DEEPREFINE: AGENTIC KNOWLEDGE REFINEMENT VIA REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

External knowledge enables large language model (LLM) agents to ground their actions and decisions beyond intrinsic parametric memory in open-ended, knowledge-intensive downstream tasks. Yet the quality of the underlying knowledge bases is systematically limited by incompleteness, incorrectness, or redundancy, manifested as missing evidence or cross-document links, low-confidence or imprecise claims, and ambiguous or coreference resolution issues. Such defects compound under iterative use, degrading retrieval fidelity and downstream task performance. We present **DeepRefine**, a reinforcement learning framework for agentic knowledge refinement that evolves the quality of pre-constructed structured knowledge bases, e.g., knowledge graphs or LLM-Wikis, with user queries to make it more suitable for the downstream tasks. DeepRefine performs multi-turn interactions with the knowledge base and conducts abductive diagnosis over the interaction history, localizes likely defects, and executes targeted refinement actions for incremental knowledge base updates. To further optimize refinement policies of DeepRefine without golden refinement trajectories, we introduce a Gain-Beyond-Draft (GBD) reward and train the reasoning process end-to-end via reinforcement learning. Extensive experiments demonstrate consistent downstream gains over strong baselines.

1 INTRODUCTION

Large Language Models (LLMs) have shown remarkable capabilities in handling natural language understanding and generation tasks (Yang et al., 2025; Jaech et al., 2024). And retrieval-augmented generation (RAG) grounds LLMs in external knowledge bases (KBs) outside model parameters (Lewis et al., 2020; Gao et al., 2023). A dominant recipe indexes external knowledge as vectors for dense retrieval. Graph-based RAG systems additionally organize salient content into graph-structured KBs to support multi-hop retrieval and reasoning (Edge et al., 2024; Guo et al., 2024; Gutiérrez et al., 2025; Huang et al., 2025; Yang et al., 2026). In both cases, the external KB is usually built once and static at serving time, which provides LLMs with richer external knowledge and enhances their capabilities in domain-specific or knowledge-intensive tasks.

Yet a static KB is largely stateless with respect to ongoing use. It does not accumulate structured lessons from repeated queries, user-specific context, or iterative agent workflows. Recently some agentic KBs (e.g., LLM-Wiki (Karpathy, 2026)) are proposed to address this gap by having LLM-based agents ingest sources, synthesize articles, and maintain cross-references over time. For example, toolkits that compile documents using agentic workflows and file outputs back into a user-owned persistent store. As shown in the left part of Figure 1, such systems make external KBs evolvable rather than frozen at test time.

Despite this progress, both of the static and agentic KBs often exhibit systematic quality defects, including incompleteness, incorrectness, and redundancy (Subagdja et al., 2024; Cimiano & Paulheim, 2017). In practice, these defects manifest as missing evidence chains or cross-document links, low-confidence or imprecise assertions, and ambiguous or coreference resolution issues. Such issues compound and degrade both retrieval fidelity and end-task performance. Recent studies mainly optimize the construction policy of KB builders (Wang et al., 2025; Tsang et al., 2025), however, in many real deployments the KB has already been constructed, so re-training the constructor and

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.

re-compiling the whole KB is often expensive and operationally impractical at scale (Choubey et al., 2024; Wolff & Bennati, 2026), motivating a post-construction knowledge refinement paradigm.

Achieving effective knowledge refinement presents two core challenges: **(1) defect localization**, i.e., identifying problematic regions in a large KB without performing expensive traversal checks; and **(2) policy optimization without golden refinement trajectories**, i.e., even though ground truth answers are usually available in various benchmarks, it is still challenging for a model to learn effective refinement strategies with the absence of golden refinement trajectories or golden KBs.

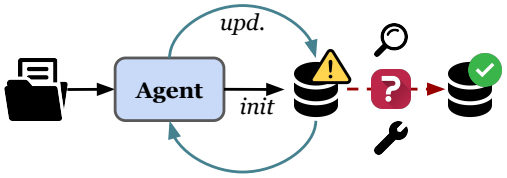

Figure 1: The quality problems of agentic KBs (left) and the challenges of KB refinement (right).

As shown in the right part of Figure 1, Challenge (1) arises because KBs are typically large and relationally connected, making exhaustive inspection infeasible for locating defects that stem from incompleteness, incorrectness, or redundancy (Dong et al., 2025). Challenge (2) arises because the LLM-based agents have difficulty refining KBs and learning how to refine KBs. The real-world refinement lacks golden trajectories and we generally do not observe the "correct" sequence of refinement actions that best aligns a KB with downstream tasks. Consequently, learning effective policies requires training objectives that can evaluate refinement quality through downstream task utility signals.

To address these challenges, we propose **DeepRefine**, a reinforcement learning (RL) framework for agentic knowledge refinement that improves pre-constructed KBs, e.g., LLM-Wiki and knowledge graphs (KGs), with user queries to make it more suitable for the downstream tasks. DeepRefine follows a three-step reasoning process: **Answerability Judgement Loop**, **Error Abduction**, and **Refinement Actions Generation**. For Challenge (1), instead of traversing the full KB, DeepRefine performs multi-turn query-conditioned interaction with external knowledge to localize potentially defective neighborhoods. This agentic design substantially reduces refinement complexity while preserving effectiveness. For Challenge (2), DeepRefine abductively infers likely defect causes from interaction histories, then generates refinement actions to update the KB incrementally. Because the utility of refinement actions is only observable through downstream task performance and is non-differentiable, we introduce a **Gain-Beyond-Draft (GBD) reward** and optimize the refinement policy end-to-end via reinforcement learning. Importantly, DeepRefine uses only the user query and multi-turn retrieved evidences during refinement, while the gold answer is used only for training-time reward computation.

In summary, we make the following main contributions:

- To the best of our knowledge, **DeepRefine** is the first RL framework for agentic knowledge refinement, capable of evolving the quality of any pre-constructed structured knowledge base using only user queries to make it more suitable for the downstream tasks.
- We design a three-step reasoning process including **Answerability Judgement Loop**, **Error Abduction**, and **Refinement Actions Generation**. We further introduce a **Gain-Beyond-Draft (GBD) reward** with reinforcement learning to enhance the knowledge base refinement capabilities of LLMs.
- We conduct extensive experiments across five datasets to demonstrate that DeepRefine can steadily improve the performance of state-of-the-art baseline methods.

2 Related Work

2.1 Agentic Knowledge Base Management

KBs are effective for agentic use because they usually capture relational and multi-hop dependencies, organize evidence hierarchically (Huang et al., 2025; Zhang et al., 2025; Xie et al.), and support efficient retrieval (Yang et al., 2026). Different from graph-enhanced RAG methods that largely treat constructed structures as static artifacts (Edge et al., 2024; Jimenez Gutierrez et al., 2024), recent agent memory systems (e.g., Zep (Rasmussen et al., 2025), Mem0 (Chhikara et al., 2025)) emphasize

continuous maintenance and evolution for new knowledge. Existing RL-based approaches, such as Mem-α (Wang et al., 2025), Memory-R1 (Yan et al., 2025), and AutoGraph-R1 (Tsang et al., 2025), primarily optimize construction-time KB management policies. In contrast, our work focuses on post-construction refinement of already deployed KBs.

2.2 LLMs and Reinforcement Learning

Reinforcement learning (RL) (Kaelbling et al., 1996) is an effective optimization framework for improving the sequential decision-making capabilities of LLMs through learning from environmental interaction and reward feedback (Xi et al., 2025). Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022) is one of the foundational methods used to align LLM outputs with preferences. And Proximal Policy Optimization (PPO) (Schulman et al., 2017), and Group Relative Policy Optimization (GRPO) (Shao et al., 2024) have provided scalable and efficient optimization algorithms for various structured decision-making tasks in several domains. For example, Graph-R1 (Luo et al., 2025) employs RL to learn effective graph-structured tools navigation to retrieve more accurately. Search-R1 (Jin et al., 2025) trains LLMs to optimize web search queries using RL to maximize final answer correctness and `s3` (Jiang et al., 2025) further proposes a principled, model-agnostic reward signal that quantifies improvements over standard retrieval.

2.3 Knowledge Refinement

Knowledge refinement aims to mitigate incompleteness, incorrectness, and redundancy in existing artifacts (Subagdja et al., 2024; Cimiano & Paulheim, 2017). Classical KG completion is one of the knowledge refinement methods, such as KG-BERT (Yao et al., 2019) and KGRefiner (Saeedizade et al., 2022), are typically limited to adding links or relations. Entity alignment and disambiguation methods, such as NeuSymEA (Chen et al., 2024) and K-NED (Feng et al., 2020), focus on correction and simplification. In contrast, knowledge refinement actually has a broader scope that subsumes both types of operations (Cimiano & Paulheim, 2017), yet remains underexplored. TRAIL (Zhao et al., 2025) enables LLM-based agents to iteratively explore and update KGs during reasoning, however, its edits are not explicitly optimized against downstream task utility. Our method instead learns refinement policies with an explicit downstream reward, yielding tighter alignment between KB updates and end-task performance.

3 Preliminaries

In this section, we formalize the KB and the refinement task with respect to RAG-based downstream tasks. We use constructed KGs and LLM-Wikis as the KBs. They are represented as structured triples where the heads and tails may be entities, events, or document-grounded spans, and relations may capture taxonomy, time, causality, or cross-document linkage.

We denote the pre-constructed KB by a set of triples $\mathcal{G}_\text{f} = \{(h, r, t) \mid h, t \in \mathcal{E}_\text{f},\ r \in \mathcal{R}_\text{f}\}$, where $\mathcal{E}_\text{f}$ is the set of entities and $\mathcal{R}_\text{f}$ is the set of relations. The KBs (KGs or LLM-Wikis) may be produced by any LLM-based constructors. We are also given a set of user queries $\mathcal{Q} = \{q_1, q_2, \ldots, q_N\}$ related to the $\mathcal{G}_\text{f}$. The goal of agentic knowledge refinement is to update $\mathcal{G}_\text{f}$ with user queries in $\mathcal{Q}$ to make it more suitable for the downstream RAG tasks. In real-world applications, these queries can be the users' query history and our method aims to make use of them to evolve the quality of the pre-constructed KB.

