{1:t}, h_q)$ for $t \ge 1$ and $Z_0 = \texttt{CA}(h_0, h_q)$ at the root, where $\texttt{CA}$ denotes learnable cross-attention. Finally, we compute a representation for the state s_t via learnable attention pooling (Lin et al., 2017) $\texttt{AP}$, $z_t = \texttt{AP}(Z_t)$. Thus, z_t represents the state s_t, given the query q.

Each candidate action $a = (r, e) \in \mathcal{A}(s_t)$ is linearized as $\texttt{[REL]}\,\tilde{r}\,\texttt{[ENT]}\,e$ and encoded by the same PLM to obtain an action embedding $u_a = \mathrm{PLM}(a)_{\texttt{[CLS]}}$. These representations are used by the forecaster heads described next. Figure 2 shows the overall flow of state and action representation.

3.1.2 Policy, Value, and Stop Heads

Given the state representation z_t and action representations $\{u_a\}_{a \in \mathcal{A}(s_t)}$, TriFOR predicts three signals required by PUCT-MCTS: an action prior, a value estimate, and a stopping probability. Each head is designed to model a distinct aspect of decision-making during multi-hop search.

Policy head. The policy head models an action prior $P(a \mid s_t)$, the probability of selecting each action given the state s_t. Since z_t aggregates information from the question and the partial reasoning path, and each action embedding u_a represents a possible next hop, we model the action prior based on their interactions. For each $a \in \mathcal{A}(s_t)$, following a standard pairwise feature matching design (Mou et al., 2016), we compute a logit as a function of z_t and u_a with a learnable $\texttt{MLP}_\pi$:

$$\text{logit}(a \mid s_t) = \texttt{MLP}_\pi\big([\, z_t \oplus u_a \oplus |z_t - u_a| \oplus (z_t \odot u_a)\,]\big),$$

where $\odot$ denotes the element-wise product. Then, we obtain the action prior by normalizing over the available actions, $\pi_\theta(\cdot \mid s_t) = \text{softmax}_{a' \in \mathcal{A}(s_t)} \text{logit}(a' \mid s_t)$. This distribution is directly used as $P(a \mid s_t) = \pi_\theta(a \mid s_t)$ in the PUCT selection rule to guide expansion toward promising actions.

Value head. In MCTS, the quantity that guides search is the action-value $Q(s, a)$. However, directly modeling $Q(s, a)$ is difficult due to the large action space and long-horizon dependencies. Instead, PUCT-MCTS maintains an empirical estimate of $Q(s, a)$ through rollout and backpropagation, which requires evaluating partial trajectories. Accordingly, we model a state-level continuation value $V_\theta(s_t)$ that estimates the usefulness of the current partial path for successful answer recovery:

$$V_\theta(s_t) = \sigma(\texttt{MLP}_{\text{value}}(z_t)) \in (0, 1).$$

During MCTS, V_θ serves as the leaf evaluation signal used to construct empirical action-value estimates during search. The offline targets described below provide path-derived proxy supervision for this quantity, while TriFOR$^+$ further refines it using empirical MCTS success ratios. Thus, V_θ is a search-oriented continuation value rather than an exact graph-theoretic reachability probability.

Stop head. Unlike standard MCTS settings, where the search continues until a fixed depth or terminal game state, KGQA requires deciding when sufficient evidence for answering the question has been obtained. In particular, once the search reaches an entity that answers the question, further expansion is unnecessary and may introduce noise. To support adaptive termination, we introduce a stop head T_θ that models the probability of stopping at s_t:

$$T_\theta(s_t) = \sigma(\texttt{MLP}_{\text{stop}}(z_t)).$$

This signal reduces reliance on heuristic depth limits by learning when to terminate based on s_t.

Note that all three heads share the same encoder and are trained jointly, allowing TriFOR to provide coherent guidance for action selection, state evaluation, and adaptive stopping. This shared formulation motivates prefix-level supervision in which policy labels guide next-hop selection, value labels distinguish gold-prefix from corrupted off-path states, and stop labels identify answer-ready states.

3.1.3 Offline Synthetic Supervision and Pretraining

Since TriFOR explicitly models a value head and a stop head in addition to an action prior, training solely on standard KGQA supervision pairs $(q, p^\star)$ is insufficient. In particular, state-level supervision is required to learn (i) whether an intermediate state remains useful for successful continuation, and (ii) whether search should terminate at a given step. We therefore construct offline synthetic supervision by decomposing gold reasoning paths into prefix states and augmenting them with corrupted variants.

Given a gold reasoning path $p^\star = (v_0, r_1^\star, v_1^\star, \ldots, r_L^\star, v_L)$, we construct a sequence of gold prefix states $s_t^+ = (q, p_{1:t}^\star)$ for $t = 0, \ldots, L$. For each prefix, we attach supervision signals for the value and stop heads, denoted by y_t^{value} and y_t^{stop}, respectively. Since every gold prefix lies on the supervised reasoning path to the target entity v_L, we assign positive continuation labels $y_t^{\text{value}} = 1$ for all $t = 0, \ldots, L$. The stop labels are defined as $y_t^{\text{stop}} = \mathbb{1}[t = L]$, so the agent terminates only at the final step of the supervised path. For policy supervision, at each non-terminal prefix $t = 0, \ldots, L-1$, we define the gold action $a_t^+ = (r_{t+1}^\star, v_{t+1}^\star)$ and supervise a one-hot target on a_t^+.

To form negative samples for the policy head, we additionally sample a set of K negative actions $\{a_{t,k}^-\}_{k=1}^K$ from $\mathcal{A}(s_t^+) \setminus \{a_t^+\}$. To obtain synthetic negative states for value/stop learning, we sample corruption steps $t \in \{0, \ldots, L-1\}$ and replace a_t^+ with an alternative action $a_t^- \in \mathcal{A}(s_t^+) \setminus \{a_t^+\}$, yielding an off-path prefix state $s_{t+1}^- = (q, p_{1:t}^\star \oplus a_t^-)$. We label such states with $y^{\text{value}} = 0$ and $y^{\text{stop}} = 0$, and extend each corrupted prefix by a short random-walk rollout to generate an additional negative state. Because alternative routes may occasionally remain answer-reachable, these corrupted off-path labels provide approximate supervision rather than certifying graph-theoretic unreachability. Algorithm 1 summarizes the full construction procedure and its implementation details.

Offline pretraining objective. From the above supervision, we train TriFOR with policy, value, stop, and pairwise ranking losses. The policy head is optimized to rank the gold next action above sampled K negative actions using a sampled softmax cross-entropy. The value head is trained with a binary cross-entropy (BCE) loss using the continuation targets defined above. The stop head is trained with a weighted BCE (WBCE) loss to handle class imbalance due to the sparsity of terminal states along reasoning paths. Formally, we define

$$\mathcal{L}_{\text{policy}} = -\log \pi_\theta(a_t^+ \mid s_t), \quad \mathcal{L}_{\text{value}} = \texttt{BCE}\big(V_\theta(s_t),\, y_t^{\text{value}}\big), \quad \mathcal{L}_{\text{stop}} = \texttt{WBCE}\big(T_\theta(s_t),\, y_t^{\text{stop}}\big). \tag{4}$$

To further separate gold-prefix and corrupted off-path states for the same question, we add a pairwise ranking loss on value logits. Here, s_t^+ and s_t^- denote gold-prefix and corrupted off-path states for the same question, respectively, and we apply `softplus` to the difference between their value logits:

$$\mathcal{L}_{\text{PR}} = \texttt{softplus}\big(\text{logit}(V_\theta(s_t^-)) - \text{logit}(V_\theta(s_t^+))\big). \tag{5}$$

The total offline objective combines all losses, with α controlling the ranking term as

$$\mathcal{L}_{\text{off}} = \mathcal{L}_{\text{policy}} + \mathcal{L}_{\text{value}} + \mathcal{L}_{\text{stop}} + \alpha\, \mathcal{L}_{\text{PR}}. \tag{6}$$

3.2 Part II: PUCT-MCTS Inference and Fine-tuning

3.2.1 Forecast-Guided PUCT-MCTS for Multi-hop Reasoning

After offline pretraining, we use TriFOR-guided PUCT-MCTS to find gold reasoning paths. During search, TriFOR provides action priors, state-value estimates, and stopping signals, enabling effective exploration without in-loop LLM calls. Refer to Algorithm 2 for the complete procedure and details.

Selection. Given s_t, the next action $a_t^\star$ is selected by:

$$a^\star = \arg\max_{a \in \mathcal{A}(s)} \left[Q(s,a) + c_{\text{PUCT}} \pi_\theta(a \mid s) \frac{\sqrt{\max\{1, N(s)\}}}{1 + n(s,a)} \right].$$

where $n(s_t, a)$ is the visit count for s_t and a, and $N(s_t) = \sum_a n(s_t, a)$. Here, $Q(s_t, a)$ denotes the empirical action-value estimate maintained by MCTS, whose update rule is described in the backpropagation step below.

Expansion and termination. In large knowledge graphs, relying solely on PUCT selection can bias exploration toward a small subset of high-degree or early-favored actions. To this end, we optionally subsample outgoing actions and retain the top-M candidates ranked by the forecaster prior to control the branching factor. At the root, Dirichlet noise is added to the action prior to encourage exploration, following Silver et al. (2017). Beyond controlling expansion, effective search also requires deciding when further exploration is no longer beneficial. Accordingly, search terminates if any of the following conditions is met: (i) the hop budget is exhausted, (ii) no outgoing actions are available, (iii) $T_\theta(s) \geq \epsilon_{\text{stop}}$, or (iv) $V_\theta(s) \leq \epsilon_{\text{value}}$ for early termination.

Backpropagation. At the end of each simulation, we set the leaf reward to $R = V_\theta(s_{\text{leaf}})$. Then R is backed up along the selected path by updating $W(s,a) \leftarrow W(s,a) + R$, $n(s,a) \leftarrow n(s,a) + 1$, and $Q(s,a) = W(s,a)/n(s,a)$ for each (s,a). Through repeated simulations, these empirical estimates guide search toward high-quality multi-hop reasoning paths.

Evidence Extraction. After all MCTS simulations are completed, it is crucial to select reliable evidence paths that can support answering the question q. We therefore extract candidate root-to-leaf paths from the search tree and rank them. For a path p, we define its support as the minimum edge visit count $n_{\min}(p) = \min_{(s,a)\in p} n(s,a)$ and its quality as the mean action value $\overline{Q}_{\text{path}}(p) = \frac{1}{|p|} \sum_{(s,a)\in p} Q(s,a)$. We prioritize candidates with larger $n_{\min}(p)$, breaking ties by larger $\overline{Q}_{\text{path}}(p)$ and then by larger $V_\theta(s_{\text{leaf}})$. Then the top-$K$ ranked paths are provided as evidence to the model.

3.2.2 Fine-Tuning with MCTS Trajectories

While TriFOR is trained offline using synthetic supervision from the gold prefixes, we optionally apply an additional finetuning stage to further improve performance. This variant, denoted as TriFOR$^+$,

finetunes the forecaster using MCTS trajectories generated on the training questions. Concretely, we run PUCT-MCTS on training instances using the pretrained TriFOR. When $\mathcal{Y}_q$ is available, simulations terminate upon reaching a gold answer entity. For each visited s, we record its visit count $N(s)$ and the number of simulations that reach a gold answer after passing through s, denoted by $S(s)$. From each search tree, we export a set of states with sufficiently large $N(s)$.

Each exported state s is assigned distilled supervision: policy target $\pi_{\text{MCTS}}(\cdot \mid s)$ obtained by normalizing edge visit counts, value target $V^\star(s) = S(s)/N(s)$, and stop label. The value target therefore estimates continuation success under the current search procedure rather than graph-theoretic reachability. Fine-tuning on these search-induced targets defines the TriFOR$^+$ refinement.

3.3 Part III: DeepHopQA Construction

Existing KGQA benchmarks rarely provide gold reasoning paths, and shortest paths are often used as proxies. However, shortest paths often fail to provide valid evidence for a question-answer pair (Table 10). Moreover, as deeper multi-hop reasoning becomes increasingly important, existing datasets provide limited coverage of long-hop and branching retrieval. To complement these benchmarks, we construct DeepHopQA, a long-hop KGQA benchmark with LLM-validated gold reasoning paths. Construction details are in Section B.2.1. DeepHopQA is designed to evaluate whether a model can find gold reasoning paths in a controlled multi-hop KG environment. We start from question-specific subgraphs $\mathcal{G}_q = (\mathcal{E}_q, \mathcal{R}_q, \mathcal{T}_q)$ and their associated seed entities in the ComplexWebQuestions dataset (Talmor & Berant, 2018), and take their union to form a unified KG.

Constrained random walk. For each seed entity v_0, we sample N random-walk paths $\{p_{v_0}^{(i)}\}_{i=1}^{N}$ of length $L \in [1, 7]$. To avoid degenerate or uninformative hops, relations are categorized into schema-level, metadata-level, and normal factual relations. Schema relations are excluded entirely, and metadata relations are discouraged near the tail, ensuring that paths terminate at concrete answer entities. This yields a set of feasible candidate paths for each seed.

Filtering and LLM validation. Candidate paths are ranked using lightweight heuristics that favor informative relations, penalize hub-like intermediate entities, encourage relation diversity, and prefer specific terminal entities. From the candidate set $\{p_{v_0}^{(i)}\}_{i=1}^{N}$, an LLM selects a suitable reasoning path $p_{v_0}^\star$. Then conditioned on its seed entity v_0 and terminal entity $v_L^\star$, the LLM generates a natural-language question $\tilde{q}$ whose answer is $a = v_L^\star$, and validates the consistency of the resulting triple $(\tilde{q}, a, p_{v_0}^\star)$. Refer to Table 17 for examples of the generated triples.

Local subgraph construction. For each validated triple, we construct a question-specific subgraph $\mathcal{G}_{\tilde{q}}$ by bounded-hop expansion on the unified graph. The construction guarantees the inclusion of the gold reasoning path and additionally expands neighborhoods around gold entities to maintain sufficient branching. This results in subgraphs that preserve the intended multi-hop reasoning path while avoiding near-deterministic search behavior. Together, these procedures complete the construction of DeepHopQA, providing long-hop questions paired with LLM-validated gold reasoning paths and structured subgraphs for search-based reasoning.

