Joint Knowledge Base Completion and Question Answering by Combining Large Language Models and Small Language Models

Yinan Liu[1*†], **Dongying Lin**[1*], **Sigang Luo**[1], **Xiaochun Yang**[1,2], **Bin Wang**[1,2]
[1]School of Computer Science and Engineering, Northeastern University, Shenyang, China
[2]National Frontiers Science Center for Industrial Intelligence and Systems optimization, Northeastern University, Shenyang, China
liuyinan@cse.neu.edu.cn, 2472128@stu.neu.edu.cn,
2301925@stu.neu.edu.cn, {yangxc, binwang}@mail.neu.edu.cn

Abstract

Knowledge Bases (KBs) play a key role in various applications. As two representative KB-related tasks, knowledge base completion (KBC) and knowledge base question answering (KBQA) are closely related and inherently complementary with each other. Thus, it will be beneficial to solve the task of joint KBC and KBQA to make them reinforce each other. However, existing studies usually rely on the small language model (SLM) to enhance them jointly, and the large language model (LLM)'s strong reasoning ability is ignored. In this paper, by combining the strengths of the LLM with the SLM, we propose a novel framework JCQL, which can make these two tasks enhance each other in an iterative manner. To make KBC enhance KBQA, we augment the LLM agent-based KBQA model's reasoning paths by incorporating an SLM-trained KBC model as an action of the agent, alleviating the LLM's hallucination and high computational costs issue in KBQA. To make KBQA enhance KBC, we incrementally fine-tune the KBC model by leveraging KBQA's reasoning paths as its supplementary training data, improving the ability of the SLM in KBC. Extensive experiments over two public benchmark data sets demonstrate that JCQL surpasses all baselines for both KBC and KBQA tasks.

1 Introduction

Knowledge Bases (KBs) storing enormous factual triples such as Wikidata (Vrandečić and Krötzsch, 2014), DBpedia (Lehmann et al., 2015), and Freebase (Bollacker et al., 2008), play a key role in various applications (Zhou et al., 2026). One of their main downstream applications is knowledge base question answering (KBQA), which is a significant research area that utilizes KBs to provide precise answers to natural language questions. However, these KBs are often greatly incomplete (Liu et al., 2022b, 2023, 2016, 2026). Along this line, the task of knowledge base completion (KBC), which aims to infer the missing triples for a KB, can help to improve the performance of KBQA. Meanwhile, KBQA can help KBC since the new knowledge inferred from KBQA can be used to complete the KB. It can be seen that KBC and KBQA are closely related and inherently complementary with each other (Liu et al., 2022a). Thus, it will be beneficial to solve KBC and KBQA jointly to make them reinforce each other.

To our best knowledge, only two existing studies have attempted to integrate KBC and KBQA tasks by leveraging the small language models (SLMs) such as BERT (Devlin et al., 2019) and T5 (Raffel et al., 2020). The first approach BiNet (Liu et al., 2022a) proposes a multi-task learning framework that shares an embedding space to facilitate latent feature exchange between these two tasks. However, this method requires manual annotation of reasoning paths (i.e., relation sequences of questions (Liu et al., 2022a)) in the embedding space to construct training data, which is time-consuming and labor-intensive. The second work, KGT5 (Saxena et al., 2022) formulates KBC and KBQA as seq2seq tasks. Specifically, KGT5 first pre-trains T5 for KBC and then fine-tunes it for KBQA, enabling knowledge integration between tasks through shared parameters. While this method is simple and intuitive, it overlooks the potential enhancing effect of KBQA on KBC. Notably, it is worth mentioning that the previous two studies are both based on SLMs, failing to use the abilities of large language models (LLMs) such as GPT-4 (Achiam et al., 2023) and LLaMA (Touvron et al., 2023). These LLMs exhibit superior reasoning abilities that can retrieve relevant triples from the KB to generate the needed reasoning paths through exploration and reasoning (Sun et al., 2024; Luo et al., 2025). A heuristic way to integrate KBC

[*] Equal contribution.
[†] Corresponding authors.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 36756–36775

and KBQA tasks employing the LLM is establishing an iterative workflow, e.g., first executing an LLM-based KBC method to complete the KB, then running an LLM-based KBQA method, and subsequently leveraging newly acquired knowledge from KBQA's reasoning paths to populate the KB, which can be used as the input of the next round of KBC. However, this workflow suffers from two limitations: (1) the cost overhead from frequent LLM calls; (2) results provided by LLMs are not precise due to the LLM's hallucination issue.