Then we further formulate KB refinement as generating sequences of actions that mitigate incompleteness, incorrectness, and redundancy in $\mathcal{G}_\text{f}$. This process can be represented as:

$$\arg\max_{S_{\mathcal{A}_\text{q}} \in S} p_\theta(S \mid \mathcal{G}_\text{f}, \mathcal{Q}) = \{\mathcal{A}_\text{q}^1, \ldots, \mathcal{A}_\text{q}^{|\mathcal{Q}|}\}, \tag{1}$$

where $S_{\mathcal{A}_\text{q}}$ is the set of action sequence $\mathcal{A}_\text{q}^i = \{a_1, a_2, \ldots, a_{M_i}\}$ with respect to the i-th query $q \in \mathcal{Q}$, S is the possible output set, and p_θ is the policy model parameterized by θ. These actions are then applied to improve the quality of the KB $\mathcal{G}_\text{f}$.

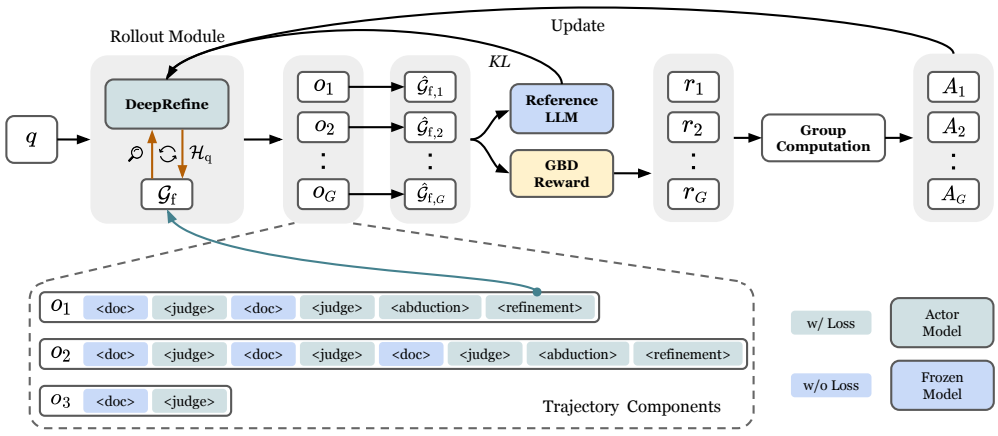

Figure 2: Demonstration of GRPO training for the agentic knowledge refinement (DeepRefine), where the orange arrows denote the multi-turn interactions of DeepRefine with KBs and the green arrows denote the refinement actions on KBs.

4 METHODOLOGY

4.1 REASONING STEPS DESIGN

In this section, we will demonstrate the reasoning steps of DeepRefine for the agentic knowledge refinement. Note that it is a multi-turn reasoning process, which means the preceding reasoning steps will be served as the context for the subsequent reasoning steps.

Answerable Judgement Loop. Initially, DeepRefine will judge the answerability of the given query with several interactions with the KB $\mathcal{G}_\text{f}$, as illustrated in the loop of the rollout module in Figure 2. If the query is answerable in the first interaction, DeepRefine will directly skip that. Otherwise, DeepRefine will refine $\mathcal{G}_\text{f}$ in later reasoning steps to make it more easily to be answered. In the first interaction with $\mathcal{G}_\text{f}$, as shown in Eq. 2, DeepRefine will conduct a simple dense retrieval to obtain the top-N related triples to the query with their embeddings, which can be considered as a 0-hop retrieved subgraph $\mathcal{G}_\text{q}^{(0)}$.

$$\mathcal{G}_\text{q}^{(0)} = \text{Top-}k(q, \mathcal{G}_\text{f}, N). \tag{2}$$

Then for each of the following interactions, DeepRefine will continue to expand the retrieved subgraph if the answerability is not satisfied in the previous interaction. And only the newly retrieved triples will be added as user messages in the next turn's assistant messages generation (judgement). To be specific, the subgraph expanding procedure is defined as follows:

$$\mathcal{G}_\text{cand}^{(i)} = \{(h, r, t) \in \mathcal{G}_\text{f} \mid h \text{ or } t \in \mathcal{E}_\text{q}^{(i-1)}\}, \tag{3}$$

$$\mathcal{G}_\text{pruned}^{(i)} = \text{Top-}k(q, \mathcal{G}_\text{cand}^{(i)}, M), \tag{4}$$

$$\mathcal{G}_\text{q}^{(i)} = \mathcal{G}_\text{q}^{(i-1)} \cup \mathcal{G}_\text{pruned}^{(i)}, \tag{5}$$

where $\mathcal{G}_\text{cand}^{(i)}$ is a set of new candidate triples that connect to the knowledge items in the subgraph $\mathcal{G}_\text{q}^{(i-1)}$ and $\mathcal{E}_\text{q}^{(i-1)}$ is the knowledge item set of the subgraph $\mathcal{G}_\text{q}^{(i-1)}$. To control the size of the retrieved subgraph with respect to the query q, we further prune the candidate triples with their embeddings, and select the top-M related triples to the query, as shown in Eq. 4. Then we merge these new triples with the previous subgraph and obtain the i-hop retrieved subgraph $\mathcal{G}_\text{q}^{(i)}$. The process of interactions will continue until the answerability is satisfied or the maximum number of expansion hops is reached. The judgement output will be enclosed within `<judge>` and `</judge>` tags as the prompt template demonstrated in Figure 10 of Appendix I. After it terminates, the query-specific interaction history $\mathcal{H}_\text{q}$ with a total length of L can be finally obtained for the following reasoning steps. The interaction history consists of the retrieved subgraph at each interaction step and the query:

$$\mathcal{H}_\text{q} = \bigcup_{i=0}^{L-1} \{(q, \mathcal{G}_\text{q}^{(i)}, \mathcal{J}_\text{q}^{(i)})\}, \tag{6}$$

where $\mathcal{J}_{\text{q}}^{(i)}$ is the answerable judgement result of the query q at the i-th interaction step.

Error Abduction. We assume that there are some potential issues in the triples of a query-specific subgraph if the query is not answerable with the 0-hop retrieved subgraph. In this reasoning step, given the interaction history $\mathcal{H}_{\text{q}}$, DeepRefine will abductively reason the potential issues $\mathcal{I}_{\text{q}}$ in the retrieved subgraphs from three perspectives: incompleteness, errors and redundancy, which are based on the definition of KG refinement (Subagdja et al., 2024) proposed by Cimiano & Paulheim (2017). The error reasons will be generated within the `<abduction>` and `</abduction>` tags as the prompt template demonstrated in Figure 11 of Appendix I. For example, if the query is about a multi-hop relationship between two entities, but the retrieved subgraphs wrongly include or even do not contain some intermediate entities or relations, DeepRefine will reason the incorrect or missing parts that may lead to the unanswerable result. Similarly, as for redundancy, DeepRefine will also check if there is any ambiguous knowledge that would affect the answerability. The analysed results in this step will be served as the reference for the following refinement actions generation.

Refinement Actions Generation. Based on the analysed potential issues $\mathcal{I}_{\text{q}}$ in the subgraphs retrieved from the previous interactions, DeepRefine will edit the full KB $\mathcal{G}_{\text{f}}$ accordingly to improve its quality. To reduce token costs and improve the refinement efficiency, instead of reconstructing the subgraphs and then inserting into the full KB, for each query q, DeepRefine will generate a series of refinement actions $\mathcal{A}_{\text{q}}$ within the `<refinement>` and `</refinement>` tags to edit the full KB directly:

$$\mathcal{A}_{\text{q}} = \arg\max_{\mathcal{A}_{\text{q}} \in \mathcal{A}} p_\theta(\mathcal{A} \mid q, \mathcal{G}_{\text{q}}^{(L)}, \mathcal{I}_{\text{q}}), \tag{7}$$

where $\mathcal{A}$ is the set of possible output from DeepRefine parameterized by θ. The generated refinement actions will be parsed into three pre-defined types of operators: `insert_edge()`, `delete_edge()`, and `replace_node()`. The action space is designed to refine the KB along the three issues discussed above. The pre-defined operators are just interfaces, which makes this paradigm can be generalizably applied across various graph databases. We also further elaborate on the rationality of the action space design from both functionality and flexibility perspectives in Appendix G. After applying the actions, we will obtain the refined KB $\hat{\mathcal{G}}_{\text{f}}$. And the full rollout process is also elaborated in Algorithm 1.

4.2 Refinement Policy Optimization

Reward Design. To optimize the knowledge refinement policy π_θ, we consider knowledge refinement as an end-to-end RL problem. Specifically, as shown in the Figure 2, inspired by the prior work (Jiang et al., 2025), we introduce Gain Beyond Draft (GBD) as the reward objective, which quantifies the gain in RAG generation accuracy achieved by the refined KB $\hat{\mathcal{G}}_{\text{f}}$ relative to the draft KB $\mathcal{G}_{\text{f}}$:

$$\text{GBD}(q) = \text{F1}(A_{\text{refined}}, A) - \text{F1}(A_{\text{draft}}, A), \tag{8}$$

where A_{refined} and A_{draft} are the generated answers with the refined KB $\hat{\mathcal{G}}_{\text{f}}$ and the draft one $\mathcal{G}_{\text{f}}$, which are both obtained through the subgraph retriever we mentioned in Section 5.1. A is the golden answer for the query and $\text{F1}(\cdot,\cdot)$ is the token-level F1 score. It is only accessed by the training system for reward computing and is never provided during the rollout.