4 Experiments

4.1 Setup

Datasets. WebQuestionsSP (WebQSP; Yih et al., 2015) contains 4,737 questions over a KG with 24.9M entities and 164.6M facts, while ComplexWebQuestions 1.1 (CWQ; Talmor & Berant, 2018) extends WebQSP with compositional constraints, yielding 34,689 multi-hop questions. Both datasets are built on the Freebase KG (Bollacker et al., 2008), where WebQSP requires at most 2 hops, while CWQ includes more complex queries requiring up to 4 hops. They are widely used KGQA testbeds, enabling controlled comparison with prior works. Following RoG (Luo et al., 2023), we use a subgraph format containing all triplets within the maximum reasoning hops from seed entities and adopt the same train/test splits as prior works. DeepHopQA complements these datasets as a controlled diagnostic benchmark for long-hop KG search, with LLM-validated gold reasoning paths of up to 7 hops and question-specific subgraphs. Details are in Appendix B.

Table 1: Selected results on WebQSP, CWQ, and DeepHopQA. For Rigel, TransferNet, UniKGQA, ChatGPT+CoT, and Nutrea, we used the reported results ("–" indicates not reported). All other methods here are averaged over three runs, except REKG-MCTS and RTSoG which are reported from a single run. The full table is provided in Table 18.

Type	Method	WebQSP		CWQ		DeepHopQA	
		H@1	F1	H@1	F1	H@1	F1
Embedding-based	Rigel	73.3	–	48.7	–	–	–
	TransferNet	71.4	–	48.6	–	–	–
Retrieval-augmented	UniKGQA	77.2	72.2	51.2	49.1	–	–
	ReaRev	76.3±0.2	71.8±0.2	55.7±0.2	50.8±0.5	64.6±0.1	53.7±0.0
LLMs	ChatGPT+CoT	75.6	–	48.9	–	–	–
	GPT-5-mini+CoT	79.5±0.0	51.7±0.1	52.5±0.5	43.7±0.0	9.9±0.1	9.9±0.1
LLMs+KGs	Nutrea	77.4	72.7	53.6	49.5	–	–
	RoG	79.8±0.4	70.0±0.2	56.2±0.2	54.1±0.1	–	–
GNN+LLMs	GNN-RAG	80.1±0.3	71.4±0.1	56.5±0.9	50.3±1.0	65.4±0.4	42.7±0.3
	GNN-RAG+RA	81.7±0.3	73.2±0.3	58.6±0.6	53.2±1.6	–	–
MCTS-based	REKG-MCTS	67.2	51.6	39.0	35.6	8.4	8.4
	RTSoG	73.1	43.4	29.6	18.7	49.7	43.8
	TriFOR (Ours)	83.0±0.4	73.7±0.1	62.1±0.2	57.2±0.5	67.2±0.2	54.3±0.3
	TriFOR$^+$ (Ours)	**83.3**±0.7	**74.0**±0.3	**63.0**±0.3	**57.6**±0.2	**67.6**±0.1	**55.4**±0.6

Baselines. We compare 27 baselines grouped into six categories: embedding-based, retrieval-augmented, LLM-only, LLM+KG, GNN+LLM, and MCTS-based methods. For RoG (Luo et al., 2023), ReaRev (Mavromatis & Karypis, 2022), GNN-RAG, GNN-RAG+RA (Mavromatis & Karypis, 2024), and RTSoG (Long et al., 2025), we rerun or adapt the released implementations under a unified evaluation setting. We implement GPT-5-mini+CoT (Wei et al., 2022) and re-implement REKG-MCTS (Song et al., 2025) because its release lacks a complete executable pipeline. Published results are reported for the remaining methods. For runtime comparison, we report latency for methods using an LLM during search or final answer generation, matching our focus on LLM-assisted KG reasoning. For GNN+LLM and MCTS-based methods, we use `Qwen-2.5-14B` as the backbone where applicable. Implementation and evaluation details are provided in Appendix D.1.

Evaluation Metrics. Following prior KGQA works, we report Hits@1 (H@1) and F1. Hits@1 measures whether the top-ranked predicted answer matches any gold answer, while F1 computes set-level overlap between predicted and gold answer sets to balance precision and recall under multi-answer supervision. We additionally report end-to-end inference latency, measured as end-to-end wall-clock time, for methods that invoke an LLM during inference or answer generation.

4.2 Results

Main Results. Table 1 shows that TriFOR outperforms all evaluated baselines in H@1/F1 across all three benchmarks, improving over the strongest baselines in the table by 1.3/0.5 percentage points on WebQSP, 3.5/3.1 on CWQ, and 1.8/0.6 on DeepHopQA. These gains are observed across benchmarks with substantially different hop distributions. TriFOR$^+$ provides additional improvements across all three benchmarks through MCTS-trajectory fine-tuning. Together, forecast-guided search achieves strong performance across standard and controlled long-hop KGQA settings.

Head Ablations. As shown in Table 3, removing any head substantially degrades H@1, confirming their complementary roles in search. The policy head prioritizes outgoing actions, the value head evaluates continuation utility, and the stop head models answer readiness for termination. Full H@1/F1 results are provided in Appendix E, with further ablations and analyses including termination strategies, search-controller comparisons, pruning, and sensitivity studies.

Table 3: Head ablations (H@1).

Setting	WebQSP	CWQ
w/o policy	60.6	43.6
w/o value	65.6	41.7
w/o stop	44.6	33.5
TriFOR	**83.0**	**62.1**

Table 2: Training and inference time for methods that use an LLM during inference or final answering. Locally hosted models were timed on a single NVIDIA RTX A6000, except REKG-MCTS and RTSoG, which were run with two GPUs and vLLM. GNN-RAG and GNN-RAG+RA used 1/10 training epochs on DeepHopQA due to the high per-epoch cost on longer-hop graphs; the reported training times correspond to this reduced-epoch configuration. N/A indicates inference-only methods.

Method	WebQSP		CWQ		DeepHopQA	
	Training	Inference	Training	Inference	Training	Inference
GNN-RAG	2h 10m	8h 44m	20h 1m	22h 2m	65h 26m*	18h 59m
GNN-RAG+RA	2h 10m	9h 58m	20h 1m	25h 31m	65h 26m*	–
GPT-5-mini+CoT	N/A	5h 6m	N/A	13h 22m	N/A	14h 27m
REKG-MCTS*	N/A	12h 51m	N/A	42h 40m	N/A	38h 33m
RTSoG*	N/A	26h 25m	N/A	79h 2m	N/A	57h 30m
TriFOR (Ours)	3h 36m	1h 52m	11h 9m	4h 15m	11h 44m	9h 36m
TriFOR$^+$ (Ours)	4h 36m	1h 53m	21h 2m	4h 23m	23h 52m	9h 40m

Efficiency and Runtime. Table 2 summarizes training and inference costs under the original configurations. By replacing in-loop LLM calls with lightweight forecaster evaluations, TriFOR is $6.9\times/14.2\times$ faster than REKG-MCTS/RTSoG on WebQSP and $10.0\times/18.6\times$ faster on CWQ, with small additional cost for TriFOR$^+$. On the same H200 and 100 questions per dataset, TriFOR remains $16.2\times/14.3\times$ faster than REKG-MCTS and $691\times/529\times$ faster than RTSoG on WebQSP/CWQ (Table 4). The LLM-call counts indicate that the gains primarily come from eliminating repeated in-loop LLM calls.

Table 4: Hardware-normalized runtime on a single H200, reported as *seconds (LLM calls)* per question.

Method	WebQSP	CWQ
REKG-MCTS	52.2 (15.1)	64.2 (23.0)
RTSoG	2,231 (308)	2,382 (326)
TriFOR	**3.23 (1.0)**	**4.50 (1.0)**

Robustness. We further evaluate whether TriFOR remains effective beyond dataset-specific training. First, we train a *single* model on the union of WebQSP, CWQ, and DeepHopQA and evaluate it on each test split. As shown in Table 5, the jointly trained model remains competitive with strong external dataset-specific baselines, with gaps within 2.3/5.8 percentage points in Hits@1/F1, indicating that a single model remains effective across heterogeneous training sources and hop regimes.

Table 5: Robustness of TriFOR under mixed training data.

Test	H@1	F1
WebQSP	79.9	67.4
CWQ	56.3	48.9
DeepHopQA	63.2	52.0

Cross-Dataset Transfer. We further evaluate transfer by training on one benchmark and directly testing on another without fine-tuning. ReaRev and GNN-RAG degrade substantially under these shifts, whereas TriFOR achieves the strongest performance among the evaluated methods across all three settings in Table 6, with gains of up to 17.6/23.6 percentage points in Hits@1/F1. These results show more consistent transfer of TriFOR's learned search guidance across Freebase benchmarks with different question distributions and hop regimes. Additional experiments and ablations are in Appendices E and F.

Table 6: Cross-dataset transfer of TriFOR (H@1/F1).

Method	DHQA →CWQ	CWQ →WEB	WEB →CWQ
ReaRev	2.6 / 2.1	52.0 / 42.1	16.7 / 15.4
GNN-RAG	36.6 / 22.2	68.3 / 48.2	36.8 / 22.6
TriFOR	**54.2 / 45.8**	**74.2 / 59.1**	**39.3 / 27.0**

5 Conclusion and Limitations

TriFOR is a forecast-guided PUCT-MCTS framework replacing in-loop LLM calls with lightweight policy, value, and stop forecasts for KGQA. Across benchmarks, it improves multi-hop QA while reducing inference cost, showing the effectiveness of learned forecasts. We also introduce DeepHopQA, a long-hop benchmark with LLM-validated gold paths, but limited to single-seed, single-answer questions. Our evaluation assumes provided seeds and subgraphs, uses constructed path supervision and synthetic negatives, and retains one final-answer LLM call. Despite these limitations, TriFOR remains effective across datasets and reasoning depths, supporting efficient and robust graph search.

AI USE STATEMENT

In this work, we used generative AI tools for synthetic-data construction, research methodology and experiment refinement, implementation and data processing, and result interpretation. In particular, GPT-4o-mini was used in the DeepHopQA construction pipeline for path selection, question generation, and validation, as described in the paper. We also used generative AI tools to assist with software code, literature and technical understanding, and drafting and editing parts of the manuscript. All AI-assisted code was manually reviewed and tested by the authors, and all generated or AI-assisted research content, analyses, and manuscript text were independently checked for correctness and consistency with the underlying experiments. The authors take full responsibility for the final content of this work, including all text, claims, code, data, and experimental results produced with the aid of generative AI.

ETHICS STATEMENT

Our human evaluation involved voluntary ratings by laboratory students and researchers who provided informed consent to participate and received no compensation. Participants evaluated only the answer validity, reasoning-path support, and naturalness of KGQA examples; no personally identifiable or sensitive information was collected. The evaluation data consist solely of these research ratings and are used only for assessing benchmark quality. We are not aware of additional material ethical risks arising from the proposed KGQA methodology or benchmark beyond those associated with the underlying public knowledge-graph data and generative models.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the TriFOR architecture, training objectives, search procedure, hyperparameters, dataset construction, baseline implementations, and evaluation protocols in the main paper and appendices. The supplementary material includes source code and the preprocessing, training, inference, and evaluation scripts needed to reproduce the reported experiments. We also report implementation details, controlled runtime settings, ablations, and DeepHopQA construction and validation procedures to facilitate independent verification. Due to submission file-size constraints, the full DeepHopQA data and trained checkpoints are not included in the current supplementary package; we will release these artifacts separately.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Þórunn Arnardóttir, Elías Bjartur Einarsson, Garðar Ingvarsson Juto, Thorvaldur Páll Helgason, and Hafsteinn Einarsson. Wikiqa-is: Assisted benchmark generation and automated evaluation of icelandic cultural knowledge in llms. In *Proceedings of the Third Workshop on Resources and Representations for Under-Resourced Languages and Domains (RESOURCEFUL-2025)*, pp. 64–73, 2025.

Jinheon Baek, Alham Fikri Aji, and Amir Saffari. Knowledge-augmented language model prompting for zero-shot knowledge graph question answering. *arXiv preprint arXiv:2306.04136*, 2023.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Neil Band, Xuechen Li, Tengyu Ma, and Tatsunori Hashimoto. Linguistic calibration of long-form generations. *arXiv preprint arXiv:2404.00474*, 2024.

Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. Constraint-based question answering with knowledge graph. In *Proceedings of COLING 2016, the 26th international conference on computational linguistics: technical papers*, pp. 2503–2514, 2016.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1533–1544, 2013.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250, 2008.

Hyeong Kyu Choi, Seunghun Lee, Jaewon Chu, and Hyunwoo J Kim. Nutrea: Neural tree search for context-guided multi-hop kgqa. *Advances in Neural Information Processing Systems*, 36: 35954–35965, 2023.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Hai Cui, Tao Peng, Feng Xiao, Jiayu Han, Ridong Han, and Lu Liu. Incorporating anticipation embedding into reinforcement learning framework for multi-hop knowledge graph question answering. *Information Sciences*, 619:745–761, 2023.

Preetam Prabhu Srikar Dammu, Himanshu Naidu, and Chirag Shah. Dynamic-kgqa: A scalable framework for generating adaptive question answering datasets. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 3498–3508, 2025.

Rajarshi Das, Manzil Zaheer, Dung Thai, Ameya Godbole, Ethan Perez, Jay-Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew McCallum. Case-based reasoning for natural language queries over knowledge bases. *arXiv preprint arXiv:2104.08762*, 2021.

Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *International semantic web conference*, pp. 69–78. Springer, 2019.

Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36:30039–30069, 2023.

Shengxiang Gao, Chao Lei, Jey Han Lau, and Jianzhong Qi. Pathise: Learning informative path supervision for knowledge graph question answering. *arXiv preprint arXiv:2605.10791*, 2026.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. Beyond iid: three levels of generalization for question answering on knowledge bases. In *Proceedings of the web conference 2021*, pp. 3477–3488, 2021.

Joshua Hare. Dealing with sparse rewards in reinforcement learning. *arXiv preprint arXiv:1910.09281*, 2019.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. Improving multi-hop knowledge base question answering by learning intermediate supervision signals. In *Proceedings of the 14th ACM international conference on web search and data mining*, pp. 553–561, 2021.

Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. G-retriever: Retrieval-augmented generation for textual graph understanding and question answering. *Advances in Neural Information Processing Systems*, 37:132876–132907, 2024.