The above analysis reveals fundamental limitations in using either SLMs or LLMs in isolation for solving KBC and KBQA jointly. Interestingly, we observe complementary strengths: LLMs can potentially address BiNet's manual annotation requirement through their reasoning abilities, while SLMs can help mitigate LLMs' high computational costs and hallucination issues. Therefore, an effective integration strategy to combine the strengths of LLMs and SLMs can better enable mutual enhancement between the two tasks (Fan et al., 2024).

To address the above issues, we propose a novel framework JCQL to **J**ointly solve KB**C** and KB**Q**A by combining the **L**LM and the SLM, which can integrate LLM's implicit parametric knowledge and KB's explicit structured knowledge. For KBC, JCQL trains an SLM by retrieving relevant triples from the KB for each triple as contextual information. For KBQA, for each question, JCQL extracts the LLM's reasoning paths by using the corresponding answer as the supervision to prompt the LLM to act as an agent in step-by-step reasoning. In order to integrate KBC and KBQA into a unified framework to enhance each other, JCQL employs two key mechanisms: (1) the SLM-trained KBC model is integrated as an action within the LLM agent's reasoning process, converting structured knowledge into parametric knowledge to alleviate the LLM's hallucination and high computational costs issue in KBQA (a.k.a KBC $\rightarrow$ KBQA); (2) KBQA's reasoning paths are utilized as the supplementary training data to fine-tune the KBC model based on the incremental learning technique, converting parametric knowledge into structured knowledge to improve the ability of the SLM in KBC (a.k.a KBQA $\rightarrow$ KBC). Through the above iterative process, the KBC model augmented by KBQA's reasoning paths becomes stronger, and KBQA's reasoning paths incorporating the KBC model are more likely to be high quality. Ultimately, JCQL can predict entities via the fine-tuned SLM and generate answers to questions via the LLM agent incorporating the fine-tuned SLM as an action, respectively.

Our main contributions can be summarized as follows: (1) to the best of our knowledge, this is the first framework combining the LLM and the SLM to enhance the task of joint KBC and KBQA; (2) to make KBC enhance KBQA, we augment the LLM agent by incorporating an SLM-trained KBC model as an action, which can alleviate the LLM's hallucination and high computational costs issue in KBQA; (3) to make KBQA enhance KBC, we incrementally fine-tune the KBC model by leveraging LLM-generated reasoning paths from KBQA as supplementary training data, which can improve the ability of the SLM in KBC; (4) a thorough experimental study over two public benchmark data sets under different settings shows our proposed framework significantly outperforms all baseline methods for both tasks.

2 Preliminaries and Task Definition

A KB can be represented as $G = \{E, R, F\}$, where E is the set of entities, R is the set of relations, and $F = \{\langle h, r, t\rangle | h, t \in E, r \in R\}$ is the set of triples. Given a KB, KBC aims to infer missing triples based on the existing information in the KB. Specifically, KBC consists of three categories of sub-tasks: (1) given a query $\langle h, r, ?\rangle$, complete the corresponding tail entity t; (2) given a query $\langle ?, r, t\rangle$, complete the corresponding head entity h; (3) given a query $\langle h, ?, t\rangle$, complete the relation r between h and t. KBQA aims to identify a set of entities $A_Q \subseteq E$ to answer the given questions Q based on G. Following (Liu et al., 2022a), we call the entities mentioned in questions as topic entities, denoted as $E_Q \subseteq E$. Ideally, each question $q \in Q$ can be mapped to a set of reasoning paths P_q in the KB, where $p_q = \{\langle e_1, r_1, e_2\rangle, \langle e_2, r_2, e_3\rangle, \ldots, \langle e_{d-1}, r_{d-1}, e_d\rangle\}$ is a unique reasoning path in P_q.