Policy Optimization. To optimize the KB refinement policy π_θ of DeepRefine, we adopt the Group-Relative Policy Optimization (GRPO) (Shao et al., 2024) algorithm to finetune the policy with the GBD reward. Formally, the objective is defined as:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{\mathbf{q}\sim P_Q, \{\mathbf{o}_i\}\sim\pi_{\theta_{\text{old}}}(\cdot|\mathbf{q})} \left[\frac{1}{G}\sum_{i=1}^{G} \frac{1}{|\mathbf{o}_i|} \sum_{t=1}^{|\mathbf{o}_i|} \min\left(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\Big(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon\Big)\hat{A}_{i,t} \right) - \beta \mathbb{D}_{\text{KL}}[\pi_\theta || \pi_{\text{ref}}] \right], \tag{9}$$

where $r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t}|\mathbf{q},\mathbf{o}_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|\mathbf{q},\mathbf{o}_{i,<t})}$ denotes the probability ratio and $\hat{A}_{i,t} = \frac{R_i - \mu_R}{\sigma_R}$ represents the group-relative advantage for the entire refinement reasoning process. And π_{ref} is the reference policy. During the training process, DeepRefine will generate the knowledge refinement actions together with the reasoning steps we described in Section 4.1 token by token, where $\mathbf{o}_i = \{o_{i,1}, \ldots, o_{i,T}\}$ represents the sequence of tokens forming the full reasoning output at the i-th group of G samples. And the reward signal R_i for the i-th query sample is calculated through the reward function, as

illustrated in Eq. equation 8. ϵ is a small clipping hyperparameter that ensures stable updates by preventing the new policy from straying too far from the old policy. The policy optimization process is also elaborated in Algorithm 2.

4.3 Inference

DeepRefine is designed for an online KB evolution setting rather than answer-aware test-time optimization. As shown in Figure 3, during inference time, an incoming user query provides the refinement signal, while the policy has access only to the query and the evidence retrieved from the current KB. The resulting refinement actions are applied to the shared KB and can persist across subsequent interactions. Therefore, it maintains two asynchronous streams: an online query stream and a KB evolution stream. Given a query q_i, DeepRefine may identify and repair a local defect, producing an updated KB $\hat{\mathcal{G}}_{\mathrm{f}}^{i+1}$. Subsequent queries are then processed against this evolved KB, which allows historical interactions to accumulate into persistent knowledge improvements that can benefit future, previously unseen queries.

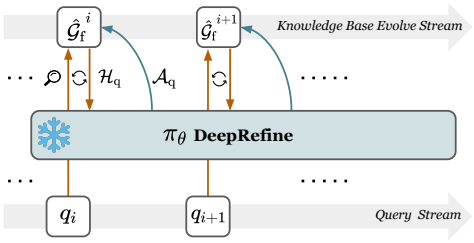

Figure 3: An overview of the inference process of DeepRefine.

5 Experiments

In this section, we will report the performance of DeepRefine on both RAG and long-term conversation memory benchmarks. We focus on the performance gain of DeepRefine over various baseline methods and the efficiency of the knowledge refinement process. We report the implementation and training details of DeepRefine in Appendix A.

5.1 Experimental Settings

Baselines. We select baselines from two perspectives. (1) We evaluate the performance gain from DeepRefine over various KB constructor: naive KB constructor, the RL fine-tuned constructor, which employs **AutoGraph-R1** (Tsang et al., 2025), and is a stronger constructor, and the agentic knowledge constructor, which is a version of LLM-Wiki and employs **Graphify** (Shamsi, 2026). Both the base and RL fine-tuned constructors share the same base model `Qwen2.5-3B-Instruct`. And we uses the `Composer-1.5` of Cursor as the agent in Graphify. (2) Conditioned on each of the above constructor, we evaluate whether the performance gains brought by DeepRefine remain stable across various knowledge retrievers. For triple retrievers, we employ **Subgraph Retriever** and **ToG** (Sun et al., 2024). For graph-based text retrievers, we employ **HippoRAG** (Jimenez Gutierrez et al., 2024) and **HippoRAG2** (Gutiérrez et al., 2025). We use the same settings of them as those in AutoGraph-R1.

Training Datasets. We randomly select $5,000$ samples from the training set of **HotpotQA** (Yang et al., 2018) to construct the training data for DeepRefine, and split them into training and validation sets with a ratio of $8{:}2$. For each original HotpotQA data point, we construct an individual KB. During the rollout process of the training, DeepRefine only refines the KB associated with the query in each sample individually.

Evaluation Datasets. We evaluate DeepRefine on three types of benchmarks, which include **Simple QA**, **Multi-hop QA** and **Conversation QA**. For simple QA, we use the Natural Questions (**NQ**) (Kwiatkowski et al., 2019) and **PopQA** (Mallen et al., 2023) datasets. For multi-hop QA, we use the **2WikiMultihopQA** (Ho et al., 2020) and **Musique** (Trivedi et al., 2022) datasets. For conversation QA, we use the **LOCOMO** (Maharana et al., 2024) dataset. For NQ and PopQA, we adopt a unified knowledge corpus constructed from the lead sections of the December 2021 Wikipedia dump (Izacard et al., 2023). For 2WikiMultihopQA and Musique, we construct benchmark-specific corpora from the documents associated with each dataset's $1,000$ evaluation instances, consistent with the settings of Jimenez Gutierrez et al. (2024). Similarly, for LOCOMO, we also sample $1,000$ evaluation instances. Following existing works (Trivedi et al., 2022; Jimenez Gutierrez et al., 2024; Yan et al., 2025), we employ the token-level F1 score as the metric for benchmark evaluation.

5.2 Experimental Results

Table 1: Averaged performance of our GBD reward finetuned DeepRefine on various knowledge retrievers. The table reports RAG F1 on five QA datasets, grouped by three constructors.

Methods	Simple QA		Multi-hop QA		Conversation	Overall
	NQ*	PopQA*	2WikiQA	Musique	LOCOMO	Avg.
Naive Constructor						
Subgraph	26.43	54.48	33.53	13.82	25.86	30.82
Subgraph + DeepRefine-4B	29.55	60.09	36.04	15.93	**29.64**	34.25$\uparrow_{3.43}$
Subgraph + DeepRefine-8B	**30.36**	**60.37**	**38.39**	**16.18**	29.07	**34.87**$\uparrow_{4.05}$
ToG	25.55	54.92	43.74	18.21	22.64	33.01
ToG + DeepRefine-4B	29.21	59.27	49.16	25.73	24.53	37.58$\uparrow_{4.57}$
ToG + DeepRefine-8B	**30.22**	**59.48**	**51.15**	**27.24**	**24.96**	**38.61**$\uparrow_{5.60}$
HippoRAG	35.54	63.56	49.77	26.54	**33.25**	41.73
HippoRAG + DeepRefine-4B	35.51	**64.97**	50.52	27.82	32.53	42.27$\uparrow_{0.54}$
HippoRAG + DeepRefine-8B	**36.35**	64.21	**51.17**	**29.07**	33.24	**42.81**$\uparrow_{1.08}$
HippoRAG2	35.91	63.08	51.47	25.33	33.33	41.82
HippoRAG2 + DeepRefine-4B	36.94	64.65	51.75	26.79	33.46	42.72$\uparrow_{0.90}$
HippoRAG2 + DeepRefine-8B	**37.32**	**64.91**	**53.18**	**27.19**	**33.55**	**43.23**$\uparrow_{1.41}$
RL Fine-Tuned Constructor						
Subgraph	29.27	58.40	35.60	14.42	26.65	32.87
Subgraph + DeepRefine-4B	31.71	62.14	37.76	16.94	**30.57**	35.82$\uparrow_{2.95}$
Subgraph + DeepRefine-8B	**33.31**	**62.25**	**38.45**	**18.10**	29.44	**36.31**$\uparrow_{3.44}$
ToG	29.88	60.34	48.56	19.06	23.64	36.30
ToG + DeepRefine-4B	32.53	63.51	**54.83**	27.09	29.05	41.40$\uparrow_{5.10}$
ToG + DeepRefine-8B	**33.42**	**63.92**	54.28	**27.82**	**29.45**	**41.78**$\uparrow_{5.48}$
HippoRAG	**39.84**	64.43	50.34	26.21	35.02	43.17
HippoRAG + DeepRefine-4B	39.76	65.66	**53.97**	29.00	**35.57**	**44.79**$\uparrow_{1.62}$
HippoRAG + DeepRefine-8B	39.73	**65.80**	53.41	**29.40**	35.36	44.74$\uparrow_{1.57}$
HippoRAG2	37.25	64.69	52.61	27.96	35.75	43.65
HippoRAG2 + DeepRefine-4B	38.81	65.67	57.29	**29.34**	**36.99**	**45.62**$\uparrow_{1.97}$
HippoRAG2 + DeepRefine-8B	**39.31**	**65.94**	**57.87**	28.16	35.87	45.43$\uparrow_{1.78}$
Graphify Constructor						
Subgraph	20.42	11.38	25.37	11.69	11.02	15.98
Subgraph + DeepRefine-4B	21.71	15.92	25.45	11.42	**13.64**	17.63$\uparrow_{1.65}$
Subgraph + DeepRefine-8B	**22.31**	**17.35**	**28.64**	**12.25**	13.59	**18.83**$\uparrow_{2.85}$
ToG	16.69	8.97	21.46	11.73	11.40	14.05
ToG + DeepRefine-4B	18.20	10.91	21.68	12.59	13.84	15.44$\uparrow_{1.39}$
ToG + DeepRefine-8B	**20.27**	**15.11**	**29.51**	**14.56**	**13.86**	**18.66**$\uparrow_{4.61}$
HippoRAG	13.23	5.57	29.03	8.67	3.44	11.99
HippoRAG + DeepRefine-4B	13.61	**5.89**	30.18	**9.25**	5.51	12.89$\uparrow_{0.90}$
HippoRAG + DeepRefine-8B	**14.05**	5.26	**36.54**	9.21	**5.98**	**14.21**$\uparrow_{2.22}$
HippoRAG2	13.62	6.19	27.98	8.10	3.57	11.89
HippoRAG2 + DeepRefine-4B	14.11	6.56	29.96	9.99	5.64	13.25$\uparrow_{1.36}$
HippoRAG2 + DeepRefine-8B	**14.53**	**8.42**	**39.33**	**11.55**	**6.27**	**16.02**$\uparrow_{4.13}$