Chaeyun Jang, Deukhwan Cho, Seanie Lee, Hyungi Lee, and Juho Lee. Reliable decision-making via calibration-oriented retrieval-augmented generation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38, 2023.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. *arXiv preprint arXiv:2212.00959*, 2022.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*, 2023.

Longquan Jiang and Ricardo Usbeck. Knowledge graph question answering datasets and their generalizability: are they enough for future research? In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 3209–3218, 2022.

Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, and Jiawei Han. Large language models on graphs: A comprehensive survey. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Yunshi Lan and Jing Jiang. Query graph generation for answering multi-hop complex questions from knowledge bases. In *Annual Meeting of the Association for Computational Linguistics*, 2020.

Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. A survey on complex knowledge base question answering: Methods, challenges and solutions. *arXiv preprint arXiv:2105.11644*, 2021.

Noah Lee, Jiwoo Hong, and James Thorne. Evaluating the consistency of llm evaluators. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 10650–10659, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.

Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. A survey on retrieval-augmented text generation. *arXiv preprint arXiv:2202.01110*, 2022.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, 2018.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. Few-shot in-context learning for knowledge base question answering. *arXiv preprint arXiv:2305.01750*, 2023.

Qian Lin, Junyi Li, and Hwee Tou Ng. Dynaquest: A dynamic question answering dataset reflecting real-world knowledge updates. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 26918–26936, 2025.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

Xiao Long, Liansheng Zhuang, Chen Shen, Shaotian Yan, Yifei Li, and Shafei Wang. Enhancing large language models with reward-guided tree search for knowledge graph question and answering. *arXiv preprint arXiv:2505.12476*, 2025.

Linhao Luo, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Reasoning on graphs: Faithful and interpretable large language model reasoning. *arXiv preprint arXiv:2310.01061*, 2023.

Elan Markowitz, Anil Ramakrishna, Jwala Dhamala, Ninareh Mehrabi, Charith Peris, Rahul Gupta, Kai-Wei Chang, and Aram Galstyan. Tree-of-traversals: A zero-shot reasoning algorithm for augmenting black-box language models with knowledge graphs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12302–12319, 2024.

Costas Mavromatis and George Karypis. Rearev: Adaptive reasoning for question answering over knowledge graphs. *arXiv preprint arXiv:2210.13650*, 2022.

Costas Mavromatis and George Karypis. Gnn-rag: Graph neural retrieval for large language model reasoning. *arXiv preprint arXiv:2405.20139*, 2024.

Costas Mavromatis, Petros Karypis, and George Karypis. Sempool: Simple, robust, and interpretable kg pooling for enhancing language models. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 154–166. Springer, 2024.

Alexander H Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents (emnlp16). *arXiv*, 2016.

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. Natural language inference by tree-based convolution and heuristic matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 130–136, 2016.

Abdelghny Orogat and Ahmed El-Roby. Smartbench: demonstrating automatic generation of comprehensive benchmarks for question answering over knowledge graphs. *Proceedings of the VLDB Endowment*, 15(12):3662–3665, 2022.

Abdelghny Orogat and Ahmed El-Roby. Maestro: Automatic generation of comprehensive benchmarks for question answering over knowledge graphs. *Proceedings of the ACM on Management of Data*, 1(2):1–24, 2023.

Abdelghny Orogat, Isabelle Liu, and Ahmed El-Roby. Cbench: Towards better evaluation of question answering over knowledge graphs. *arXiv preprint arXiv:2105.00811*, 2021.

Karl Pearson. The problem of the random walk. *Nature*, 72(1867):342–342, 1905.

Aleksandr Perevalov, Dennis Diefenbach, Ricardo Usbeck, and Andreas Both. Qald-9-plus: A multilingual dataset for question answering over dbpedia and wikidata translated by native speakers. In *2022 IEEE 16th International Conference on Semantic Computing (ICSC)*, pp. 229–234. IEEE, 2022.

Yunqi Qiu, Yuanzhuo Wang, Xiaolong Jin, and Kun Zhang. Stepwise reasoning for multi-relation question answering over knowledge graph with weak supervision. In *Proceedings of the 13th international conference on web search and data mining*, pp. 474–482, 2020.

Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

Amrita Saha, Vardaan Pahuja, Mitesh Khapra, Karthik Sankaranarayanan, and Sarath Chandar. Complex sequential question answering: Towards learning to converse over linked question answer pairs with a knowledge graph. In *Proceedings of the AAAI conference on artificial intelligence*, 2018.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 4498–4507, 2020.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

John Schulman, Barret Zoph, Christina Kim, Jacob Hilton, Jacob Menick, Jiayi Weng, Juan Felipe Ceron Uribe, Liam Fedus, Luke Metz, Michael Pokorny, et al. Chatgpt: Optimizing language models for dialogue. *OpenAI blog*, 2(4), 2022.

Priyanka Sen, Armin Oliya, and Amir Saffari. Expanding end-to-end question answering on differentiable knowledge graphs with intersection. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 8805–8812, 2021.

Tiesunlong Shen, Jin Wang, Xuejie Zhang, and Erik Cambria. Reasoning with trees: faithful question answering over knowledge graph. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 3138–3157, 2025.

Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search. *Advances in Neural Information Processing Systems*, 31, 2018.

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. Transfernet: An effective and transparent framework for multi-hop question answering over relation graph. *arXiv preprint arXiv:2104.07302*, 2021.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Xiaozhuang Song, Shufei Zhang, and Tianshu Yu. 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*, pp. 9288–9306, 2025.

Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 4231–4242, 2018.

Haitian Sun, Tania Bedrax-Weiss, and William Cohen. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 2380–2390, 2019.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel M Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. *arXiv preprint arXiv:2307.07697*, 2023.

Yawei Sun, Lingling Zhang, Gong Cheng, and Yuzhong Qu. Sparqa: skeleton-based semantic parsing for complex questions over knowledge bases. In *Proceedings of the AAAI conference on artificial intelligence*, pp. 8952–8959, 2020.

Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions. *arXiv preprint arXiv:1803.06643*, 2018.

Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V Chawla, and Panpan Xu. Graph neural prompting with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 19080–19088, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International semantic web conference*, pp. 210–218. Springer, 2017.

Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

Keheng Wang, Feiyu Duan, Sirui Wang, Peiguang Li, Yunsen Xian, Chuantao Yin, Wenge Rong, and Zhang Xiong. Knowledge-driven cot: Exploring faithful reasoning in llms for knowledge-intensive question answering. *arXiv preprint arXiv:2308.13259*, 2023.

Yingxu Wang, Shiqi Fan, Mengzhu Wang, Siyang Gao, Chao Wang, and Nan Yin. Damr: Efficient and adaptive context-aware knowledge graph question answering with llm-guided mcts. *arXiv preprint arXiv:2508.00719*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yike Wu, Nan Hu, Sheng Bi, Guilin Qi, Jie Ren, Anhuan Xie, and Wei Song. Retrieve-rewrite-answer: A kg-to-text enhanced llms framework for knowledge graph question answering. *arXiv preprint arXiv:2309.11206*, 2023.

Tianjun Yao, Haoxuan Li, Zhiqiang Shen, Pan Li, Tongliang Liu, and Kun Zhang. Learning efficient and generalizable graph retriever for knowledge-graph question answering. *arXiv preprint arXiv:2506.09645*, 2025.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 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, 2022.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1321–1331, 2015.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Wang, Zhiguo Wang, and Bing Xiang. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases. *arXiv preprint arXiv:2210.00063*, 2022.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. *arXiv preprint arXiv:2202.13296*, 2022a.

Liangliang Zhang, Zhuorui Jiang, Hongliang Chi, Haoyang Chen, Mohammed Elkoumy, Fali Wang, Qiong Wu, Zhengyi Zhou, Shirui Pan, Suhang Wang, et al. Diagnosing and addressing pitfalls in kg-rag datasets: Toward more reliable benchmarking. *arXiv preprint arXiv:2505.23495*, 2025a.

Qixuan Zhang, Xinyi Weng, Guangyou Zhou, Yi Zhang, and Jimmy Xiangji Huang. Arl: An adaptive reinforcement learning framework for complex question answering over knowledge base. *Information Processing & Management*, 59(3):102933, 2022b.

Xikun Zhang, Antoine Bosselut, Michihiro Yasunaga, Hongyu Ren, Percy Liang, Christopher D Manning, and Jure Leskovec. Greaselm: Graph reasoning enhanced language models for question answering. *arXiv preprint arXiv:2201.08860*, 2022c.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, et al. Siren's song in the ai ocean: A survey on hallucination in large language models. *Computational Linguistics*, pp. 1–46, 2025b.

Dongzhuoran Zhou, Yuqicheng Zhu, Xiaxia Wang, Hongkuan Zhou, Yuan He, Jiaoyan Chen, Steffen Staab, and Evgeny Kharlamov. What breaks knowledge graph based rag? benchmarking and empirical insights into reasoning under incomplete knowledge. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 2522–2538. Association for Computational Linguistics, 2026.

Terry Yue Zhuo, Yujin Huang, Chunyang Chen, and Zhenchang Xing. Exploring ai ethics of chatgpt: A diagnostic analysis. *ArXiv*, abs/2301.12867, 2023. URL `https://api.semanticscholar.org/CorpusID:256390238`.

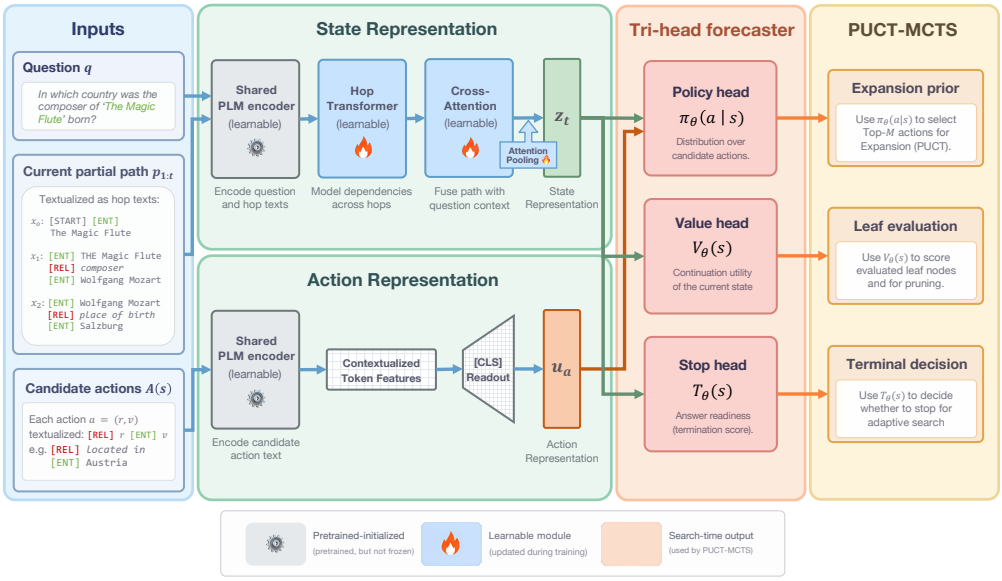

Figure 2: Overview of **TriFOR**'s state and action representation flow. TriFOR encodes partial reasoning paths and candidate actions into shared representations, which feed its policy, value, and stop heads to guide PUCT-MCTS expansion, evaluation, and termination.

A Related Work

A.1 Retrieval and Reasoning for KGQA

KGQA paradigms. Early knowledge graph question answering (KGQA) work can be broadly classified into semantic parsing approaches and retrieval-based approaches (Lan et al., 2021). Semantic parsing translates questions into executable logical forms (Sun et al., 2020; Lan & Jiang, 2020; Ye et al., 2022), but often requires expensive supervision and can fail on incomplete KGs or non-executable parses (Das et al., 2021; Yu et al., 2022). Retrieval-based methods instead score entities or paths and propagate question-conditioned signals over local subgraphs (Berant et al., 2013; Saxena et al., 2020; Zhang et al., 2022a; Mavromatis & Karypis, 2022), although dense GNN encoding can be costly and deeper propagation may suffer from oversmoothing (Li et al., 2018).

LLM-based KG reasoning. Graph-augmented language models that combine LMs with graph-structured knowledge have become a promising direction for KGQA (Jin et al., 2024). Previous works broadly fall into two categories: (i) injecting *latent* graph information into the LM (Zhang et al., 2022c; Tian et al., 2024), and (ii) inserting *verbalized* graph evidence into the LM input (Ye et al., 2022; Jiang et al., 2023). Latent-injection approaches can be limited by a modality mismatch between language and graphs, which may degrade performance on knowledge-intensive tasks (Mavromatis et al., 2024). Input-insertion approaches better leverage the LM's language understanding by providing explicit textual evidence (Wu et al., 2023; He et al., 2024; Mavromatis & Karypis, 2024), but can retrieve noisy or redundant graph facts at scale.

Sequential graph search. Beyond feed-forward retrieval, prior work has formulated graph traversal as sequential decision making, including reinforcement-learning approaches that learn policies to traverse from seed entities toward answers (Qiu et al., 2020; Zhang et al., 2022b; Cui et al., 2023). Such approaches naturally model path-dependent decisions but can face difficult credit assignment under sparse terminal rewards (Hare, 2019). M-Walk (Shen et al., 2018) further combines a recurrent neural policy and learned action values with MCTS for graph walking and knowledge-base completion, providing an early learned-MCTS formulation for graph traversal.

LLM-guided search for KGQA. Recent work also uses LLMs to guide explicit search over knowledge graphs. ToG (Sun et al., 2023) performs LLM-guided beam search, using the LLM to prune candidate relations and entities and to determine whether the retrieved paths provide sufficient

evidence for answering. Tree-of-Traversals (Markowitz et al., 2024) instead performs zero-shot best-first tree search over LLM-generated thoughts and KG-interface actions, with an LLM value function guiding node expansion and answer termination. These methods demonstrate the benefit of adaptive graph search, but repeatedly invoke an LLM for search decisions.

Learned MCTS for KGQA. More recent methods combine MCTS with learned or LLM-based search guidance (Shen et al., 2025; Long et al., 2025; Song et al., 2025; Wang et al., 2025). RwT (Shen et al., 2025) combines an LLM policy with a learned PLM evaluator in PUCT-based search, while DAMR (Wang et al., 2025) uses LLM-guided expansion together with a lightweight context-aware path evaluator and search-derived pseudo-path refinement. REKG-MCTS and RTSoG likewise rely on LLMs for internal search operations such as evaluation, rollout, expansion, or termination (Song et al., 2025; Long et al., 2025). Such in-loop LLM use can substantially increase inference cost and may introduce inconsistent evaluation across long search trajectories (Lee et al., 2025).