Definition 1 (**Joint knowledge base completion and question answering**). *Given a set of training triples for the KBC task and a set of training questions for the KBQA task, it is required to output answers to test triples for the KBC task and answers to test questions for the KBQA task.*

3 The Framework: JCQL

The overall framework JCQL is shown in Figure 1.

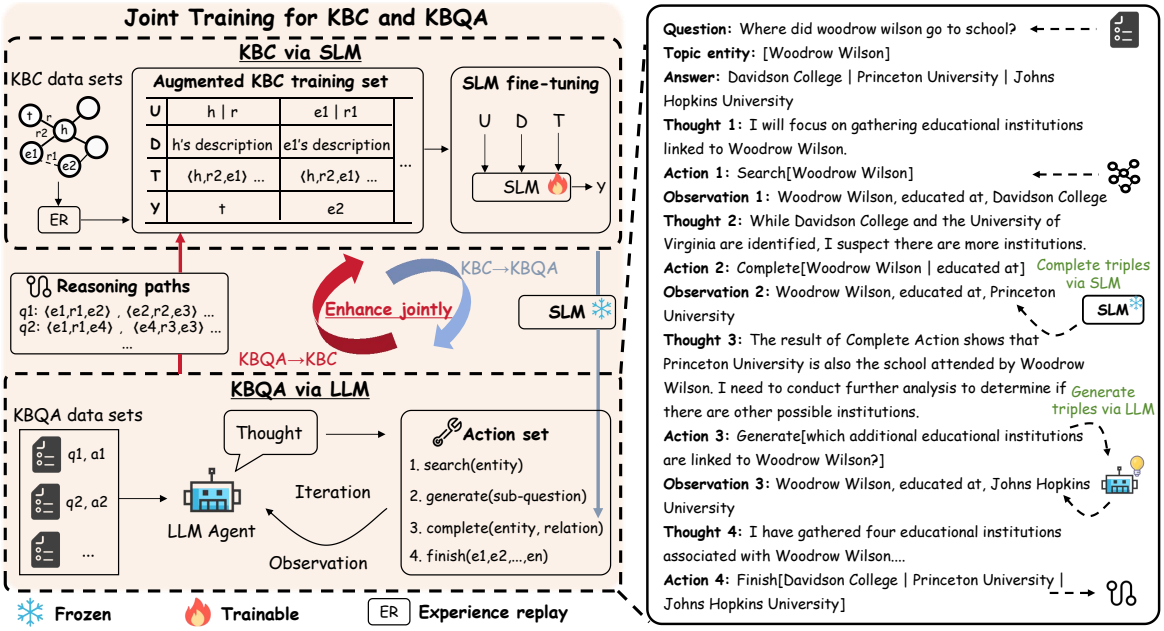


Figure 1: Overview of our framework JCQL.

3.1 KBC via SLM

Previous studies (Saxena et al., 2022; Yao et al., 2019) demonstrate that SLMs can complete missing triples by learning semantic information from KBs. Inspired by this, we train an encoder-decoder Transformer model (with a similar architecture as T5) using the cross-entropy as the loss function to solve the KBC task. Specifically, the generation process of this SLM can be formalized as follows:

$$\mathcal{P}(Y|[\mathcal{U};\mathcal{D};\mathcal{T}]) = \prod_{i=1}^{l} \mathcal{P}(y_i|y_{<i},[\mathcal{U};\mathcal{D};\mathcal{T}]), \quad (1)$$