Insight #1: ***DeepRefine brings consistent downstream gains over various KB constructors and retrievers with better knowledge carrying and knowledge indexing capabilities.*** We refine the KBs constructed by the naive constructor, the RL fine-tuned constructor and the Graphify constructor. As shown in Table 1, with DeepRefine, the quality of the KBs can be stably improved for their better downstream performance. This suggests the refinement actions generated by the DeepRefine's policy can indeed improve the quality of KBs. And DeepRefine-8B can perform better than DeepRefine-4B, which demonstrates the effectiveness can be improved by a larger scale model. We also demonstrate some cases about how DeepRefine can resolve issues in the constructed KBs in Appendix F. Besides, we can also observe that DeepRefine introduces more performance gain for triple retrievers than text retrievers. This is because the refinement actions are triple-level operators and will not change the original text chunks. So the performance gains for triple retrievers are typically from both content and structure optimizations and the gains for text retrievers are only from structure optimization. It also suggests that DeepRefine can not only evolve the KB's knowledge carrying capabilities, but also the knowledge indexing quality. We also conduct held-out query evaluation in Appendix E to further

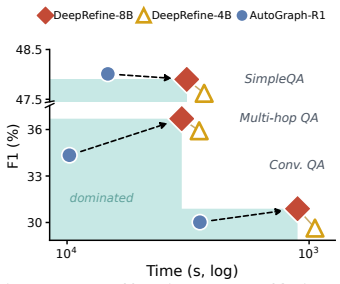

Figure 4: Effectiveness-efficiency comparisons of DeepRefine versus AutoGraph-R1.

validate that query-guided refinement accumulates as KB-level improvements rather than overfitting only to the queries used to drive the agent.

Insight #2: ***DeepRefine can introduce larger or competitive downstream gains than the reconstruction method.*** As shown in Table 1 and Figure 4, the downstream performance of the KBs constructed by a naive knowledge constructor can even outperform the reconstruction methods (AutoGraph-R1, which is also RL fine-tuned) in most cases after they are optimized by `DeepRefine-8B`. For example, in 2WikiQA, the performance of "ToG+`DeepRefine-8B`" with the naive constructor is much higher than that of "ToG" with the RL fine-tuned constructor (51.15 vs. 48.56). This is mainly because the issues may not always happen in the whole KB, but may only in some smaller regions. Regenerating the whole KB may introduce additional noise that may harm the downstream performance.

Insight #3: ***DeepRefine is more efficient than reconstructing the full KB.*** We collect the average time consumptions of both the refinement method DeepRefine and the reconstruction method AutoGraph-R1. As shown in Table 2, on all three different kinds of benchmarks, DeepRefine exhibits obvious superiority with respect to the reconstruction method. This is mainly because instead of taking into account all documents and regenerating the whole KB, the agentic design of DeepRefine makes the model will only focus on the important and potentially problematic parts, which largely reduces the amount of information that is needed to be processed by the model. Consequently, according to Figure 4, taking into account both efficiency and effectiveness, DeepRefine is a better choice to improve the quality of KBs than the reconstruction methods.

Table 2: Average time consumptions (s) ↓ of DeepRefine vs. AutoGraph-R1.

Methods	Simple QA	Multi-hop QA	Conversation QA
AutoGraph-R1	6,782.8	9,780.4	2,826.5
DeepRefine	**3,201.7**	**3,357.5**	**1,115.8**

5.3 Ablation Study

Table 3: Performance comparisons between the DeepRefine full framework and other variants under the naive constructor.

Variants	Simple QA		Multi-hop QA		Conversation	Overall
	NQ*	PopQA*	2WikiQA	Musique	LOCOMO	Avg.
`DeepRefine-4B`	**29.38**	**59.68**	**42.60**	**20.83**	**27.09**	**35.92**
`DeepRefine-4B` *w/o RL*	28.74	58.89	41.37	19.86	26.89	35.15
`DeepRefine-4B` *w/o Abduction*	28.82	58.86	41.66	19.78	25.41	34.91
`DeepRefine-8B`	**30.29**	**59.93**	**44.77**	**21.71**	**27.02**	**36.74**
`DeepRefine-8B` *w/o RL*	29.19	59.73	42.71	20.62	25.08	35.47
`DeepRefine-8B` *w/o Abduction*	28.36	59.17	42.85	18.91	25.37	34.93

To demonstrate the effectiveness and necessity of RL training and the error abduction reasoning process in DeepRefine, we compare the performance between the full framework and different variants. We remove the error abduction reasoning of the refinement process, which is denoted as `DeepRefine-8B/4B` *w/o Abduction* and also test DeepRefine without RL training, which is denoted as `DeepRefine-8B/4B` *w/o RL*. The performance is the average of two triple retrievers under the naive constructor, as triple retrievers are more consistent with the training settings and can reflect both of the knowledge indexing and knowledge carrying capabilities. We also further construct a benchmark and verify the effectiveness of the RL training for resolving the incompleteness, incorrectness and redundancy issues, respectively. Benchmark construction details are elaborated in Appendix C.

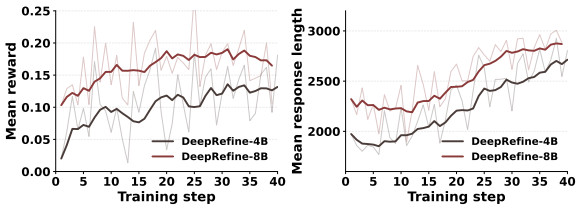

Figure 5: GBD reward and response length curves of `DeepRefine-4B/8B` during RL training.

Insight #4: ***Error Abduction reasoning is useful for agentic knowledge refinement.*** As shown in Table 3, after we remove the error abduction reasoning from the refinement process, the performance will drop a lot. It is mainly because, similar to CoT (Wei et al., 2022), it can introduce explicit intermediate hypotheses about potential issues in the retrieved evidences, which provide guidance to the refinement action generation, hence yielding better refinement quality.

Insight #5: ***RL training with GBD reward can introduce consistent performance gain.*** As shown in Table 3, removing RL training would lead to lower performance, especially for multi-hop QA and conversation QA. Because without RL optimization, it would be hard for the agent to fully utilize the interaction history to generate useful actions. As shown in Figure 5, we also demonstrate the training dynamics with average GBD reward and response length curves of DeepRefine. As the training progress goes on, the rewards are steadily increasing. And the model will also first learn to reduce the response length by judging with more answerable queries to avoid reward deduction, but then it learns to explore longer trajectories and reach a better refinement policy π_θ.

Insight #6: ***DeepRefine learns issue-specific action specialization.*** We quantify specialization via incidence lift $L(t_\mathcal{A}, t_\mathcal{I}) = P(t_\mathcal{A} \mid t_\mathcal{I})/P(t_\mathcal{A} \mid \bar{t_\mathcal{I}})$, i.e., how much more often action type $t_\mathcal{A}$ appears on its target issue types $t_\mathcal{I}$ (Appendix G) than on others $\bar{t_\mathcal{I}}$. Figure 6 shows `delete_edge` is $\times 22 \sim 33$ more likely for incorrectness, and `replace_node` is $\times 2.75 \sim 3$ more likely for incorrectness and redundancy. Both effects are stronger with DeepRefine than with the prompted base model. `insert_edge` remains near $\times 1$ because adding missing facts is a generic fix across error types.

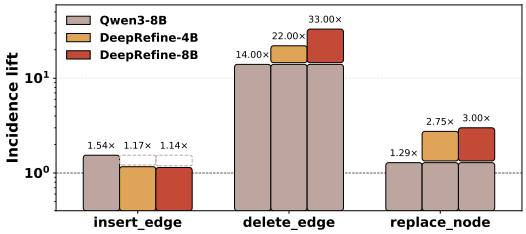

Figure 6: Incidence lift on three types of operators among different refinement models.

Insight #7: ***The base model is preliminary effective for resolving incompleteness and incorrectness, but to a lesser degree for resolving redundancy issues.*** As shown in Figure 7, the base model `Qwen3-8B` that employs the DeepRefine's agentic framework can resolve some incompleteness and incorrectness issues to some degree. But it can hardly resolve the redundancy issues. Specifically, the incompleteness issue degrades performance very hard and can also be improved the most easily. The incorrectness issue can hurt performance the most, and it is harder to be resolved. Because it requires the model to identify the incorrect information and then correct it according to the source passages. Besides, the redundancy issue degrades the downstream performance the mildest, and it is also the hardest to be resolved. Because it requires the model to do the coreference resolution and disambiguation, which is more challenging than the other two tasks.

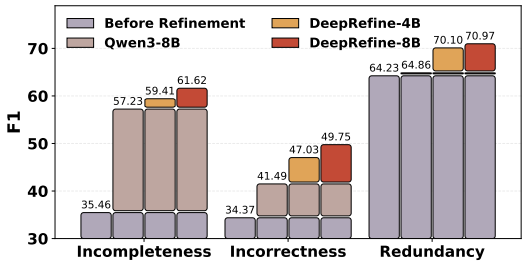

Figure 7: Effectiveness of DeepRefine and base models on resolving three types of issues.