As summarized in Table 7, prior KGQA systems already combine structured search with learned or LLM-based guidance, and some additionally incorporate adaptive termination or search-derived refinement. TriFOR builds on this line but jointly models the signals for expansion, continuation evaluation, and termination with a lightweight forecaster, so no LLM is invoked at internal tree nodes. It further refines these signals using search trajectories, retaining adaptive tree search while amortizing semantic guidance into inexpensive forward passes. For TriFOR^{+}, search-derived refinement specifically distills MCTS visit distributions and empirical continuation success into the forecaster.

Table 7: Comparison with dynamic search-based KGQA methods. Refinement indicates whether search trajectories are reused to update a learned search-guidance model.

Method	Search	Expansion	Evaluation	Stop	In-loop LLM	Refinement
ToG	Beam	LLM pruning	LLM scoring	LLM sufficiency	✓	×
Tree-of-Traversals	Best-first	LLM actions	LLM value	Answer / threshold	✓	×
RwT	PUCT-MCTS	KG neighbors + LLM prior	Learned PLM	Terminal / max-hop	✓	✓
REKG-MCTS	UCB-MCTS	KG neighbors	LLM evaluator + rollout	Reward / depth	✓	×
RTSoG	SC-MCTS	LLM policy	LLM value	LLM EoS	✓	×
DAMR	UCT-MCTS	LLM planner	Transformer scorer	Answer / depth	✓	✓
TriFOR	PUCT-MCTS	**Learned policy**	**Learned value**	**Learned stop**	×	✓

Path-level supervision. Recent work also studies richer supervision than shortest-path labels for KG retrieval. RAPL (Yao et al., 2025) combines heuristic paths with LLM-rationalized labels, while PathISE (Gao et al., 2026) estimates relation-path informativeness from answer-level supervision. These methods focus on constructing path-level supervision, whereas TriFOR uses validated reasoning paths to train policy, continuation-value, and stop signals and to evaluate retrieved reasoning paths.

A.2 KGQA BENCHMARKS AND DATASET CONSTRUCTION.

Progress in KGQA has been driven by modeling advances and diverse benchmarks spanning different knowledge bases, formats, and reasoning demands. Early but still widely-used datasets were built on Freebase (Bollacker et al., 2008) by collecting web query questions and annotating gold answers with human workers (Yih et al., 2015; Bao et al., 2016; Talmor & Berant, 2018). Later benchmarks shifted to Wikidata and DBpedia with hybrid construction and human validation (Trivedi et al., 2017; Dubey et al., 2019), sometimes extending to multilingual or conversational evaluation (Saha et al., 2018; Perevalov et al., 2022) or targeting generalization (Gu et al., 2021; Jiang & Usbeck, 2022). Beyond proposing new benchmarks, several works meta-evaluate KGQA datasets via structural and linguistic properties (Orogat et al., 2021) or automate construction with rule-based frameworks (Orogat & El-Roby, 2022; 2023). More recent efforts use scalable generation, including LLM-assisted pipelines, but may introduce KG sparsity or hallucinated outputs (Dammu et al., 2025).

Existing KGQA benchmarks often lack gold reasoning paths, so shortest paths are used as proxies that can be insufficient for reasoning. Moreover, despite the widespread use of WebQSP and CWQ, recent works report substantial benchmark quality issues such as incomplete annotations, ambiguous or trivial questions, and outdated knowledge, with a low factual correctness rate (Zhang et al., 2025a). DeepHopQA complements these benchmarks by providing long-hop instances with LLM-validated gold reasoning paths, enabling explicit path-level supervision and verifiable evaluation.

Table 8: Seed entity count distribution for WebQSP and CWQ.

#Seed	0	1	2	3	4	5	Avg.
WebQSP	1	4,608	79	11	0	1	1.02
CWQ	8	18,576	13,943	2,066	84	12	1.53

Table 9: Split statistics and subgraph coverage with reachability.

Dataset	**Split**	*#Total*	*#Ans Exist*	*Ans Cov. (%)*	*Path Cov. (%)*
WebQSP	Train	2,826	2,715	96.07	95.82
	Valid	246	235	95.53	95.12
	Test	1,628	1,557	95.64	95.33
CWQ	Train	27,639	22,089	79.92	79.34
	Valid	3,519	2,848	80.93	80.59
	Test	3,523	2,840	80.66	80.12
DeepHopQA	Train	24,108	24,108	100.0	100.0
	Valid	3,013	3,013	100.0	100.0
	Test	3,015	3,015	100.0	100.0

B DATASETS

B.1 WEBQSP AND CWQ

We evaluate on two widely used Freebase-based KGQA benchmarks, WebQSP (Yih et al., 2015) and CWQ (Talmor & Berant, 2018). Both datasets are grounded on the same Freebase KG (Bollacker et al., 2008) that contains 164.6 million triplets and 24.9 million entities, but differ in query complexity and hop requirements. WebQSP mainly contains single-hop and simple multi-hop questions, and is treated as requiring at most 2-hop reasoning. CWQ extends WebQSP with compositional constraints (e.g., conjunction, comparison, nested logical structures), and typically requires up to 4 hops.

B.1.1 ANSWER COVERAGE AND REACHABILITY.

We use the RoG subgraph format (Luo et al., 2023), where each question is paired with a question-specific subgraph extracted around the provided seed entities, containing all triplets within the maximum reasoning-hop neighborhood of those seeds. In addition, the number of provided seed entities varies across questions (Table 8), directly affecting the size and connectivity of the extracted subgraph. However, in these datasets, the resulting subgraphs can be *incomplete* for some questions: the gold answer entities may be absent from the extracted subgraph, or they may appear but still be unreachable from the seed entities within the maximum hop (WebQSP ≤ 2, CWQ ≤ 4).

In Table 9, we report *#Ans Exist*, the number of questions whose extracted subgraph contains at least one gold answer entity. We define *Ans Cov.* as the percentage of questions whose subgraph contains at least one gold answer entity. We additionally report *Path Cov.*, the percentage of questions for which at least one gold answer entity is reachable from any seed entity within the hop budget. These statistics provide an upper bound on performance under this subgraph setting, since the correct answer cannot be supported by any reasoning path when no gold entity exists in the given subgraph. However, following previous works, we include all questions in evaluation for fair comparison, including those with zero seed entities and those whose extracted subgraphs contain no gold answer entity.

B.1.2 DATASET PREPROCESSING

We preprocess WebQSP and CWQ to attach explicit reasoning supervision in the form of *shortest paths* and LLM-validated *gold reasoning paths* for each question. Given a question q with seed entities $\mathcal{S}_q$, a question-specific subgraph $\mathcal{G}_q$, and the set of gold answer entities $\mathcal{Y}_q$, we run a multi-source BFS on $\mathcal{G}_q$ from all seeds in $\mathcal{S}_q$ to compute the shortest paths from any seed to each reachable answer entity. We denote the resulting set of shortest paths as $\mathcal{P}_q = \{\texttt{path}(y) \mid y \in \mathcal{Y}_q\}$, where each $\texttt{path}(y) = (v_0, \ldots, v_L)$ is a shortest path in $\mathcal{G}_q$ from some $v_0 \in \mathcal{S}_q$ to $v_L = y$, if any. This returns a shortest path whenever a gold answer entity exists in $\mathcal{G}_q$ and is reachable from the seeds.

Table 10: An example where the shortest path is semantically invalid, while an LLM-validated gold reasoning path provides valid evidence.

Seed	Question	Paths (Shortest vs. Gold)	Answer
Mary	What does the religion, whose figure is **Mary**, call God?	**Shortest path (rejected):** **Mary** →`people.person.children` →**Jesus Christ** **Gold reasoning path (accepted):** **Mary** →`base.saint.venerated_in` →Protestantism →`religion.religion.deities` →**Jesus Christ**	Jesus Christ

Table 11: Effect of path-supervision type on TriFOR pretraining.

Data	Training supervision	H@1	F1
WebQSP	Shortest-only	80.0	71.1
	Validated + fallback	82.4	73.4
	Validated-only	**83.0**	**73.7**
CWQ	Shortest-only	58.8	53.4
	Validated + fallback	60.9	56.6
	Validated-only	**62.1**	**57.2**

However, a shortest path is not necessarily a semantically valid explanation for q, as illustrated in Table 10. To filter such cases and obtain a set of gold reasoning paths, we employ an LLM-based verifier that judges whether a candidate path $\texttt{path}(y) \in \mathcal{P}_q$ supports the answer y for the given question q. We first verify the shortest path; if rejected, we enumerate additional candidate paths from $\mathcal{S}_q$ to y within a hop limit and validate them until a gold reasoning path is found or the candidate budget is exhausted. We denote the resulting set of accepted paths as $\tilde{\mathcal{P}}_q$. Finally, we store both $\mathcal{P}_q$ and $\tilde{\mathcal{P}}_q$ as additional attributes of each instance and use them to construct the offline synthetic supervision for forecaster pretraining, shown in Algorithm 1. We use `GPT-4o-mini` as the LLM-based verifier.

Since the verifier rejects semantically invalid paths, LLM-validated supervision contains fewer training paths than shortest-path supervision. However, this reduction reflects a quality-control step rather than a loss of useful evidence: rejected shortest paths are often weak or misleading reasoning signals. We therefore analyze whether higher-quality validated paths provide better supervision despite the smaller training set.

After LLM validation, validated gold paths cover 88.53% of WebQSP and 78.73% of CWQ training instances; the remaining 11.47% and 21.27% correspond to shortest-path fallback candidates. Thus, the verifier removes a non-trivial but limited fraction of shortest paths that are likely to provide semantically weak or misleading supervision. This suggests that LLM validation improves the quality of path supervision while retaining most of the available training coverage. We next evaluate whether this higher-quality validated supervision improves TriFOR pretraining despite using fewer paths.

As shown in Table 11, shortest-only supervision performs worst, confirming that shortest paths can provide semantically noisy reasoning signals. Validated-only supervision achieves the best performance on both WebQSP and CWQ, even though it uses fewer training paths than the fallback-based variant. These results suggest that higher-quality path supervision is more beneficial than simply increasing the number of training paths, motivating our use of LLM-validated gold paths for TriFOR pretraining. At the same time, the strong shortest-only performance shows that TriFOR remains effective even when such validated paths are unavailable.

Path supervision and outcome supervision. TriFOR uses reasoning paths to provide step-level supervision for its three forecasts: next-action targets for the policy head, intermediate-state targets for the value head, and termination targets for the stop head. By contrast, answer labels provide only terminal outcome supervision without identifying the intermediate decisions associated with that outcome. Accordingly, answer-only training would require a different credit-assignment objective from the path-supervised formulation used here.

B.2 DEEPHOPQA

B.2.1 CONSTRUCTION DETAILS

We describe the detailed pipeline used to construct **DeepHopQA**, a controlled diagnostic benchmark for long-hop KG search. Each instance pairs a path-grounded question with an LLM-validated gold reasoning path and a question-specific subgraph, enabling path-level supervision and analysis of deep search behavior. The construction consists of four stages: constrained random walk for candidate path sampling, heuristic path filtering, LLM-based selection/generation/validation, and local subgraph construction that guarantees gold reasoning path inclusion while preserving sufficient branching.

Constrained random walk. We begin by forming a unified directed knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ as the union of question-specific local subgraphs from the CWQ training, validation, and test splits. Let $\mathcal{S}$ denote the set of seed entities appearing in CWQ instances. For each seed entity $v_0 \in \mathcal{S}$, we perform constrained random walks (Pearson, 1905) on $\mathcal{G}$ to sample candidate reasoning paths. A path is represented as a sequence

$$p_{v_0} = (v_0, r_1, v_1, \ldots, r_L, v_L),$$

where $L \in [1, 7]$ is the hop length. To avoid degenerate or ontological chains, we partition relations into (i) schema/ontology/meta relations and (ii) normal factual relations. Concretely, we define a meta-relation subset $\mathcal{R}_{\text{meta}} \subseteq \mathcal{R}$ that collects schema-level and Freebase-internal edges (e.g., typing, domain/range, and other ontology relations). We enforce that the last two relations are non-meta, i.e., $r_{L-1}, r_L \notin \mathcal{R}_{\text{meta}}$ for $L \geq 2$, which prevents paths from terminating with schema-level statements rather than concrete answer entities. For each seed v_0, we draw N_{walk} random-walk paths and collect them as a candidate pool $\mathcal{P}_{v_0} = \{p_{v_0}^{(i)}\}_{i=1}^{N_{\text{walk}}}$.

Heuristic path scoring and pruning. Since raw random walks can be noisy and highly redundant, we first rank candidate paths using lightweight heuristics, retaining only a small top-K subset per seed as input to an LLM-based selector. We score each candidate path $p \in \mathcal{P}_{v_0}$ via a weighted sum:

$$\text{score}(p) = \alpha s_{\text{rel}}(p) + \beta s_{\text{hub}}(p) + \gamma s_{\text{div}}(p) + \delta s_{\text{len}}(p) + \eta s_{\text{spec}}(v_L) \tag{7}$$

where each component encourages interpretability and answerability. $s_{\text{rel}}(p)$ is the average inverse document frequency (IDF) (Sparck Jones, 1972) of the relations along p, so that rare and more informative relations are preferred. $s_{\text{hub}}(p)$ penalizes hub-like intermediate nodes using a degree-based term, e.g., $1/\log(\deg(v) + 2)$ averaged over intermediate nodes, discouraging paths that route through overly generic entities. $s_{\text{div}}(p)$ encourages the relation diversity, measured as the ratio of distinct relations to L, penalizing repetitive patterns. $s_{\text{len}}(p)$ softly prefers longer paths within the allowed range to promote long-hop reasoning. Finally, $s_{\text{spec}}(v_L)$ is a terminal specificity score that favors concrete endpoints using the same degree-based proxy as $s_{\text{hub}}(p)$, encouraging specific entities rather than abstract nodes of high-degree. We use $(\alpha, \beta, \gamma, \delta, \eta) = (0.3, 0.2, 0.2, 0.3, 0.2)$, sample $N_{\text{walk}} = 100$ paths per seed, and retain the top $K = 10$ candidates.