where l is the length of sequence, $[\mathcal{U};\mathcal{D};\mathcal{T}]$ is the input sequence to the encoder of the SLM for each training triple $\langle h,r,t\rangle$, ";" is the concatenation operation between texts, $\mathcal{U}$ denotes the concatenation of h and r, and Y is the surface form of t. We obtain h's descriptive text as $\mathcal{D}$ from the KB, promoting the SLM to understand the entity better and alleviate the entity ambiguity issue. $\mathcal{T}$ is the context consisting of related triples to $\mathcal{U}$. Specifically, we rank the one-hop neighborhoods of h based on a co-occurrence score $\mathcal{S}(r,\hat{r})$ between r and neighboring relations $\hat{r}$ and subsequently select the triples with high scores to construct the context set $\mathcal{T}$. Formally, $\mathcal{S}(r,\hat{r})$ is defined as follows:

$$\mathcal{S}(r,\hat{r}) = \sum_{e\in\mathcal{E}} \mathbb{I}(r \in R_e \wedge \hat{r} \in R_e), \quad (2)$$

where R_e denotes the set of all relations incident to entity $e \in \mathcal{E}$. $\mathbb{I}(\cdot)$ denotes the indicator function and $\wedge$ denotes the logical conjunction. This score represents the frequency of r and $\hat{r}$ sharing a common entity in the KB.

3.2 KBQA via LLM

Inspired by (Sun et al., 2024), we utilize an LLM as the agent for KBQA. The LLM operates in an iterative cycle of three alternating states, i.e., `thought`, `action`, and `observation` to drive autonomous reasoning. During the reasoning process, the LLM continuously explores the KB and generates the reasoning process, dynamically integrating structured knowledge from the KB. To enable the LLM to access the KB and alleviate the KB's incomplete issue, we define the agent's action space consisting of the following three categories of actions: (1) **`search(entity)`**. This action can return the most relevant entities from the neighbors of the target entity based on some relevant relations from all triples associated with the target entity in the KB selected by the LLM (detailed in Appendix). (2) **`generate(sub-question)`**. The LLM is prompted to analyze the last `thought` to generate a sub-question. Based on the sub-question, this action retrieves related triples via BM25 algorithm (Robertson and Zaragoza, 2009) from previous observations (i.e., action execution results), and then prompts the LLM to generate new triples beyond the KB via the LLM's parametric knowledge. Yet, the LLM may frequently return unreliable triples (e.g., inverted head-tail entity pairs

and synthesized relationships unsupported by the KB). To solve this issue, JCQL prompts the LLM to correct these triples based on the description and schema of relationships (detailed in Appendix). (3) **`finish(`**$e_1, e_2, \ldots, e_n$**`)`**. This action returns an entity list as final answers, marking the completion of the reasoning process.

Like ReAct (Yao et al., 2023; Wang et al., 2026), we treat the action execution results as observations and the entire reasoning process as a sequence of actions with corresponding observations. Specifically, the agent generates a thought to analyze the current state of the reasoning process before taking action and receives the observation after taking action. Formally, the interaction trajectory at step i can be represented as:

$$\mathcal{C}_i = (q, E_q, \tau_0, \alpha_0, o_0, \ldots, \tau_{i-1}, \alpha_{i-1}, o_{i-1}), \tag{3}$$

where q denotes a question, E_q denotes q's topic entity set, $\alpha_i \in \{\texttt{search}, \texttt{generate}, \texttt{finish}\}$, o_i denotes the observation, and τ_i denotes the thought. Based on this interaction trajectory, the generating process for the subsequent thought τ_i and action α_i can be formulated as follows:

$$\mathcal{P}(\tau_i|C_i) = \prod_{j=1}^{|\tau_i|} \mathcal{P}(\tau_i^j|C_i, \tau_i^{<j}), \tag{4}$$

$$\mathcal{P}(\alpha_i|C_i, \tau_i) = \prod_{z=1}^{|\alpha_i|} \mathcal{P}(\alpha_i^z|C_i, \tau_i, \alpha_i^{<z}), \tag{5}$$

where τ_i^j and $|\tau_i|$ are the j-th token and the length of τ_i, α_i^z and $|\alpha_i|$ denote the z-th token and the length of α_i. The LLM repeats this iterative process until it obtains adequate information and outputs final answers in the form of **`finish`**.