Insight #8: ***RL training with GBD reward can introduce better effectivenss for resolving incompleteness, incorrectness and redundancy issues.*** As shown in Figure 7, both of `DeepRefine-8B` and `DeepRefine-4B` that are optimized on the GBD reward are more effective than the base model `Qwen3-8B` in resolving three types of issues. Specifically, the superiority is more significant for the incorrectness and redundancy issues, which are harder to be resolved as we mentioned before. The base model has almost no effectiveness for fixing the redundancy issue (only $\sim 0.63\%$), however, the RL fine-tuned models can improve the performance to a certain extent ($\sim 6\%$), which is $\times 10$ better than the base model. And the RL fine-tuned models also exhibit superior performance over the base model for resolving the incompleteness issues. And the results can also suggest that not only DeepRefine can localize the problematic parts in KBs, but also the RL fine-tuning strategy can further improve the refinement effectiveness through solving the three potential issues.

6 Conclusion

In this paper, we propose DeepRefine, an RL framework for agentic knowledge refinement. DeepRefine is designed to refine pre-constructed structured KBs by leveraging the multiple interactions with the KB, which is trained with a newly designed GBD reward finetuned RL policy. DeepRefine makes the KBs evolvable with historical user interactions. Experimental results show that DeepRefine can consistently improve the quality of the constructed KB and achieve better downstream performance compared with that of the original KB.

7 AI DISCLOSURE

In this work, we used generative AI tools to generate synthetic datasets for benchmark construction. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, assisting with translation, cleaning and reformatting datasets, supporting qualitative and thematic data analysis, or interpreting results. The remaining required-disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to edit the manuscript to improve readability (grammar, wording, and clarity). All AI-assisted outputs, including the generated benchmark data and the manuscript edits, were reviewed and verified by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

8 ETHICS STATEMENT

We affirm adherence to the ICLR Code of Ethics. This work develops DeepRefine does not involve human subjects or interventions. We use publicly available datasets (e.g., 2WikiMultiHopQA, HotpotQA, 2021Wiki, Musique, and LOCOMO) under their licenses. We construct the evaluation benchmark via HotpotQA without collecting new personal data or attempting deanonymization. No undisclosed conflicts of interest exist. All experiments were performed on 8$\times$ A800 GPUs, and we encourage energy-efficient configurations.

9 REPRODUCIBILITY STATEMENT

LLMs for Training and Evaluation. Three LLMs are used in our DeepRefine experiments: Qwen3-8B, Qwen3-4B-Instruct-2507en, Qwen3-Embedding-0.6B, and Qwen2.5-7B-Instruct. We use Qwen3-8B and Qwen3-4B-Instruct-2507en as the backbone of DeepRefine and it can be obtained at Hugging Face.

Training and Evaluation Details. We provide comprehensive descriptions about our training and evaluation settings in Section 5.1, including the hyper-parameter settings and detailed processing steps. All of the datasets we used in our work can be obtained through public resources.

REFERENCES

Shengyuan Chen, Zheng Yuan, Qinggang Zhang, Wen Hua, Jiannong Cao, and Xiao Huang. Neuro-symbolic entity alignment via variational inference. *arXiv preprint arXiv:2410.04153*, 2024.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Prafulla Kumar Choubey, Xin Su, Man Luo, Xiangyu Peng, Caiming Xiong, Tiep Le, Shachar Rosenman, Vasudev Lal, Phil Mui, Ricky Ho, et al. Distill-synthkg: Distilling knowledge graph synthesis workflow for improved coverage and efficiency. *arXiv preprint arXiv:2410.16597*, 2024.

Philipp Cimiano and Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semant. Web*, 8(3):489–508, January 2017. ISSN 1570-0844. doi: 10.3233/SW-160218. URL https://doi.org/10.3233/SW-160218.

Na Dong, Natthawut Kertkeidkachorn, Xin Liu, and Kiyoaki Shirai. Refining noisy knowledge graph with large language models. In *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*, pp. 78–86, 2025.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Zhifan Feng, Qi Wang, Wenbin Jiang, Yajuan Lyu, and Yong Zhu. Knowledge-enhanced named entity disambiguation for short text. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pp. 735–744, 2020.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, Haofen Wang, et al. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1):32, 2023.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tian Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*, 2(3), 2024.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*, 2025.

Aric Hagberg and Drew Conway. Networkx: Network analysis with python. *URL: https://networkx. github. io*, 1031, 2020.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Haoyu Huang, Yongfeng Huang, Junjie Yang, Zhenyu Pan, Yongqiang Chen, Kaili Ma, Hongzhi Chen, and James Cheng. Retrieval-augmented generation with hierarchical knowledge. *arXiv preprint arXiv:2503.10150*, 2025.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43, 2023.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Pengcheng Jiang, Xueqiang Xu, Jiacheng Lin, Jinfeng Xiao, Zifeng Wang, Jimeng Sun, and Jiawei Han. s3: You don't need that much data to train a search agent via rl. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21610–21628, 2025.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems*, 37:59532–59569, 2024.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

Andrej Karpathy. LLM Wiki: A pattern for building personal knowledge bases using LLMs. GitHub Gist, April 2026. URL https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

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.

Haoran Luo, Guanting Chen, Qika Lin, Yikai Guo, Fangzhi Xu, Zemin Kuang, Meina Song, Xiaobao Wu, Yifan Zhu, Luu Anh Tuan, et al. Graph-r1: Towards agentic graphrag framework via end-to-end reinforcement learning. *arXiv preprint arXiv:2507.21892*, 2025.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*, 2024.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 9802–9822, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

Mohammad Javad Saeedizade, Najmeh Torabian, and Behrouz Minaei-Bidgoli. Kgrefiner: Knowledge graph refinement for improving accuracy of translational link prediction methods. In *Proceedings of the Third Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, pp. 10–16, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Safi Shamsi. Graphify. https://github.com/safishamsi/graphify, 2026. GitHub repository.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

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

Budhitama Subagdja, D Shanthoshigaa, Zhaoxia Wang, and Ah-Hwee Tan. Machine learning for refining knowledge graphs: A survey. *ACM Computing Surveys*, 56(6):1–38, 2024.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *International Conference on Learning Representations*, volume 2024, pp. 3868–3898, 2024.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition, 2022. URL https://arxiv.org/abs/2108.00573.

Hong Ting Tsang, Jiaxin Bai, Haoyu Huang, Qiao Xiao, Tianshi Zheng, Baixuan Xu, Shujie Liu, and Yangqiu Song. Autograph-r1: End-to-end reinforcement learning for knowledge graph construction. *arXiv preprint arXiv:2510.15339*, 2025.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-{\alpha}: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 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.

Benedict Wolff and Jacopo Bennati. Cost and accuracy of long-term graph memory in distributed llm-based multi-agent systems. *arXiv preprint arXiv:2601.07978*, 2026.

Yunjia Xi, Jianghao Lin, Yongzhao Xiao, Zheli Zhou, Rong Shan, Te Gao, Jiachen Zhu, Weiwen Liu, Yong Yu, and Weinan Zhang. A survey of llm-based deep search agents: Paradigm, optimization, evaluation, and challenges. *arXiv preprint arXiv:2508.05668*, 2025.

Zhongwei Xie, Xiyu Ren, Tianshi Zheng, and Jiaxin Bai. A survey on ai agent harness.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chang Yang, Chuang Zhou, Yilin Xiao, Su Dong, Luyao Zhuang, Yujing Zhang, Zhu Wang, Zijin Hong, Zheng Yuan, Zhishang Xiang, et al. Graph-based agent memory: Taxonomy, techniques, and applications. *arXiv preprint arXiv:2602.05665*, 2026.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Liang Yao, Chengsheng Mao, and Yuan Luo. Kg-bert: Bert for knowledge graph completion. *arXiv preprint arXiv:1909.03193*, 2019.

Yaoze Zhang, Rong Wu, Pinlong Cai, Xiaoman Wang, Guohang Yan, Song Mao, Ding Wang, and Botian Shi. Leanrag: Knowledge-graph-based generation with semantic aggregation and hierarchical retrieval. *arXiv preprint arXiv:2508.10391*, 2025.

Xinkui Zhao, Haode Li, Yifan Zhang, Guanjie Cheng, and Yueshen Xu. Trail: Joint inference and refinement of knowledge graphs with large language models. *arXiv preprint arXiv:2508.04474*, 2025.

Appendix of DeepRefine

CONTENTS

A Implementation Details **15**

B Coverage-Based Query Selection **15**

C Benchmark Construction **15**

D Training and Inference Algorithms **16**

E Held-out Query Evaluation **17**

F Case Study **18**

F.1 Case 1: Incompleteness 18

F.2 Case 2: Incorrectness 18

F.3 Case 3: Redundancy 19

G Rationality of the Action Space Design **20**

H Training System Overview of DeepRefine **21**

I Prompt Templates **22**

J Limitations **23**

A Implementation Details

We implement the DeepRefine training and inference processes using both of the `Qwen3-8B` and `Qwen3-4B-Instruct-2507en` as the backbone model for their better instruction following capabilities. And we employ the `Qwen3-Embedding-0.6B` as the embedding model for all vector search in both our method and the baseline methods and the frozen `Qwen2.5-7B-Instruct` as the reader model for the RAG generation. For both of the 4B and 8B models, we use a batch size of 64 with a mini-batch size of 32 and a group size of 16 for the 40 and 30 steps' GRPO training, respectively. The learning rate is $1e$-6 and the micro-batch size is 4. To further improve the efficiency of the refinement process, we select a subset of queries before refinement using a greedy maximum-coverage procedure over query-related triples, as described in Appendix B. Our models are trained on $8\times$ NVIDIA A800-SXM4-80GB GPUs. Results are averaged under 3 times.