For each seed v_0, we keep the top-K paths by $\text{score}(p)$ and denote the retained set as $\tilde{\mathcal{P}}_{v_0} \subset \mathcal{P}_{v_0}$. This heuristic pruning is only used to reduce the candidate set size; the final decision is delegated to an LLM that uses language understanding to judge whether a path can support a meaningful multi-hop question.

LLM-based path selection, question generation, and validation. We then apply an LLM-based pipeline with three steps: (i) path selection, (ii) question generation, and (iii) validation. Given $\tilde{\mathcal{P}}_{v_0}$, the selector prompt asks the LLM to choose exactly one path $p_{v_0}^\star = (v_0, r_1^\star, v_1^\star, \ldots, v_L^\star) \in \tilde{\mathcal{P}}_{v_0}$ that is suitable for constructing a multi-hop question, enforcing constraints such as: the terminal entity should be a concrete instance, the path should avoid abstract endpoints, and the resulting question should require the *entire* path to answer rather than an early hop. Conditioned on the selected path $p_{v_0}^\star$, the seed entity v_0, and the terminal entity $v_L^\star$, the generator prompt asks the LLM to produce a path-grounded natural-language question $\tilde{q}$ whose answer is exactly $a = v_L^\star$. Finally, a validator prompt checks the logical consistency of the triple $(\tilde{q}, a, p_{v_0}^\star)$ and rejects samples with missing constraints, shortcut questions, or semantic mismatches. Only validated triples are retained for dataset construction. We use `GPT-4o-mini` for all LLM-based steps in this pipeline as it provides a practical trade-off between instruction-following quality and scalability for structured generation and validation. We further analyze sensitivity to alternative validators and generators in Section B.2.3.

Quality control and construction statistics. Beyond LLM-based semantic validation, each generated instance must pass three deterministic checks: the seed entity must appear verbatim in the question, the generated answer must exactly match the terminal entity, and the terminal entity must not appear in the question. Table 12 summarizes the stage-wise reduction or failure rates. The relatively large reduction during path scoring reflects deliberate candidate pruning rather than generation failure.

Table 12: Stage-wise reduction or failure rates in the DeepHopQA construction.

Procedure	Rate
Random walk	6.4%
Path scoring/pruning	56.2%
LLM sel./gen./val.	54.9%
Subgraph construction	∼0%

Only structurally promising paths are retained before LLM-based selection, generation, and validation. After an instance passes semantic and deterministic checks, subgraph construction introduces essentially no additional failures.

Local subgraph construction. For each validated triple $(\tilde{q}, a, p^{\star}_{v_0})$, we construct a question-specific subgraph $\mathcal{G}_{\tilde{q}}$ by running a bounded-hop BFS expansion on the unified graph $\mathcal{G}$, starting from the seed entity v_0. We ensure that the entire gold reasoning path $p^{\star}_{v_0}$ is included in $\mathcal{G}_{\tilde{q}}$ by explicitly injecting all path edges into the subgraph. Beyond simple inclusion, we additionally expand local neighborhoods around nodes on the gold reasoning path to maintain sufficient branching, preventing near-deterministic search behavior. This design yields subgraphs that preserve the intended multi-hop reasoning chain while still requiring non-trivial exploration for search-based retrievers.

The resulting dataset consists of samples in the form $\{(\tilde{q}, a, p^{\star}_{v_0}, \mathcal{G}_{\tilde{q}})\}$, where each example includes a natural-language question, a concrete answer entity, an LLM-validated gold reasoning path, and a question-specific KG subgraph that supports multi-hop retrieval and reasoning.

B.2.2 DeepHopQA Structural Analysis

Overlap audit. Because DeepHopQA is constructed from CWQ-derived graph material, we audit direct overlap between DeepHopQA training instances and the CWQ test set used in the transfer evaluation. Exact question overlap is 0.00%, and exact gold-path overlap is only 0.11%, indicating negligible direct duplication between source training instances and target evaluation examples. We therefore interpret DeepHopQA→CWQ in Table 6 as cross-benchmark transfer within related Freebase graph material rather than transfer to a fully independent unseen KG.

Path-length audit. Because question-specific subgraphs contain branching alternatives, we distinguish the annotated gold-path length from the shortest graph-theoretic route to the answer. Although 38.5% of instances contain a structurally shorter route, only 19.9% admit a shorter path that also satisfies the question under semantic validation, with an average reduction of 1.6 hops. This gap shows that shortest answer reachability does not necessarily provide question-relevant evidence. Accordingly, we interpret DeepHopQA hop length as the length of the validated gold evidence path and use it as a controlled diagnostic of search over longer semantically grounded evidence chains.

Shortcut-controlled evaluation. To test whether shorter valid alternatives materially drive the benchmark results, we additionally reevaluate the compared methods on a strict subset where no shorter semantically valid path is detected by the path validator. In this diagnostic, the ranking of the evaluated methods remains unchanged, with absolute Hits@1 shifts of at most 1.4 percentage points overall and less than 0.8 points for TriFOR. This stability indicates that the comparative results are not primarily driven by instances containing validated shorter alternatives.

B.2.3 Validator and Generator Sensitivity

LLM-assisted benchmark construction has been increasingly used recently for KG-driven automatic generation (Orogat & El-Roby, 2022; 2023), LLM-assisted, grounded KGQA construction (Dammu et al., 2025), and automated or document-grounded pipelines beyond KGQA (Lin et al., 2025; Arnardóttir et al., 2025). At the same time, such pipelines may introduce model-specific biases if examples are generated or filtered by a single LLM. DeepHopQA mitigates this issue through a graph-grounded construction pipeline: candidate paths are first sampled from the KG, filtered by structural heuristics, and then passed to the LLM only for path selection, question generation, and consistency validation.

To examine validator sensitivity, we re-evaluate DeepHopQA triples using multiple validators. As shown in Table 13, the acceptance ratios are broadly comparable across validators, suggesting that the validation outcome is not uniquely tied to a single verifier. This supports the role of graph-grounded path construction and explicit consistency checking in reducing validator-specific artifacts.

Table 13: Acceptance ratios under different validators on DeepHopQA.

Validator	Accept ratio (%)
GPT-4o-mini	77.6
Gemini 3 Flash Preview	80.3
Phi-4	86.6
GPT-5-mini	87.2

We further test whether the benchmark trend depends on the specific LLM used during construction. For this purpose, we reconstruct a DeepHopQA variant using Qwen3-8B as the backbone LLM for the construction pipeline and re-evaluate representative baselines. As shown in Table 14, the overall trend is preserved: TriFOR remains the strongest method, outperforming both GNN-based and MCTS-based baselines. This changes the construction backbone, not just the final verifier, providing a stronger check against generator-specific artifacts. These results suggest that DeepHopQA captures challenging graph-search behavior beyond artifacts of a particular construction LLM.

Table 14: Results on the DeepHopQA variant reconstructed with Qwen3-8B.

Method	H@1	F1
ReaRev	51.0	40.0
GPT-5-mini+CoT	4.78	4.78
GNN-RAG	52.3	42.3
REKG-MCTS	3.91	4.12
RTSoG	21.9	15.9
TriFOR (Ours)	**57.6**	**52.4**

B.2.4 Human Validation

We further assess DeepHopQA quality through human evaluation. For CWQ and WebQSP, we exclude instances whose paths do not reach a gold answer, since the path support cannot be meaningfully assessed; DeepHopQA requires no such filtering. We then sample 90 examples from each dataset, with every example evaluated by more than three annotators and 11 human evaluators overall. Annotators rate whether the answer fits the question (answer validity), the path sufficiently justifies the answer (path support), and the question is clear and natural (naturalness) on a five-point scale.

Table 15: Human evaluation results, reported as mean ratings with 95% confidence intervals.

Dataset	Answer validity	Path support	Naturalness
CWQ	4.30 [4.14, 4.44]	3.89 [3.69, 4.07]	4.47 [4.34, 4.59]
WebQSP	4.50 [4.36, 4.61]	4.38 [4.23, 4.52]	4.45 [4.30, 4.57]
DeepHopQA	4.39 [4.28, 4.50]	4.16 [4.03, 4.28]	4.41 [4.32, 4.49]

As shown in Table 15, DeepHopQA receives consistently high ratings across all three criteria. Its answer validity and naturalness are comparable to WebQSP and CWQ, while reasoning-path support remains strong despite substantially longer reasoning chains. These results provide human evidence that DeepHopQA answers are valid, the annotated paths meaningfully support them, and the questions remain clear and natural even under longer-hop reasoning.

B.2.5 Statistics and Examples

Table 9 summarizes the split sizes and subgraph coverage. Unlike WebQSP and CWQ, DeepHopQA is constructed to guarantee every instance has a valid gold answer entity and at least one supporting gold reasoning path within the hop budget. Consequently, both answer existence and reachability are 100% for all splits. In addition, Table 16 reports the hop-length distribution of DeepHopQA. Our construction explicitly targets longer gold evidence paths, resulting in a distribution slightly skewed toward longer annotated paths. This enables controlled evaluation of retrieval and search over longer semantically grounded evidence chains.

Table 16: DeepHopQA gold reasoning path length distribution.

Length	#Examples	Ratio (%)
1	3,574	11.86
2	4,086	13.56
3	4,421	14.67
4	5,293	17.56
5	8,073	26.79
6	3,221	10.69
7	1,468	4.87

DeepHopQA also differs from existing benchmarks in supervision structure. Each instance is generated from a *single* seed entity (single source) and has a *single* terminal answer entity (single answer), paired with an LLM-validated gold reasoning path. Accordingly, DeepHopQA does not cover the multi-seed or multi-answer cases, and results should be interpreted under this single-seed, single-answer setting. Table 17 shows representative DeepHopQA instances produced by our pipeline, where an LLM selects a reasoning path and generates a consistent natural-language question whose answer is the terminal entity of the path.

C BASELINES

We provide brief method descriptions on 27 baselines grouped into the following six categories: embedding-based, retrieval-augmented, LLM-only, LLM+KG, GNN+LLM, and MCTS-based methods. Since Table 1 in the main paper omits several methods due to space constraints, we provide the complete results table for all 27 baselines in Table 18.

Embedding-based Methods. These methods primarily rely on learned KG embeddings and differentiable matching over entities and relations, typically without explicit iterative subgraph expansion at inference.

- **KV-Mem** (Miller et al., 2016): Stores KG facts in a memory (key–value) structure and answers questions by attending relevant memory slots; prediction by retrieval over stored memory.
- **EmbedKGQA** (Saxena et al., 2020): Uses pre-trained KG embeddings to support multi-hop QA by scoring candidate entities through embedding-based composition of question over relation and entity representations.
- **TransferNet** (Shi et al., 2021): Transfers question semantics to relation path representations and ranks candidate paths; emphasizes transparent path scoring over relational graphs.
- **Rigel** (Sen et al., 2021): Targets questions involving multiple entities by implementing set intersection operation, which performs reasoning on multiple entities questions.

Retrieval-Augmented Methods. These methods retrieve a question-specific subgraph or a set of candidate paths and then perform neural reasoning to rank answer entities.

- **GraftNet** (Sun et al., 2018): Builds a question-conditioned heterogeneous graph of KG facts and text nodes, and applies directed propagation method to follow paths starting from seed nodes.
- **PullNet** (Sun et al., 2019): Iteratively expands a question subgraph by "pulling" relevant facts from a KB, enabling multi-step retrieval before final reasoning over the expanded evidence.
- **NSM** (He et al., 2021): Neural-symbolic multi-hop KBQA model that learns to select relations and entities step-by-step under weak answer supervision, using teacher–student intermediate supervision to guide reasoning.
- **SR+NSM** (Zhang et al., 2022a): Improves NSM by adding a subgraph retrieval to increase evidence coverage, then runs a subsequent reasoning process on the retrieved subgraph.
- **UniKGQA** (Jiang et al., 2022): Unifies retrieval and reasoning by using a PLM-based question–relation semantic matching module and a directed KG propagation module, trained with shared pretraining and stage-specific fine-tuning.
- **ReaRev** (Mavromatis & Karypis, 2022): Iteratively refines the question instructions using KG-aware feedback and executes them via a GNN-based traversal that dynamically chooses the instruction order to reach answer entities.

LLM-only Methods. These baselines answer questions directly from the LLM without explicit KG retrieval at inference time.

- **Flan-T5-xl** (Chung et al., 2024): Instruction-finetuned LLM used for direct answer generation from the question.

Table 17: Examples of DeepHopQA instances by hop length (1–6).

Hop	Seed	Question & Gold path	Answer
1	The New Atlantis	**Question:** What is the ISBN of the book edition of **The New Atlantis**? **Gold Path:** **The New Atlantis** → `book_edition.isbn` → **9780486430829**	9780486430829
2	Cara Silverman	**Question:** Who is the person responsible for the film set decoration of the film edited by **Cara Silverman**? **Gold Path:** **Cara Silverman** → `film.editor.film` → There's Always Woodstock → `film_set_decoration_by` → **Kris Fuller**	Kris Fuller
3	Sean Stackley	**Question:** Which color is associated with the sports team located in the birthplace of **Sean Stackley**? **Gold Path:** **Sean Stackley** → `person.place_of_birth` → Baltimore → `sports_team_location.teams` → Baltimore Bandits → `sports_team.colors` → **White**	White
4	Rapa Nui Language	**Question:** Which tourist attraction can be found in the region that contains the main country of the **Rapa Nui Language**? **Gold Path:** **Rapa Nui Language** → `human_language.main_country` → Chile → `country.second_level_division` → Magallanes Province → `location.containedby` → Patagonia → `tourist_attractions` → **Fitz Roy**	Fitz Roy
5	Arlington Memorial Bridge	**Question:** Which architectural style is associated with the museum directed by the individual linked to the **Arlington Memorial Bridge**'s architecture firm? **Gold Path:** **Arlington Memorial Bridge** →`architecture_firm` →McKim, Mead & White →`architecture_firm.project` →Metropolitan Museum of Art →`architecture.museum.director` →Thomas P. Campbell →`museum_director.museum` →The Cloisters →`structure.architectural_style` →**Medieval architecture**	Medieval architecture
6	John B. Mansbridge	**Question:** Who is the director of the play composed by the composer associated with the film designer **John B. Mansbridge**'s film set decorator? **Gold Path:** **John B. Mansbridge** →`films_production_designed` →The Ghosts of Buxley Hall →`film_set_decoration_by` →Norman Rockett →`film_sets_designed` →Rhinoceros →`film.film.music` →Galt MacDermot →`composer.plays_composed` →Two Gentlemen of Verona →`theater_production.director` →**Charles Newell**	Charles Newell

* For readability, some long relation names in the gold reasoning paths are truncated.