3.3 Joint Training for KBC and KBQA

KBC → KBQA. To alleviate potential errors caused by the LLM's hallucination in KBQA, we augment the LLM agent by integrating an SLM-trained KBC model as an action within the agent's reasoning process. Specifically, in addition to the three actions `search`, `generate`, and `finish`, we add an additional action denoted as **`complete(entity, relation)`**. This action first converts the head entity and the relation into the input format of our SLM-trained KBC model and leverages this KBC model to predict entities. This mechanism creates a tight coupling between KB's explicit structured knowledge and LLM's implicit parametric knowledge to make the answer results more accurate. Although we constrain the LLM to generate the relation appearing in observations via the design of a specific prompt (detailed in Appendix), it may still generate some relations beyond the scope of the KB. To address this issue, we link the relation generated by the LLM to the most relevant one within the KB based on BM25 scores. In practice, a simple approach of determining whether to call the `complete` action is to set a threshold on the SLM's decoding probability. However, such threshold is often difficult to obtain. Therefore, we still adopt Formula 5 to calculate the probability of invoking the action `complete`.

KBQA → KBC. In order to make KB more complete to assist KBQA, we parse the reasoning paths of KBQA as the supplementary training data to fine-tune the KBC model. As shown in the right of Figure 1, given a question-answer pair, we leverage the answers as the supervision to guide the LLM to perform step-by-step reasoning, which makes the LLM agent generate more comprehensive and accurate triples by actively exploring multiple reasoning paths. Specifically, when the reasoning process of a question q is finished, we first compare predicted answers with golden answers and filter the incorrect predicted answers with the corresponding observations from the reasoning process $\mathcal{C}_q$ generated by the LLM agent. Next, we extract all triples from the reasoning process, including all observations returned from the actions `search`, `complete`, and `generate`. Then, we construct a reasoning subgraph G_q based on the extracted triples, and search for the paths through Breadth-First Search (BFS) from the topic entity to the answer entity in G_q. For each entity on the path, we use triples consisting of it and its one-hop neighborhoods to expand the paths. Ultimately, we prompt the LLM to select the triples most relevant to the question based on q and the current paths. The triples selected by the LLM are regarded as the final reasoning paths P_q of KBQA to fine-tune the SLM.

Based on the above two mechanisms, JCQL's overall training process is depicted in Algorithm 1. First, in $\textsc{Pretrain}(\{\langle h, r, t\rangle\}, \mathcal{M}_S)$, JCQL pre-trains a KBC model via the SLM $\mathcal{M}_S$ based on the KBC training set $\{\langle h, r, t\rangle\}$ (introduced in Section 3.1). Next, in $\textsc{Agent}(q, a_q, e_q, \mathcal{M}_S)$, given a question-answer pair from KBQA's training set, the LLM agent incorporates the pre-trained $\mathcal{M}_S$ to perform reasoning step-by-step to obtain the interaction trajectory C_q. Sub-

sequently, in PARSE$(C_q, \mathcal{M}_L)$, JCQL leverages the LLM $\mathcal{M}_L$ to extract the reasoning paths P_q based on C_q (introduced in Section 3.3). In INCFINETUNE$(\{\langle h, r, t\rangle\}, P_q, \mathcal{M}_S)$, the triples of P_q are utilized to fine-tune $\mathcal{M}_S$ in real time. Note that each reasoning path contains only a small number of triples, which include those in the KBC's training set used for pre-training (i.e., triples returned from the action `search`) and supplementary triples provided by $\mathcal{M}_L$ (i.e., triples returned from actions `complete` and `generate`). To mitigate the issue of catastrophic forgetting, we employ a simple incremental learning technique known as experience replay (Mnih et al., 2013) to combine triples in reasoning paths with few triples drawn uniformly at random from replay memory and apply mini-batch updates to all these samples to fine-tune $\mathcal{M}_S$. In practice, we store all triples of the KBC's training set in the replay memory. To reduce training time, we sample only ten triples for each triple in reasoning paths. Thus, by fine-tuning $\mathcal{M}_S$ continuously based on the incremental learning technique, KBC and KBQA can mutually influence each other, in the sense that the KBC model augmented by KBQA's reasoning paths becomes stronger, and the reasoning paths of KBQA incorporating the KBC model are more likely to be high quality. To further enhance performance, we can use more advanced incremental learning methods (Jia et al., 2025), but this is not the focus of our paper.