B Coverage-Based Query Selection

Considering there could be overlappings among the effective regions of the refinement processes with different queries in each benchmark dataset, to make the knowledge refinement process both efficient and effective, we select a small subset of queries before refinement using a **greedy maximum-coverage procedure** over query-related triples. For each candidate query respect to the full knowledge base in a benchmark dataset, we obtain a set of triples by (i) taking the top-k related triples with their sentence embeddings, and (ii) augmenting with up to m additional triples drawn from the one-hop neighborhoods of the entities appearing in those triples, where neighborhood triples are ranked by embedding similarity to the query and the highest-scoring ones are kept. Treating each distinct triple as a coverage element, we greedily choose queries that maximize the number of newly covered elements until either a budget of B queries is reached or the fraction of covered elements in the union of all candidates' sets exceeds a target threshold ρ. Only the selected queries are then passed through the refinement pipeline to update the shared knowledge base. Then all benchmark queries are evaluated on the resulting knowledge base with the standard RAG pipeline. The corresponding hyperparameter values are listed in Table 4. For LOCOMO dataset, m is set to 100 and ρ is set to 1.0. For other larger datasets, m is set to 500 and ρ is set to 0.8.

Table 4: Hyperparameters for greedy maximum-coverage query selection.

Parameter	Value
k	10
m	100 / 500
B	1000
ρ	1.0 / 0.8

C Benchmark Construction

To further evaluate the effectiveness of DeepRefine on resolving incompleteness, incorrectness or redundancy issues, we provide a rigorous evaluation. We construct a benchmark characterized by explicit error types and diverse corruption patterns. By editing the knowledge bases with controlled defects that mirror real-world failure modes, it enables a precise assessment of a model's performance in defect localization and refinement processes across various error types.

We constuct the benchmark based on the HotpotQA (Dev-Distractor) (Yang et al., 2018) dataset, which is different from the training set of HotpotQA that is used for our RL training. It consists of 501 samples, with 167 samples dedicated to each of the three systematic error types defined in our study. Each data point includes the original query, the corrupted knowledge base, and comprehensive metadata (e.g., corruption details and action counts). The construction pipeline utilizes `Qwen2.5-32B-Instruct` (denoted as the corruption model) for generating defects, `Qwen2.5-3B-Instruct` as the naive knowledge base constructor, and `Qwen3-Embedding-0.6B` for vector-based retrieval. `Qwen2.5-7B-Instruct` serves as the reader model for RAG-based evaluation.

Error Types and Corruption Strategies. For each error type, we apply a specific corruption strategy to a high-quality knowledge base: *Incompleteness*: We identify and remove critical triples that are essential for the reasoning path, creating "missing evidence" scenarios. *Incorrectness*: We modify correct knowledge items into incorrect claims based on the original context, simulating low-confidence or imprecise information. *Redundancy*: This is achieved via two methods: (1) Inserting

semantically similar but redundant triples to introduce ambiguity. (2) Replacing entities in existing triples with aliases or synonymous names to create coreference resolution challenges.

Construction Pipeline. The construction pipeline follows a rigorous multi-stage pipeline. Initially, for each sample, we construct an individual KB based on its context. We then perform RAG. Only samples achieving a F1 score higher than 0.95 are retained. This ensures that the initial KB is relatively accurate and contains sufficient information to answer the query. Then, for the filtered samples, the corruption process is specifically targeted at the top-N triples retrieved during the previous RAG stage. This ensures that the induced defects are concentrated on the critical knowledge components most relevant to the query, rather than introducing peripheral noise. Within this focused scope, the corruption model generates a series of actions. To ensure the generation of ideal, high-quality defects, we designed specialized prompt templates and carefully curated few-shot samples tailored to each specific error type. These actions are subsequently parsed into three predefined operators: `insert_edge()`, `delete_edge()`, and `replace_node()`. The action space is designed to be symmetric with DeepRefine's refinement action set to facilitate direct performance mapping. Notably, the replace_node() operator is restricted to acting upon a single, specific edge. This fine-grained precision allows us to create nuanced coreference resolution issues and subtle semantic defects that test the model's diagnostic depth. To maintain the structural integrity and ensure the refinability of the knowledge base, the total number of corruption actions is strictly limited to 5 per sample. Then we perform a second RAG evaluation on the corrupted KB. Only samples where the RAG F1 score drops below 0.6 are included in the final benchmark. This ensures that the corruption is effective, which suggests that it significantly impairs the model's ability to answer the query without refinement.

D Training and Inference Algorithms

We summarize the three-step reasoning rollout, GRPO training with the GBD reward, and deployment-time inference in Algorithms 1–3. As shown in Algorithm 1, note that according to prior work (Jin et al., 2025),

Algorithm 1 DeepRefineRollout(q, $\mathcal{G}_\mathrm{f}$, π_θ, L_max, N, M)

Input: Query q, per-query draft KB $\mathcal{G}_\mathrm{f}$ (each query has its own separated sub-KB), policy π_θ, hop limit L_max, top-N / top-M retrieval sizes.
Output: Multi-turn reasoning output $\mathbf{o}$ with refinement actions $\mathcal{A}_\mathrm{q}$.

$\mathcal{G}_\mathrm{q}^{(0)} \leftarrow$ Top-$k(q, \mathcal{G}_\mathrm{f}, N)$; $\mathcal{H}_\mathrm{q} \leftarrow \emptyset$; $i \leftarrow 0$
while $i < L_\mathrm{max}$ **do**
  $\mathcal{J}_\mathrm{q}^{(i)}, \mathbf{o} \leftarrow \mathrm{Judge}_\theta(q, \mathcal{G}_\mathrm{q}^{(i)})$ ▷ output in <judge> tags
  Append $(q, \mathcal{G}_\mathrm{q}^{(i)}, \mathcal{J}_\mathrm{q}^{(i)})$ to $\mathcal{H}_\mathrm{q}$
  if $\mathcal{J}_\mathrm{q}^{(i)} =$ Yes **then**
    break
  end if
  $\mathcal{G}_\mathrm{q}^{(i+1)} \leftarrow \mathcal{G}_\mathrm{q}^{(i)} \cup$ Top-$k\big(q, \mathcal{G}_\mathrm{cand}^{(i+1)}, M\big)$ ▷ one-hop neighbor collection in Eqs. equation 3–equation 5
  $i \leftarrow i + 1$
end while
if $\mathcal{J}_\mathrm{q}^{(0)} \neq$ Yes **then**
  $\mathcal{I}_\mathrm{q}, \mathbf{o} \leftarrow \mathrm{Abduce}_\theta(\mathcal{H}_\mathrm{q})$ ▷ output in <abduction> tags
  $\mathcal{A}_\mathrm{q}, \mathbf{o} \leftarrow \mathrm{Refine}_\theta(\cdot \mid \mathcal{G}_\mathrm{q}^{(i)}, \mathcal{I}_\mathrm{q})$ ▷ output in <refinement> tags
else
  $\mathcal{A}_\mathrm{q} \leftarrow \emptyset$
end if

Algorithm 2 DEEPREFINEGRPO$\left(\mathcal{Q}_{\text{train}},\ \{\mathcal{G}_{\text{f}}^{(q)}\},\ \pi_\theta,\ G,\ \epsilon\right)$

Input: Training queries $\mathcal{Q}_{\text{train}}$, per-query draft KBs $\{\mathcal{G}_{\text{f}}^{(q)}\}$, policy π_θ, frozen reader model $\textsc{Reader}(\cdot)$, group size G, clipping hyperparameter ϵ.
Output: Optimized refinement policy π_θ.

for each GRPO training step **do**
  for each query q in the mini-batch sampled from $\mathcal{Q}_{\text{train}}$ **do**
    $\mathcal{G}_{\text{f}} \leftarrow \mathcal{G}_{\text{f}}^{(q)}$; $A \leftarrow$ golden answer of q; $A_{\text{draft}} \leftarrow \textsc{Reader}(q, \mathcal{G}_{\text{f}})$
    for $i = 1$ **to** G **do**
      $\mathbf{o}_i, \mathcal{A}_{\text{q}}^{(i)} \leftarrow \textsc{DeepRefineRollout}(q, \mathcal{G}_{\text{f}}, \pi_{\theta_{\text{old}}})$ ▷ Alg. 1
      $\hat{\mathcal{G}}_{\text{f}}^{(i)} \leftarrow \textsc{ApplyActions}(\mathcal{G}_{\text{f}}, \mathcal{A}_{\text{q}}^{(i)})$
      $A_{\text{refined}}^{(i)} \leftarrow \textsc{Reader}(q, \hat{\mathcal{G}}_{\text{f}}^{(i)})$
      $R_i \leftarrow \text{F1}(A_{\text{refined}}^{(i)}, A) - \text{F1}(A_{\text{draft}}, A)$
    end for
    $\mu_R \leftarrow \frac{1}{G}\sum_{i=1}^{G} R_i$; $\sigma_R \leftarrow \textsc{Std}(\{R_i\}_{i=1}^{G})$
    for $i = 1$ **to** G **do**
      for $t = 1$ **to** $|\mathbf{o}_i|$ **do**
        $\hat{A}_{i,t} \leftarrow (R_i - \mu_R)/\sigma_R$
        $r_{i,t}(\theta) \leftarrow \frac{\pi_\theta(o_{i,t}|q,\mathbf{o}_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q,\mathbf{o}_{i,<t})}$
        Accumulate clipped GRPO objective using $r_{i,t}(\theta)$ and $\hat{A}_{i,t}$
      end for
    end for
  end for
  Update θ by gradient ascent on the GRPO objective
end for

Algorithm 3 DEEPREFINEINFER$(\mathcal{G}_{\text{f}},\ \mathcal{Q},\ \pi_\theta,\ k,\ m,\ B,\ \rho)$

Input: Pre-constructed KB $\mathcal{G}_{\text{f}}$, query set $\mathcal{Q}$, frozen policy π_θ, coverage hyperparameters (k, m, B, ρ).
Output: Evolved KB $\mathcal{G}$ for downstream RAG.