- **Alpaca-7B** (Dubois et al., 2023): Instruction-following model fine-tuned from LlaMA-7B via self-instruct techniques.
- **Llama2-Chat-7B** (Touvron et al., 2023): Chat-tuned LLM used for direct QA generation.

- **ChatGPT** (Schulman et al., 2022): A closed-source conversational LLM, based on GPT architecture.
- **ChatGPT+CoT, GPT-5-mini+CoT** (Wei et al., 2022): Uses chain-of-thought prompting to elicit intermediate reasoning steps before the final answer, improving performance on compositional questions.

LLM+KG Methods. These methods explicitly incorporate KG evidence into the LLM reasoning process, typically by retrieving KG facts and injecting them into prompts or using the LLM to plan hop-by-hop evidence acquisition.

- **KD-CoT** (Wang et al., 2023): Turns CoT into multi-round, structured QA where the LLM retrieves external knowledge each step to verify and revise its reasoning trace.
- **StructGPT** (Jiang et al., 2023): Improves zero-shot QA over structured data by iteratively "reading" relevant evidence and then "reasoning" over the linearized evidence with invoking–linearization–generation.
- **KB-BINDER** (Li et al., 2023): Uses an LLM to draft logical forms from demonstrations and then binds them to executable queries by grounding them to the KB via BM25-based schema/item matching.
- **Nutrea** (Choi et al., 2023): Probes unreached subtrees to incorporate broader KG context and uses RF-IEF embeddings to better represent ambiguous or uninformative entities during path selection.
- **ToG+ChatGPT** (Sun et al., 2023): Iteratively performs beam search over a KG to explore and rank promising paths, improving traceable and correctable KG-grounded reasoning.
- **RoG** (Luo et al., 2023): LLM generates relation-path plans, retrieves valid KG reasoning paths using these plans, and then reasons over the retrieved paths to produce faithful answers.

GNN+LLM Methods. These methods use a GNN as a retrieval and reasoning module over KG subgraphs and then feed graph-derived evidence into an LLM for final answer generation.

- **G-Retriever** (He et al., 2024): Retrieves a question-relevant subgraph from a textual graph and uses it as RAG context to ground LLM answers and reduce hallucinations.
- **GNN-RAG** (Mavromatis & Karypis, 2024): Performs GNN-based propagation over a question subgraph to retrieve candidate answers and supporting shortest-path evidence, then verbalizes the evidence for LLM answering.
- **GNN-RAG+RA** (Mavromatis & Karypis, 2024): Extends GNN-RAG with the retrieval augmentation of LLM module to improve coverage and robustness under long-hop reasoning.

MCTS-based Methods. These baselines formulate KG reasoning as tree search over actions and use MCTS-style exploration to discover high-reward reasoning trajectories.

- **REKG-MCTS** (Song et al., 2025): Uses training-free MCTS to explore KG reasoning trajectories guided by LLM/heuristic priors, balancing exploration and exploitation to find high-quality evidence paths.
- **RTSoG** (Long et al., 2025): Enhances LLM-based KG reasoning with reward-guided tree search, iteratively refining multi-hop reasoning paths and selecting the final answer based on searched evidence.

Matched final-answer LLM comparison results. Because reproduced GNN-RAG results can depend on the final-answer LLM, we additionally evaluate GNN-RAG and GNN-RAG+RA using the ChatGPT (`gpt-3.5-turbo`) setting adopted in the original GNN-RAG evaluation, while retaining the same RoG graph inputs and evidence-grounded output constraint. As shown in Table 19, the matched setting substantially strengthens the reproduced GNN-RAG baselines, yet TriFOR remains the strongest method on both datasets. This confirms that its advantage persists under the final-answer LLM setting.

Table 19: Matched ChatGPT comparison (H@1).

Method	WebQSP	CWQ
GNN-RAG	84.2	62.9
GNN-RAG+RA	86.8	64.0
TriFOR	**87.7**	**65.2**

Table 18: Full results on WebQSP, CWQ, and DeepHopQA. Published results are taken from the original papers, while reproduced results (RoG, ReaRev, GPT-5-mini+CoT, GNN-RAG, GNN-RAG+RA) are averaged over three runs and reported as mean ± standard deviation. REKG-MCTS and RTSoG are reproduced results from a single run. "–" indicates not reported.

Type	Method	WebQSP		CWQ		DeepHopQA	
		H@1	F1	H@1	F1	H@1	F1
Embedding-based	KV-Mem	46.7	38.6	18.4	–	–	–
	EmbedKGQA	66.6	–	45.9	–	–	–
	TransferNet	71.4	–	48.6	–	–	–
	Rigel	73.3	–	48.7	–	–	–
Retrieval-augmented	GraftNet	66.7	62.4	36.8	32.7	–	–
	PullNet	68.1	–	45.9	–	–	–
	NSM	68.7	62.8	47.6	42.4	–	–
	SR+NSM	69.5	64.1	50.2	47.1	–	–
	UniKGQA	77.2	72.2	51.2	49.1	–	–
	ReaRev	76.3 ±0.2	71.8 ±0.2	55.7 ±0.2	50.8 ±0.5	64.6 ±0.1	53.7 ±0.0
LLMs	Flan-T5-xl	31.0	–	14.7	–	–	–
	Alpaca-7B	51.8	–	27.4	–	–	–
	Llama2-Chat-7B	64.4	–	34.6	–	–	–
	ChatGPT	66.8	–	39.9	–	–	–
	ChatGPT+CoT	75.6	–	48.9	–	–	–
	GPT-5-mini+CoT	79.5 ±0.0	51.7 ±0.1	52.5 ±0.5	43.7 ±0.0	9.9 ±0.1	9.9 ±0.1
LLMs+KGs	KD-CoT	68.6	52.5	55.7	–	–	–
	StructGPT	72.6	–	–	–	–	–
	KB+BINDER	74.4	–	–	–	–	–
	Nutrea	77.4	72.7	53.6	49.5	–	–
	ToG+ChatGPT	76.2	–	58.9	–	–	–
	RoG	79.8 ±0.4	70.0 ±0.2	56.2 ±0.2	54.1 ±0.1	–	–
GNN+LLMs	G-Retriever	70.1	–	–	–	–	–
	GNN-RAG	80.1 ±0.3	71.4 ±0.1	56.5 ±0.9	50.3 ±1.0	65.4 ±0.4	42.7 ±0.3
	GNN-RAG+RA	81.7 ±0.3	73.2 ±0.3	58.6 ±0.6	53.2 ±1.6	–	–
MCTS-based	REKG-MCTS	67.2	51.6	39.0	35.6	8.4	8.4
	RTSoG	73.1	43.4	29.6	18.7	49.7	43.8
	TriFOR (Ours)	83.0 ±0.4	73.7 ±0.1	62.1 ±0.2	57.2 ±0.5	67.2 ±0.2	54.3 ±0.3
	TriFOR^{+} (Ours)	**83.3** ±0.7	**74.0** ±0.3	**63.0** ±0.3	**57.6** ±0.2	**67.6** ±0.1	**55.4** ±0.6

D Implementation Details

D.1 Baselines

For controlled comparison, we evaluate all reproduced methods on the official WebQSP and CWQ splits, including instances with formatting irregularities, missing seeds, or answers absent from the provided subgraph. Methods receive the same question-specific graph inputs, and LLM-assisted baselines use a standardized, evidence-grounded final-answer setup so that performance reflects the retrieved graph evidence rather than unconstrained parametric knowledge of the LLM. These choices prioritize comparability across methods, although reproduced scores can differ from those originally reported under method-specific preprocessing, prompting, or backbone LLMs.

Some released implementations omit subsets of WebQSP and CWQ because of malformed or incomplete instances. We retain all examples and compute metrics over the full evaluation set for every reproduced method to avoid differences caused by dataset filtering. Method-specific implementation details are described below.

ReaRev. We evaluate ReaRev (Mavromatis & Karypis, 2022) on WebQSP, CWQ, and DeepHopQA with a unified configuration and dataset-specific search/instruction parameters. We keep the entity embedding dimension and the number of adaptation stages fixed to 50 and 2, respectively, and adjust the number of instructions and GNN reasoning steps per dataset as summarized in Table 20. All remaining hyperparameters and training details follow the original ReaRev model setup. For DeepHopQA, we reduce the number of training epochs to 10 because each epoch takes more than 5 hours on a single RTX A6000 GPU, making 100-epoch training exceed 500 GPU-hours and impractical under comparable compute constraints.

Table 20: The dimension of entity embeddings and the number of adaptation stages are fixed across datasets, and all other settings follow Mavromatis & Karypis (2022).

Hyperparameter	**WebQSP**	**CWQ**	**DeepHopQA**
entity dimensions	50	50	50
adaptation stages T	2	2	2
instructions K	2	4	7
reasoning steps L	2	4	7
epochs	100	100	10
batch size	8	8	3

RoG. RoG (Luo et al., 2023) is originally trained and released for WebQSP and CWQ, so we do not run RoG inference on DeepHopQA. We follow the standard RoG pipeline that performs relation-path planning and then retrieves valid KG paths consistent with the plan before reasoning over the retrieved evidence. For reproducibility, we use the publicly released pretrained checkpoint `rmanluo/RoG`, which is reported as 38 hours of pretraining by the authors.

GPT-5-mini+CoT. We implement an LLM-only chain-of-thought baseline following standard CoT prompting (Wei et al., 2022). Due to limited resources, we use `GPT-5-mini` instead of larger proprietary chat models, while still observing strong performance in Table 1 that exceeds the previously reported ChatGPT+CoT results. We set the reasoning effort to medium. Because LLM outputs can be phrased differently (e.g., "NY" vs. "New York"), we use an external judge `GPT-4o-mini` to score the generated answer list element-wise against gold answers, and we report the judged correctness under the same Hits@1/F1 scores used for other baselines.

GNN-RAG. We run GNN-RAG (Mavromatis & Karypis, 2024) using the same ReaRev retriever outputs for subgraph propagation to ensure identical graph evidence across methods. Although the original GNN-RAG uses pretrained RoG as the LLM reasoner, we replace all LLM-based decision modules with `Qwen-2.5-14B` to ensure consistent LLM capacity and a fair comparison across methods. Since `Qwen-2.5-14B` is not fine-tuned for the original pipeline, direct string matching often fails unless the output format is constrained. To make answer extraction robust, we adopt the stricter answer-only prompt in Table 21, which forces the model to copy entity spans exactly from the provided reasoning paths. We keep the evidence linearization unchanged, following the original path verbalization convention in which entities and relations alternate and are connected by an arrow (`->`) in the context.

GNN-RAG+RA. We evaluate GNN-RAG+RA (Mavromatis & Karypis, 2024) under the same LLM setup as above, using `Qwen-2.5-14B` for final answer generation with the prompt in Table 21. Because GNN-RAG+RA relies on RoG for relation-path planning, we do not run it on DeepHopQA where RoG is not available or trained. For WebQSP and CWQ, we keep the original RoG-based plan generation component to produce relation-path plans, and we append the resulting plans to the evidence context as in the released pipeline. We then performed the final answering step with `Qwen-2.5-14B` to keep the LLM consistent across baselines that use an LLM for final answering.

REKG-MCTS. For REKG-MCTS (Song et al., 2025), the released repository does not provide a complete executable pipeline for our benchmark setting. We therefore re-implement the method by following the paper specification and reported hyperparameters as closely as possible. Because implementation details absent from the release can affect reproduced accuracy, we treat these results

Table 21: LLM prompt used for Qwen-2.5-14B during inference.

```
Based on the reasoning paths, please answer the given question.
You must answer using only the provided reasoning paths.
Return all plausible answers if multiple are required.
Your answers MUST be copied EXACTLY from entity spans that appear
in the paths.

Reasoning Paths:
{Reasoning Paths}

Question:
{question}
```

as a controlled reproduction rather than an exact replication of the originally reported numbers. The method is also memory- and latency-intensive on a single RTX A6000, so we use two GPUs with vLLM serving for `Qwen-2.5-14B` under the original runtime experiments.

RTSoG. For RTSoG (Long et al., 2025), we use the released implementation (`MCTS-KGQAv2`) as the primary reference. Similar to REKG-MCTS, naive generation is not practically runnable on a single RTX A6000 due to memory and latency constraints. Hence, we use `Qwen-2.5-14B` with vLLM on two GPUs while keeping the search logic and prompting consistent.

Hardware-normalized runtime evaluation. In addition to the runtime measurements under the original experimental configurations, we conduct a controlled latency comparison of TriFOR, REKG-MCTS, and RTSoG on a single NVIDIA H200. We use the same randomly sampled 100 questions from each of WebQSP and CWQ for all three methods and report end-to-end wall-clock seconds per question together with the number of LLM calls. All methods use the same `Qwen-2.5-14B` backbone where applicable. The resulting comparison is reported in Table 4.

D.2 TriFOR and TriFOR$^+$ Hyperparameters

TriFOR training. For each corrupted prefix, we sample one random-walk rollout of one to two additional hops, choosing uniformly among the current node's outgoing edges, with probability 0.1 replacing the local choice by a uniformly sampled edge from the question subgraph; gold-path edges are excluded. Both the corrupted prefix and the rollout end state are labeled as value/stop negatives. We summarize the hyperparameters used for offline training of the TriFOR forecaster. We set the hop budget (i.e., the maximum reasoning depth) to $L_{\max} = 2$ for WebQSP, $L_{\max} = 4$ for CWQ, and $L_{\max} = 7$ for DeepHopQA. We optimize the model using the AdamW optimizer and initialize the shared pretrained language model, $\mathrm{LM}_{\mathrm{SR}}$ (Jiang et al., 2022).