$\mathcal{G} \leftarrow \mathcal{G}_{\text{f}}$
$\mathcal{Q}_{\text{refine}} \leftarrow \textsc{GreedyCoverageSelect}(\mathcal{Q}, \mathcal{G}, k, m, B, \rho)$ ▷ Appendix B
for each $q \in \mathcal{Q}_{\text{refine}}$ **do**
  $\mathbf{o}, \mathcal{A}_{\text{q}} \leftarrow \textsc{DeepRefineRollout}(q, \mathcal{G}, \pi_\theta)$ ▷ Alg. 1
  if $\mathcal{A}_{\text{q}} \neq \emptyset$ **then**
    $\mathcal{G} \leftarrow \textsc{ApplyActions}(\mathcal{G}, \mathcal{A}_{\text{q}})$
  end if
end for

E HELD-OUT QUERY EVALUATION

In our experiments in Section 5.1, the results have suggested that refining the KBs with only a small set of queries (from $\sim 10\%$ to $\sim 40\%$ of the queries in the benchmarks) can lead to a general performance gain under all queries in the benchmarks. In this section, to further demonstrate the refinement actions applied to the KBs are also useful for different query sets, we refine only on the selected subset, then evaluate ToG and Subgraph retrievers on the complementary holdout, comparing the original KB with the KB after refinement.

Table 5: Query-select refinement on 2WikiMultihopQA and Musique, evaluated on queries excluded from the refine subset.

Methods	2WikiQA	Musique
Subgraph	32.51	13.51
Subgraph + DeepRefine-4B	34.38$\uparrow_{1.87}$	**20.08**$\uparrow_{6.57}$
Subgraph + DeepRefine-8B	**34.53**$\uparrow_{2.02}$	18.07$\uparrow_{4.56}$
ToG	42.58	16.78
ToG + DeepRefine-4B	**44.85**$\uparrow_{2.27}$	**29.71**$\uparrow_{12.93}$
ToG + DeepRefine-8B	44.11$\uparrow_{1.53}$	22.64$\uparrow_{5.86}$

We use the same coverage-based selector as in the main experiments, with $(\rho, B) = (0.8, 500)$ on 2WikiMultihopQA and MuSiQue (1,000 queries

each). This selects 446 queries for `DeepRefine-4B / 8B` on 2Wiki (holdout 554) and 413 on MuSiQue (holdout 587). As shown in Table 5, the same refinement actions remain useful for queries that were never selected. Both retrievers improve on both datasets under `DeepRefine-4B / 8B`. The holdout result therefore supports that query-guided refinement accumulates as KB-level improvements rather than overfitting only to the queries used to drive the agent.

F Case Study

In this section, we will provide a case study to demonstrate the effectiveness of DeepRefine. We will give some cases to show how DeepRefine can refine the knowledge base to deal with the potential incompleteness, incorrectness or redundancy issues.

F.1 Case 1: Incompleteness

Incompleteness is the most common issue in the knowledge base. It is not possible to connect all potentially relevant knowledge items in the real world, which makes it necessary to do the completions that are useful for the downstream tasks. As shown in Table 6, the question asks for the 2010 population of the city that is popular with tourists. In the original query-related knowledge items, there are tourism-related clues and several population facts, but the key bridge from the target city to its 2010 population is missing. DeepRefine addresses this incompleteness by inserting series of missing population edges (e.g., city → "population in 2010" → value), which shortens the reasoning path and makes the answer directly retrievable.

Original Passages	***Query***	***Query-Related Triples***
... Instead, Nanjing, as a popular tourist destination, hosts a series of government-organised events throughout the year... The total population of the City of Nanjing reached 8.005 million in 2010. The statistics in 2011 estimated the total population to be 8.11 million... Camdenton is a city in Camden County, Missouri, United States. The population was 3,718 at the 2010 census... ...	What is the population in 2010 of the city popular with tourists?	... <*'Water tourism inside Strasbourg', 'attracts', 'hundreds of thousands of tourists yearly'*>, <*'Oklahoma', 'included four cities over 100,000 in population', 'in 2010'*>, <*'New York City', 'population reached all-time high', 'in the 2010 Census'*>, <*'Nanjing', 'has annual attraction', 'thousands of tourists'*>, <*'Oklahoma City', 'had the largest metropolitan area in the state in 2010', 'with 1,252,987 people'*>
Refinement Actions		
<refinement>insert_edge(*'Palermo', 'population in 2010', '1.2 million'*)|insert_edge(*'Nanjing', 'population in 2010', '8.005 million'*)|insert_edge(*'Tranquillity', 'population in 2010', '799'*)|insert_edge(*'Quincy', 'population in 2010', '7,972'*)|insert_edge(*'York', 'population in 2010', '910'*)|insert_edge(*'Oklahoma City', 'population in 2010', '620,602'*)|insert_edge(*'New York City', 'population in 2010', '8.17 million'*)|insert_edge(*'Seattle', 'population in 2010', '608,000'*)|insert_edge(*'Oklahoma', 'population in 2010', '3.8 million'*)|</refinement>		

Table 6: The case of refinement for incompleteness issue.

F.2 Case 2: Incorrectness

Errors usually come from the incorrect or conflicting information in the original passages. As shown in Table 7, the error in this case is a factual contradiction introduced by a misalignment between the knowledge base content and the ground truth evidence. Specifically, the retrieved query-related triples contains a wrong edge <*'Kree Harrison', 'took runner-up spot', 'American Idol (season 3)'*> which contradicts the gold evidence stating that Diana DeGarmo was the runner-up of that season.

If this incorrect edge is preserved, the model will inevitably reach the wrong answer regarding the contestant's identity and subsequent reasoning hops.

DeepRefine handles this through its refinement mechanism: it identifies the mismatch and applies `delete_edge` as well as `insert_edge` to restore the correct factual link <*'Diana DeGarmo', 'runner-up of', 'American Idol (season 3)'*> while effectively overriding the incorrect noise. This ensures the reasoning chain remains faithful to the supporting paragraphs.

Original Passages	***Query***	***Query-Related Triples***
... Fantasia and **Diana DeGarmo** were the last two finalists, and Fantasia was crowned as the winner. The third season was won by Fantasia Barrino, who defeated **Diana DeGarmo**... ...	Which season was the runner up of the third season of American Idol on?	... <*'American Idol (season 3)', 'winner', 'Fantasia Barrino'*> , <*'American Idol (season 3)', 'runner-up', 'Kree Harrison'*> , <*'Kree Harrison', 'took runner-up spot', 'American Idol'*>, ...
Refinement Actions		
<refinement>delete_edge(*'American Idol (season 3)', 'runner-up', 'Kree Harrison'*) | insert_edge(*'American Idol (season 3)', 'runner-up', 'Diana DeGarmo'*) | insert_edge(*'Diana DeGarmo', 'runner-up of', 'American Idol (season 3)'*)</refinement>		

Table 7: The case of refinement for Incorrectness issue involving factual contradiction.

Original Passages	***Query***	***Query-Related Triples***
... **James:** Yesterday I took my three dogs to a beach outing to have fun and bond with other dogkeepers. **John:** Cool! Surely you gained a new experience from communicating with other dog lovers! **James:** Yes, we had fun and I even met one beautiful girl. I'm thinking of asking her out on a date! She left me her phone number, I think I'll call tomorrow. **John:** Wow! That's cool, what's her name? Be sure to call her, everything will work out! **James:** She is Samantha. I'll definitely call her! **John:** Yoohoo! Hope you have a wonderful time! ...	Whose phone number did James receive during the beach outing?	... <*'James', 'left', 'the girl's phone number'*>, <*'James', 'announced', 'decision to move in together with Samantha'*>, <*'James', 'took his three dogs to', 'beach outing'*>, <*'James', 'received', 'congrats James! That's awesome.'*>, <*'John', 'wishing', 'James a great time'*>, ...
Refinement Actions		
<refinement>replace_node(*'the girl's phone number', 'Samantha's phone number'*)|insert_edge(*'James', 'received', 'Samantha's phone number'*)|</refinement>		

Table 8: The case of refinement for coreference resolution issue.

F.3 Case 3: Redundancy

We mainly focus on two types of redundancy issues: coreference resolution and disambiguation. And we will give some examples to show how DeepRefine can deal with these issues.

Coreference resolution. As shown in Table 8, the original knowledge base contains a coreference issue: According to the original passages, *'the girl's phone number'* is extracted as an knowledge item in the query-related triples, and is referenced by both James. However, as the query is about whose phone number did James receive during the beach outing, even with some contextual information in the query-related triples, the models still cannot figure out who is *'the girl'* mentioned in the triple. DeepRefine can correctly resolve this issue by first locating the potentially problematic region, then replacing *'the girl's phone number'* with *'Samantha's phone number'* and inserting a new edge between James and Samantha's phone number.

Disambiguation. As shown in Table 9, the original knowledge base contains a potential disambiguation issue: The person in the original document, *'Ray Taylor'*, is extracted exactly as the same name string in the knowledge base. However, this simple extraction strategy may cause the ambiguity of the knowledge base, for example, there could be another person with the same name string in the knowledge base, which may introduce some structural noise to the related triples. DeepRefine can correctly resolve this issue by first locating the potentially problematic subgraph, then replacing *'Ray Taylor'* with *'Ray Taylor (1888-12-01 to 1952-02-15)'*, which smartly handles the disambiguation issue by adding more specific information to the knowledge items in the query-related triples. In this way, if any new people with the same name appear, the knowledge base will not confuse them.