TriFOR$^+$ fine-tuning. For TriFOR$^+$, we distill the normalized MCTS visit distribution using cross-entropy, train the value head on the continuous target $V^\star(s) = S(s)/N(s)$ using binary cross-entropy (BCE), and train the stop head on the hit-only label $y^{\mathrm{stop}}_{\mathrm{FT}}(s) = \mathbb{1}[v(s) \in \mathcal{Y}_q]$ using weighted BCE (WBCE). We retain the pairwise ranking loss with weight $\alpha = 0.1$, pairing states from the same question such that $V^\star(s^+) > V^\star(s^-)$. Fine-tuning uses AdamW with batch size 8, learning rate 1×10^{-5}, and 5 epochs, with the remaining optimization settings following offline pretraining.

D.3 PUCT-MCTS and TriFOR$^+$ Hyperparameters

PUCT-MCTS. We run PUCT-MCTS guided by the trained TriFOR forecaster. For each seed tree, we perform N_{sim} MCTS simulations, where action selection follows the PUCT rule with exploration constant c_{puct}. At each state, we retain all outgoing actions when $|\mathcal{A}(s)| \le M_{\mathrm{pool}}$ and otherwise sample M_{pool} actions without replacement. The policy head scores the resulting candidate pool, and the top-M actions are retained for tree expansion. For the stop head, we terminate the search when the predicted stopping probability exceeds ϵ_{stop}, and filter states with low predicted continuation value using a value threshold ϵ_{value}. For fine-tuning data export, we limit the number of exported states per question and require a minimum number of rollouts, using a temperature parameter τ_π to normalize the visit-count policy. Final answer generation uses the same LLM, `Qwen-2.5-14B`.

Table 22: Hyperparameters for offline training of the TriFOR forecaster.

Hyperparameter	**Value**
Max question token length	128
Max hop-text token length	64
Batch size	16
Training epochs	5
Learning rate	1×10^{-5}
Weight decay	1×10^{-2}
Negative actions per positive	24
Policy MLP hidden width	256
Positive-class weight for value BCE	1.0
Positive-class weight for stop BCE	3.0
Pairwise ranking loss weight (α)	0.1
Ranking pairs per positive	4
Hop-level Transformer layers	2
Dropout probability	0.1

Table 23: Hyperparameters for PUCT-MCTS inference (excluding dataset/tokenization limits).

Hyperparameter	**Value**
Simulations per seed tree (N_{sim})	5000
PUCT exploration constant (c_{puct})	1.0
Candidate pool size (M_{pool})	512
Top actions expanded per node (M)	64
Dirichlet concentration (α_{dir})	0.3
Dirichlet mixing weight (ϵ_{dir})	0.25
Stop head threshold (ϵ_{stop})	0.95
Value head threshold (ϵ_{value})	0.05
Export fine-tuning states per question	32
Minimum rollouts for export	4
Policy temperature for export (τ_π)	1.0

E Additional Experimental Results

E.1 Effect of the Pairwise Ranking Loss Weight

We study the impact of the pairwise ranking (PR) term in the offline pretraining objective (Eq. 6) by comparing $\alpha = 0$, which removes $\mathcal{L}_{\text{PR}}$, with $\alpha = 0.1$. The PR loss encourages relative ordering between gold-prefix and corrupted off-path states for the same question, complementing the pointwise BCE supervision on the value head. As shown in Table 24, enabling the PR term consistently improves both Hits@1 and F1 on DeepHopQA.

Table 24: Effect of the PR loss weight α on DeepHopQA.

	Hits@1	**F1**
$\alpha = 0$	66.9	52.7
$\alpha = 0.1$	**67.2**	**54.3**

E.2 Ablation of the Three Forecasting Heads

We evaluate the contribution of the three forecasts by removing one head at a time while retaining the remaining search framework. For the policy-head ablation, we use the same candidate pool of at most 512 actions, score each resulting child state with the value head, retain the top-64 actions, and normalize the child values to form the PUCT prior; the value/stop heads and root Dirichlet noise are otherwise unchanged. For the value-head ablation, we disable value-based pruning and use the stop-head probability in place of the value estimate, so the leaf stop probability is backed up as the MCTS return. For the stop-head ablations, we disable the stop head and terminate either when $V_\theta(s) \geq 0.95$ or at the dataset-specific maximum hop budget $L_{\max} = 2/4/7$ for WebQSP/CWQ/DeepHopQA.

Table 25: Ablation of the three forecasting heads and alternative termination strategies. Each cell reports Hits@1 / F1.

Setting	WebQSP	CWQ
w/o policy head	60.6 / 45.1	43.6 / 39.8
w/o value head	65.6 / 50.7	41.7 / 37.5
w/o stop: value threshold	44.6 / 33.9	33.5 / 29.3
w/o stop: fixed depth	43.0 / 31.8	24.7 / 24.1
TriFOR	**83.0 / 73.7**	**62.1 / 57.2**

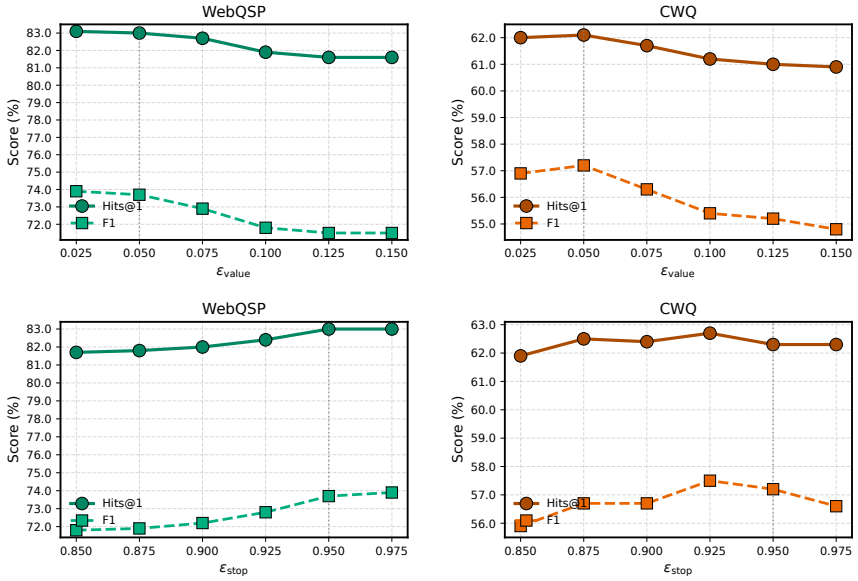

Figure 3: Sensitivity to value/stop thresholds in TriFOR. We vary one threshold at a time while keeping the other at the default (ϵ_{value}=0.05, ϵ_{stop}=0.95).

As shown in Table 25, removing any forecast substantially degrades performance on both datasets, indicating that the three heads provide complementary search guidance. The policy head prioritizes promising outgoing actions, while the value head evaluates whether a partial path remains useful for successful continuation. For termination, both value-threshold stopping and dataset-specific maximum-depth stopping perform substantially worse than the learned stop signal, showing the benefit of instance-adaptive answer-readiness prediction. Together, these results support explicitly separating expansion, continuation evaluation, and termination into policy, value, and stop forecasts.

E.3 Sensitivity to Value/Stop Thresholds

We study the sensitivity to the early-termination thresholds for the value and stop heads. Following our default setting, we use the value threshold $\epsilon_{\text{value}} = 0.05$ for pruning low-value states and the stop threshold $\epsilon_{\text{stop}} = 0.95$ for deciding when a state is ready to terminate the search and yield an answer. We then vary one threshold at a time while keeping the other fixed at its default.

Figure 3 summarizes the results. For the value threshold, increasing ϵ_{value} from 0.025 to 0.150 generally decreases performance on both WebQSP and CWQ, suggesting that overly aggressive value-based pruning can discard states that still lead to correct answers downstream. For the stop threshold, WebQSP performs best in the high-threshold regime, while CWQ slightly benefits from a moderate reduction to $\epsilon_{\text{stop}} = 0.925$. However, lowering the threshold further to 0.850 degrades both datasets, indicating that premature stopping can limit useful downstream exploration. Overall, the default thresholds provide a robust operating point, with moderate variations changing the balance between pruning low-value states and terminating at candidate answers.

E.4 Calibration of the Stop Head

Table 26: Calibration and discrimination of the TriFOR stop head.

Data	ECE ↓	AUROC ↑
WebQSP	17.3	68.9
CWQ	14.0	77.6

We evaluate whether the stop probability used during search provides a meaningful confidence signal for answer readiness. ECE measures calibration against the held-out stop labels, while AUROC measures discrimination between answer-ready and non-terminal states. As shown in Table 26, the stop head provides useful discrimination and calibration for termination decisions. We use this analysis as a diagnostic of the learned stopping signal rather than as a direct calibration comparison with models that predict different events.

E.5 Direct Path-Level Search Quality

End-to-end Hits@1 and F1 can conflate the quality of graph search with the behavior of the final answer LLM. We therefore directly evaluate the retrieved evidence using two complementary metrics. Recall@K measures whether a gold reasoning path is recovered among the top-K retrieved paths. Shortest-support overlap measures the fraction of hops in the top-ranked answer-reaching path that also belong to a shortest answer-supporting path. This is a structural compactness measure and does not by itself assess semantic validity.

Table 27: Path-level search quality (%).

Dataset	Method	Recall@1 / @5 / @20	Shortest-support overlap
WebQSP	**TriFOR**	**41.7 / 56.0 / 60.0**	98.8
	RoG	22.7 / 32.0 / 34.0	**99.3**
CWQ	**TriFOR**	**19.0 / 32.0 / 41.3**	**98.7**
	RoG	3.0 / 5.0 / 6.0	85.7
DeepHopQA	**TriFOR**	17.0 / 34.0 / 43.0	60.2

As shown in Table 27, TriFOR substantially improves gold-path recall over RoG on WebQSP and CWQ, reaching 60.0% vs. 34.0% and 41.3% vs. 6.0% at R@20, respectively, while maintaining high shortest-support overlap (98.8%/98.7%). On DeepHopQA, recall increases from 17.0% at R@1 to 43.0% at R@20, showing that useful long-hop evidence is often recovered beyond the top-ranked path. Gold-path recall is stricter than answer coverage, since a path may reach the correct answer without matching the annotated gold reasoning path. Thus, downstream QA can still succeed when the gold path is not recovered. RoG is omitted on DeepHopQA because it requires dataset-specific pretraining unavailable for this new benchmark. Overall, these results confirm that TriFOR directly improves search quality rather than relying on the final answer generator.

E.6 Gold-Action Retention under Policy Pruning

PUCT explores only the actions retained after policy-based top-M pruning. We measure the conditional recall of the gold next action after it enters the 512-action candidate pool.

As shown in Table 28, policy pruning preserves most gold actions once they enter the candidate pool, with the default top-64 setting retaining 96.61%–99.02% across datasets. Recall largely saturates by $M = 64$, supporting the default setting as a balance between preserving promising actions and controlling the branching factor. DeepHopQA also maintains high top-64 recall across annotated hop depths, suggesting that deeper paths do not systematically degrade policy-based action retention.

Table 28: Gold-action recall after policy-based top-M pruning conditioned on the gold action being present in the candidate pool.

Dataset	Top-8	Top-32	Top-64	Top-128
WebQSP	81.04	93.01	96.61	98.91
CWQ	85.33	94.76	97.64	99.29
DeepHopQA	84.38	96.06	99.02	99.92

E.7 Sensitivity to the PUCT Exploration Constant

We analyze the sensitivity of TriFOR to the PUCT exploration constant c_{PUCT} used in the selection rule (Section 3.2.1). Recall that c_{PUCT} scales the prior-guided exploration term, controlling how strongly the search prioritizes actions with high policy probability $\pi_\theta(a \mid s_t)$ relative to the empirical action-value estimate $Q(s_t, a)$, which corresponds to exploitation based on accumulated rewards. Smaller c_{PUCT} makes selection more exploitative, whereas larger values encourage broader exploration guided by the learned policy.

Figure 4 plots Hits@1 on WebQSP and CWQ when varying $c_{\text{PUCT}} \in \{0.6, 0.8, 1.0, 1.2, 1.4\}$. Performance remains stable across a moderate range, with WebQSP peaking at $c_{\text{PUCT}} = 1.0$ and CWQ slightly favoring 1.2–1.4. Overall, the default choice $c_{\text{PUCT}} = 1.0$ provides a strong operating point and performs well without extensive tuning.

E.8 Effect of the Simulation Budget

Table 29: Effect of the per-seed simulation budget on accuracy and inference time.

Data	Simulations	H@1	F1	Inference time
WebQSP	1000	81.3	72.3	1h 42m
	3000	82.5	73.6	1h 44m
	5000	83.0	73.7	1h 52m
	7000	83.1	73.9	1h 56m
	9000	83.4	74.2	2h 1m
CWQ	1000	61.7	56.2	3h 59m
	3000	61.8	56.6	4h 5m
	5000	62.1	57.2	4h 15m
	7000	62.3	57.7	4h 27m
	9000	62.3	57.8	4h 40m
DeepHopQA	1000	57.9	51.2	6h 19m
	3000	65.3	53.6	8h 14m
	5000	67.2	54.3	9h 36m
	7000	67.4	54.5	10h 27m
	9000	66.4	54.3	11h 17m

We analyze the accuracy–latency trade-off induced by the per-seed MCTS simulation budget while keeping the remaining inference pipeline fixed. As shown in Table 29, performance generally improves with additional search, with the largest gains on DeepHopQA. Increasing the budget from 1000 to 5000 simulations raises DeepHopQA H@1 from 57.9 to 67.2, whereas WebQSP and CWQ largely saturate by 5000 simulations. These results support our default budget of 5000 as a practical operating point that captures most of the accuracy gains without the higher latency of larger budgets.

Importantly, the simulation budget does not correspond one-to-one with neural forecaster evaluations. TriFOR caches the policy, value, and stop predictions of previously evaluated tree states, so revisiting a state during subsequent simulations reuses its stored forecasts rather than invoking another forward pass. Consequently, the default 5000 simulations require only approximately 74–821 unique forecaster evaluations per question, depending on the dataset, substantially fewer than the nominal number of simulations. Thus, the computational cost of PUCT is more accurately reflected by unique state evaluations and measured runtime than by the simulation count alone.