Original Passages	***Query***	***Query-Related Triples***
... Ray Taylor (1 December 1888 - 15 February 1952) was an American film director. He directed 159 films between 1926 and 1949. His debut was the 1926 film serial Ḟighting with Buffalo Bill.̤ Ravina is an Indian actress who acted in Dhallywood movies. She acted inP̈raner Cheye Priyoẅhich film is considered as turning point of the career of Riaz. ...	Which film has the director died later, Modern Husbands or The Fighting Vigilantes?	... <*'"Fighting with Buffalo Bill"'*, *'was directed by'*, *'Ray Taylor'*>, <*'Modern Husbands'*, *'starring'*, *'Olinda Bozán'*>, <*'Ray Taylor'*, *'debut film'*, *'"Fighting with Buffalo Bill"'*>, <*'Ray Taylor'*, *'directed'*, *'159 films'*>, <*'Ray Taylor'*, *'lived from'*, *'1888-12-01 to 1952-02-15'*>, <*'Ray Taylor'*, *'was a'*, *'American film director'*> ...
Refinement Actions		
<refinement>insert_edge(*'Modern Husbands'*, *'directed-by'*, *'Luis Bayón Herrera'*)|insert_edge(*'The Fighting Vigilantes'*, *'directed-by'*, *'Ray Taylor'*)|insert_edge(*'Modern Husbands'*, *'released-on'*, *'1948-01-01'*)|insert_edge(*'The Fighting Vigilantes'*, *'released-on'*, *'1947-11-15'*)|replace_node(*'Ray Taylor'*, *'Ray Taylor (1888-12-01 to 1952-02-15)'*)|</refinement>		

Table 9: The case of refinement for disambiguation issue.

G Rationality of the Action Space Design

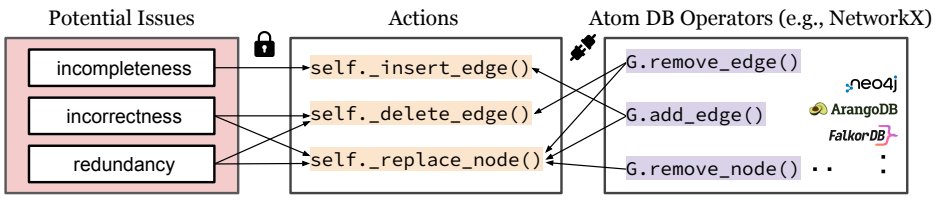

Figure 8: An overview about how the actions are corresponding to the potential issues in the KB and how the actions are executed with atom graph database operators in flexible database backends.

We demonstrate how the three predefined types of actions are corresponding to the three aspects of potential issues in the KB and how the actions can be executed with atom graph database operators in flexible database backends in Figure 8. To be specific, from the functionality perspective, the edge

insertion action complements missing relations or knowledge items to address incompleteness. Edge deletion action removes incorrect or redundant relations to mitigate errors and redundancy. And node replacement action resolves ambiguity to further reduce redundancy. Other actions are not included as they can either be implemented through the three pre-defined operators or are unnecessary (e.g., inserting an isolated node) for the knowledge refinement task.

From the flexibility perspective, although the action space is fixed with the schema, they can be executed by different graph database backends via flexible mappings. The predefined actions are just interfaces between natural language sentences generated by the LLM and the common atom operators defined in various graph databases like NetworkX (Hagberg & Conway, 2020), Neo4j, ArangoDB, and so on. For example, the node replacement action can be implemented through an integration of the original node deletion, original node related edge deletion, new node insertion and the edge insertion of the new node. Consequently, the fixed action space would not affect the generalizability across different backend graph database systems.

H Training System Overview of DeepRefine

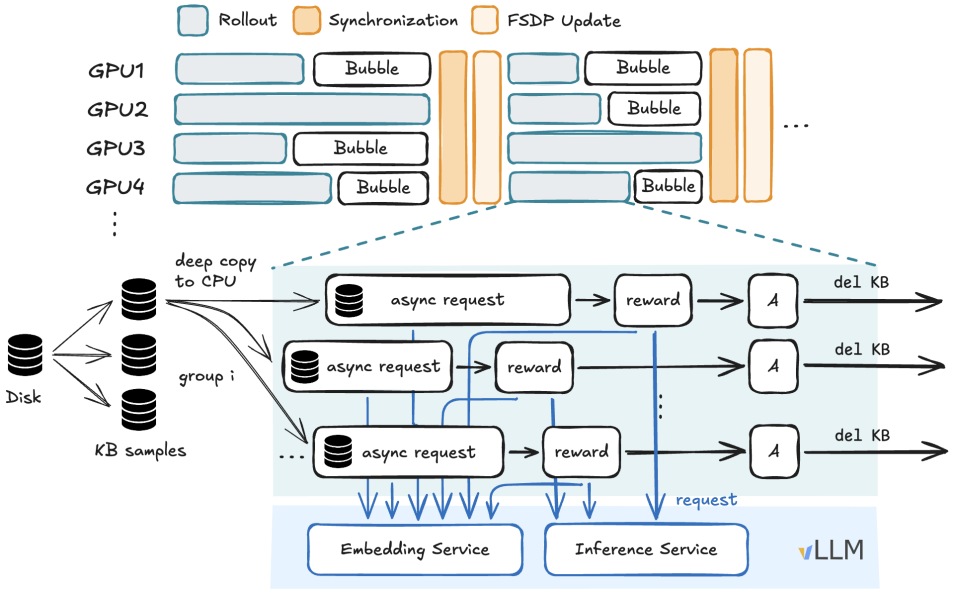

Figure 9: An overview about the training system of DeepRefine.

To further understand the training workflow of DeepRefine and improve the reproductivity, we provide a detailed overview of the training system of DeepRefine in this section.

To be specific, as shown in the upper part of Figure 9, we adopt the verl (Sheng et al., 2024) framework to conduct the multi-turn RL training for its better systematic performance. In this figure, we take 4 GPUs and a group of samples as an example. Generally, the actor will update its policy when the slowest GPU is finished with the current batch of samples, the status of the other GPUs are all idle, and the synchronization across GPUs has also been finished.

Within each rollout in each GPU device, directly uploading a large knowledge base from the disk to the CPU memory and then make copies would lead to system crash. Because different rollouts should be conducted in isolated knowledge bases. So here we choose to use the small knowledge bases constructed from the local context of each query, which not only contains the query-related documents, but also the irrelevant documents. These small KB samples will first be sampled from disk and copy G times to the CPU memory, which introduces much lower memory overhead. After each coroutine of the rollout and the reward computations are finished, the KBs will be remved from the CPU memory.

I PROMPT TEMPLATES

In this section, we will provide the prompt templates for the DeepRefine model. We demonstrate the prompt templates for each reasoning step in DeepRefine in Figure 10, Figure 11, and Figure 12.

Prompt Template for Answerable Judgement.

System:
As an advanced judgement assistant, your task is to judge whether the given question is answerable based on the provided KG context.
Evaluate whether the given question is answerable based on the provided KG context. Output your judgement in the following format:
<judge>Yes</judge> or <judge>No</judge>
Important: You must think carefully about the question and the KG context before making your judgement. And output your judgement result directly in the specified format.

User (first-turn):
Question: **{question}**
Knowledge Graph (KG) context: **{triples_string}**

User (multi-turn):
Additional KG context: **{triples_string}**
Using the original question and all KG context in this conversation, judge answerability again.

Output:

```
<judge>Yes</judge> or <judge>No</judge>
```

Figure 10: The prompt template for answerable judgement in DeepRefine.

Prompt Template for Error Abduction.

System:
As an advanced error abduction assistant, your task is to analyze the error reasons based on the given interaction history.
Analyze the reasons of the unanswerable questions based on the given interaction history. Output your analysis in the following format:
<abduction>...</abduction>
Important: You must think carefully about the interaction history before making your analysis. And output your analysis result directly in the specified format.

User:
The KG expansion stage is complete. Using the original question, all KG context, and judgement results already present in this conversation, analyze why the question was initially unanswerable.

Output:

```
<abduction>...</abduction>
```

Figure 11: The prompt template for error abduction in DeepRefine.

Prompt Template for Refinement Action Generation.

System:
As an advanced knowledge graph refinement assistant, your task is to generate a series of actions to refine the given KG to make it more suitable for answering the given question. Based on the given KG and the analysed error reasons, refine the given KG to make it more easily for retrieval and answering the given question. You have the following three types of actions to conduct:
- insert_edge(subject, relation, object): Insert a new edge into the KG to complete the missing information.
- delete_edge(subject, relation, object): Delete an edge from the KG to remove the redundant information or conflicting information.
- replace_node(old_entity, new_entity): Replace an entity in the KG to correct the errors or deal with disambiguation.
Output a series of actions in the following format:
<refinement>insert_edge("...", "...", "...")|delete_edge("...", "...", "...")|replace_node("...", "...")|...</refinement>
Important: You must think carefully about the given KG and the analysed error reasons before making your refinement. DO NOT DELETE ANY IRRELEVANT TRIPLES FROM THE ORIGINAL KG. TRY TO KEEP THE ORIGINAL KG AS MUCH AS POSSIBLE. And output your refinement result directly in the specified format.

User:
Given the Original Text: **{original_text}**,
Using the original question, accumulated KG context, and the abduction immediately above, generate the graph refinement actions.

Output:

```
<refinement>insert\_edge("...", "...", "...")|delete\_edge("...", "...",
"...")|replace\_node("...", "...")|...</refinement>
```

Figure 12: The prompt template for refinement action generation in DeepRefine.

J LIMITATIONS

In this work, we propose an RL framework for agentic knowledge base refinement. Although DeepRefine has shown promising performance in the experiments, there are still some potential improvements that can be made to the model. For example, we only construct the training data from HotpotQA with easy filtering criteria, which makes the quality of the training data not high enough. Finding out higher quality and more diverse training data is a promising direction for future work. And how to combine DeepRefine with a robust systematic refinement framework to make the performance better and more stable is also underexplored.