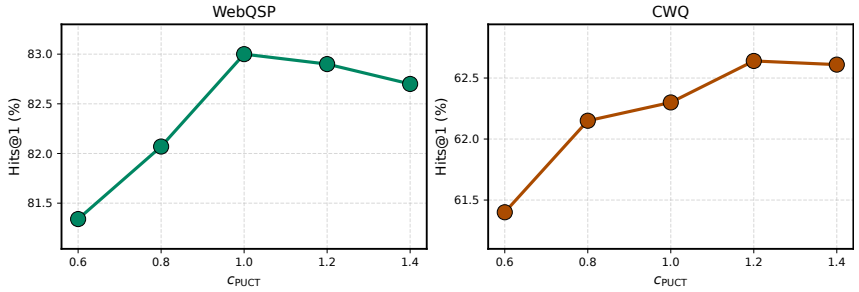

Figure 4: Sensitivity to the exploration constant c_{PUCT} in the PUCT selection.

E.9 Search Controller Comparison

To isolate the effect of the search controller, we compare greedy, beam, best-first, uniform-prior PUCT, and TriFOR's learned-prior PUCT under a shared diagnostic protocol. All controllers use the same fixed forecaster, candidate-pool and top-M settings, top-20 evidence interface, and `Qwen-2.5-14B` final-answer model. We evaluate the controllers on the same fixed 300-question diagnostic subsets of WebQSP and CWQ. Because this experiment is designed to compare controllers under a common diagnostic setting, its absolute scores are not intended to reproduce the full-test results in Table 1.

Controller settings. Greedy follows the highest-prior action at each step, beam search keeps multiple partial paths ranked by cumulative policy score, and best-first search repeatedly expands the frontier state with the highest value. PUCT balances learned priors with empirical action values, while the uniform-prior variant uses a uniform PUCT prior while retaining policy-based filtering.

Table 30: End-to-end search-controller comparison on fixed 300-question diagnostic subsets. Each cell reports Hits@1 / F1 using the top-20 retrieved evidence paths.

Controller	WebQSP	CWQ
Greedy	62.7 / 39.0	31.3 / 24.1
Beam	73.3 / 61.5	39.3 / 34.6
Best-first	70.3 / 57.6	42.0 / 39.8
Uniform-prior PUCT	**75.7** / 61.7	32.0 / 28.8
PUCT	75.0 / **61.7**	**44.0 / 39.9**

As shown in Table 30, multi-path search substantially improves over greedy decoding under this controlled setting. PUCT achieves the strongest performance on CWQ, while uniform-prior PUCT obtains a slightly higher Hits@1 on WebQSP. Beam and best-first search are also competitive, indicating that the gains cannot be attributed simply to replacing greedy decoding with a particular tree-search controller. These results motivate separating the contribution of explicit multi-branch search from that of TriFOR's learned policy prior, which we analyze next.

Effect of the learned policy prior. To isolate the contribution of the learned policy prior within PUCT, we replace $\pi_\theta(a \mid s)$ with a uniform prior while matching the number of unique evaluated states. Across three seeds, the learned prior improves retrieval ranking on both datasets, while its effect on evidence coverage differs across regimes. On WebQSP, Hit@20 changes only marginally despite a large Hit@1 gain, whereas on CWQ both coverage and final QA improve substantially. This suggests that the learned prior consistently improves search ranking, while end-to-end gains arise when this improvement also changes the evidence available to the final answer model.

E.10 Answer Coverage vs. the Number of Provided Paths

We analyze how often the gold answer is already contained in the retrieved evidence before LLM reasoning. The reported Hits@k does not measure the end-to-end QA accuracy after LLM inference, but instead measures *answer coverage* of the retrieved paths, i.e., whether the gold answer appears in the top-k paths provided to the LLM. This diagnostic isolates the effect of evidence availability from downstream generation quality.

Table 31: Effect of the learned policy prior, reported as mean differences in percentage points between learned-prior and uniform-prior PUCT over three seeds. Positive values favor the learned prior.

Dataset	Retrieval Δ H@1	Retrieval Δ H@20	End-to-end Δ H@1
WebQSP	+15.00	+0.55	−0.22
CWQ	+5.67	+9.11	+10.67

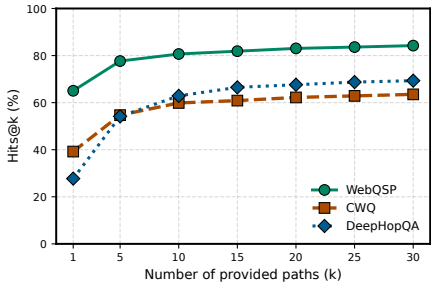

Figure 5: Answer coverage as a function of the number of provided paths k.

Figure 5 shows that the answer coverage increases rapidly from k=1 to k=10 and then saturates, indicating the diminishing returns from providing additional paths beyond moderate k. Across datasets, WebQSP exhibits higher coverage than CWQ and DeepHopQA, consistent with their greater complexity and longer-hop requirements. In our main system, we provide the top-20 paths to the LLM, which lies in the saturated regime and balances coverage with prompt length, since supplying too many paths can overload the context and potentially reduce generated answer quality or accuracy.

F Generalization to Incomplete Knowledge Graphs

F.1 BRINK-Wikidata5m Setup

The main experiments evaluate TriFOR on Freebase-based benchmarks where the relevant graph evidence is generally available in the provided subgraph. To test whether forecast-guided search remains effective when direct evidence is missing, we additionally evaluate on BRINK-Wikidata5m (Zhou et al., 2026), a benchmark designed to assess reasoning under incomplete knowledge graphs. BRINK removes the direct supporting triple for a designated hard answer, requiring models to recover the answer through alternative multi-hop evidence remaining in the graph. The benchmark uses Wikidata5m, containing approximately 20.5M triples, and we evaluate on all 3,464 test questions.

We follow the BRINK evaluation protocol and report four metrics. Hits@Any measures whether the predicted answer set contains any gold answer, while Hits@Hard measures recovery of the designated hard answer whose direct supporting evidence has been removed. F1 evaluates precision–recall over the predicted answer set. HHR (Hard Hit Ratio) is defined as Hits@Hard / Hits@Any and measures the fraction of successful questions for which the missing-evidence hard answer is recovered.

Implementation. TriFOR is trained from scratch on the training split using rule-grounding proof paths for policy, value, and stop supervision. We textualize entities and relations using the released primary labels, and search the incomplete KG bidirectionally with $L_{\max} = 4$, $N_{\text{sim}} = 1{,}280$, $M_{\text{pool}} = 128$, and $M = 32$. No LLM reader is used; answers are read directly from the search tree.

F.2 Incomplete-KG Generalization Results

As shown in Table 32, TriFOR achieves the strongest Hits@Any, Hits@Hard, and F1 among the compared methods. Its 50.8 Hits@Hard shows that forecast-guided search can recover answers through alternative evidence paths after direct supporting triples are removed, while its HHR of 82.5 remains near the best. These results extend the main evaluation to incomplete Wikidata5m and show robustness when the most direct supporting evidence is unavailable. They also support the use of corrupted off-path supervision. Although an off-path corruption may leave an alternative route to the answer, TriFOR remains effective at recovering question-relevant evidence.

Table 32: Results on BRINK-Wikidata5m under incomplete graph evidence. The direct supporting triple for the designated hard answer is removed, requiring recovery through alternative paths.

Method	Hits@Any	Hits@Hard	F1	HHR
RoG	57.0	34.8	37.0	61.1
GPT-5-mini+CoT	9.5	7.9	3.5	**83.0**
GNN-RAG	59.0	37.3	39.0	63.2
REKG-MCTS	42.3	19.2	21.4	45.4
TriFOR	**61.6**	**50.8**	**48.4**	82.5

Comparison with LLM-guided MCTS. BRINK also provides a challenging setting for testing whether removing repeated in-loop LLM calls sacrifices reasoning ability. Compared with REKG-MCTS, TriFOR improves Hits@Any from 42.3 to 61.6, Hits@Hard from 19.2 to 50.8, and F1 from 21.4 to 48.4. Thus, TriFOR does not exhibit an accuracy–efficiency trade-off in this incomplete-KG setting: its learned search guidance recovers substantially more alternative evidence while avoiding repeated LLM evaluation inside the search loop.

G ALGORITHMS

We summarize two algorithmic procedures used in our TriFOR pipeline. Algorithm 1 details the offline data construction used for pretraining, producing tuples that supervise policy for action selection and state-level value/stop prediction from gold paths with additional negatives. Algorithm 2 describes forecast-guided PUCT-MCTS inference, including expansion control, termination conditions, backpropagation updates, followed by ranking and selecting the top-K evidence paths.

Algorithm 1 Offline Synthetic Dataset Construction for Pretraining

Input: question–gold path pairs $\{(q, p^\star)\}$, policy negatives K, value negatives N_{val}.
Output: offline supervision set $\mathcal{D}_{\text{off}}$ with tuples $(s, a^+, \{a_k^-\}, y^{\text{value}}, y^{\text{stop}})$.
Initialize $\mathcal{D}_{\text{off}} = \emptyset$.
for each $(q, p^\star = (v_0, r_1^\star, v_1^\star, \ldots, r_L^\star, v_L))$ **do**
  // Positive samples and Policy negatives
  for $t = 0$ **to** L **do**
    Define $s_t^+ = (q, p_{1:t}^\star)$.
    Set $y^{\text{value}} = 1$, $y^{\text{stop}} = \mathbb{1}[t = L]$.
    if $t < L$ **then**
      Set $a_t^+ = (r_{t+1}^\star, v_{t+1}^\star)$
      Sample policy negatives $\{a_{t,k}^-\}_{k=1}^K \sim \mathcal{A}(s_t^+) \setminus \{a_t^+\}$.
    else
      Set $a_t^+ = \varnothing, \{a_{t,k}^-\}_{k=1}^K = \varnothing$
    end if
    Add $(s_t^+, a_t^+, \{a_{t,k}^-\}, y^{\text{value}}, y^{\text{stop}})$ to $\mathcal{D}_{\text{off}}$.
  end for
  // Value negatives and Stop negatives
  for $j = 1$ **to** N_{val} **do**
    Select $t \in \{0, \ldots, L-1\}$; let $s_t^+ = (q, p_{1:t}^\star)$ and $a_t^+ = (r_{t+1}^\star, v_{t+1}^\star)$.
    Sample a corrupt action $a_t^- \sim \mathcal{A}(s_t^+) \setminus \{a_t^+\}$ and form $s_{t+1}^- = (q, p_{1:t}^\star \oplus a_t^-)$.
    Add $(s_{t+1}^-, \varnothing, \varnothing, 0, 0)$ to $\mathcal{D}_{\text{off}}$.
    Sample one short random-walk rollout from the leaf of s_{t+1}^- to obtain an additional negative state s^-.
    Add $(s^-, \varnothing, \varnothing, 0, 0)$ to $\mathcal{D}_{\text{off}}$.
  end for
end for

Algorithm 2 Forecast-Guided PUCT-MCTS Inference and Evidence Extraction

Input: question q, subgraph $\mathcal{G}_q$, seed entities S_q, tri-head $(\pi_\theta, V_\theta, T_\theta)$, hop budget $L_{\max}$, c_{PUCT}, top-M expansion, stop threshold ϵ_{stop}, value threshold ϵ_{value}, simulations N_{sim}, output top-K.
Output: ranked evidence reasoning paths $\mathcal{P}_K$.

Initialize $\mathcal{P}_{\text{cand}} \leftarrow \emptyset$.
for each seed entity v_0 in S_q **do**
  Initialize root state $s_0 \leftarrow (q, p_0)$ with $p_0 = (v_0)$.
  Score root: $(\pi_\theta(\cdot \mid s_0), V_\theta(s_0), T_\theta(s_0))$.
  Retain the top-M candidate actions ranked by $\pi_\theta(\cdot \mid s_0)$.
  Renormalize the retained priors, apply Dirichlet noise, and expand the retained actions.
  for $i = 1$ **to** N_{sim} **do**
    $s \leftarrow s_0$; initialize trajectory $\tau \leftarrow \langle\rangle$.
    // Selection (PUCT)
    while s is expanded **and not** ISTERMINAL(s) **do**
      Select action $a^\star \leftarrow \arg\max_{a \in \mathcal{A}(s)} \left(Q(s,a) + c_{\text{PUCT}} \pi_\theta(a \mid s) \frac{\sqrt{\max\{1, N(s)\}}}{1 + n(s,a)} \right)$.
      Append $(s, a^\star)$ to τ.
      For $s = (q, p_{1:t})$, transition to $s \leftarrow (q, p_{1:t} \oplus a^\star)$.
      if $V_\theta(s)$ and $T_\theta(s)$ are not cached **then**
        Evaluate and cache $V_\theta(s)$ and $T_\theta(s)$.
      end if
    end while
    // Expansion
    if not ISTERMINAL(s) **then**
      Score candidate actions with $\pi_\theta(\cdot \mid s)$ and expand s with top-M actions in $\mathcal{A}(s)$.
    end if
    // Backpropagation
    Set leaf reward $R \leftarrow V_\theta(s)$.
    for each (s_t, a_t) along τ in reverse order **do**
      $n(s_t, a_t) \leftarrow n(s_t, a_t) + 1$; $N(s_t) \leftarrow N(s_t) + 1$.
      $W(s_t, a_t) \leftarrow W(s_t, a_t) + R$; $Q(s_t, a_t) \leftarrow W(s_t, a_t)/n(s_t, a_t)$.
    end for
  end for
  // Evidence extraction
  Enumerate candidate evidence paths $\mathcal{P}$ from the search tree.
  For path $p \in \mathcal{P}$, compute $n_{\min}(p) = \min_{(s,a) \in p} n(s,a)$ and $\overline{Q}_{\text{path}}(p) = \frac{1}{|p|} \sum_{(s,a) \in p} Q(s,a)$.
  Add all paths in $\mathcal{P}$ to $\mathcal{P}_{\text{cand}}$.
end for
Rank $\mathcal{P}_{\text{cand}}$ by $(n_{\min}(p), \overline{Q}_{\text{path}}(p), V_\theta(s_{\text{leaf}}))$.
Set $\mathcal{P}_K$ to the top-K ranked paths in $\mathcal{P}_{\text{cand}}$.

function ISTERMINAL(s)
  Let $s = (q, p_{1:t})$.
  if $t \geq L_{\max}$ **or** $\mathcal{A}(s) = \emptyset$ **or** $T_\theta(s) \geq \epsilon_{\text{stop}}$ **or** $V_\theta(s) \leq \epsilon_{\text{value}}$ **then**
    return True
  else
    return False
  end if
end function