3.4 Inference

Given the query $\langle h, r, ?\rangle$ from the KBC's test set and the fine-tuned SLM, JCQL first converts it into the input sequence of the SLM (introduced in Section 3.1). Then, JCQL samples R sequences from SLM's decoder. Based on the pre-constructed mapping dictionary (e.g., {Justin Bieber: Q34086}), the output sequences that do not contain KB entities are filtered. Thus, for each remaining output sequence, we assign a score to the corresponding entity ID based on the probability of decoding the sequence. Simultaneously, we assign a negative infinity score for each entity ID that is not in the output sequences. In this way, the top k predicted entities can be returned. Given KBQA's test questions and the fine-tuned SLM, JCQL obtains the answers via the LLM agent without gold answers as the supervision, which is the only difference from the training phase. The results of the action `finish`$(e_1, e_2, \ldots, e_n)$ will serve as final answers. Prompts used here can be found in Appendix.

Formally, the calls to the LLM for the whole inference process are $|Q| \cdot L \cdot N$, where $|Q|$ denotes the number of questions, L denotes the maximum number of `thought` for each question, and N denotes the maximum number of `action` calls to the LLM for each question. JCQL first calls the LLM during the action `search` execution phase to select relevant relations. The second call may occurs during the action `generate` execution phase, which can generate and correct new triples through the LLM. Thus the value of N is 2. To reduce the inference time, L is set to 10.

Algorithm 1 JCQL

Input: The KB G, KBQA's training set $\{Q, A_Q, E_Q\}$, KBC's training set $\{\langle h, r, t\rangle \in G\}$, an SLM $\mathcal{M}_S$, an LLM $\mathcal{M}_L$
Output: The fine-tuned $\mathcal{M}_S$

$\mathcal{M}_S \leftarrow$ PRETRAIN$(\{\langle h, r, t\rangle\}, \mathcal{M}_S)$
for $\{q, a_q, e_q\} \in \{Q, A_Q, E_Q\}$ **do**
  // KBC → KBQA
  Trajectory $C_q \leftarrow$ AGENT$(q, a_q, e_q, \mathcal{M}_S)$
  // KBQA → KBC
  Paths $P_q \leftarrow$ PARSE$(C_q, \mathcal{M}_L)$
  $\mathcal{M}_S \leftarrow$ INCFINETUNE$(\{\langle h, r, t\rangle\}, P_q, \mathcal{M}_S)$
end for

4 Experimental Study

4.1 Experiment Settings

Data Sets. In the experiments, we utilize two common benchmark data sets, WebQuestionSP (WebQSP) and Complex WebQuestion (CWQ). WebQSP has $4,000$ questions, which consist of both one-hop and two-hop questions. CWQ contains about $30,000$ questions, which include questions requiring composite reasoning and superlative reasoning. Following ToG (Sun et al., 2024), we use Wikidata as the background KB. To reduce cost in evaluating our joint task, following (Liu et al., 2022a), we restrict the KB used in this paper to a subset of Wikidata for the evaluation of all methods by encompassing all triples within two hops of any entity mentioned in WebQSP/CWQ. This scope is small yet enough, covering over 99% of question answers of WebQSP/CWQ. Thus, we can obtain the corresponding background KB for WebQSP and CWQ respectively. For each KB, we randomly sample $20,000$ triples to construct the validation set, $20,000$ triples to construct the test set, and the remaining triples to construct the training set. Following (Liu et al., 2022a), for each training set, we randomly delete 50% and 70% edges as two challenge settings of our work, i.e., 50% KB and 30% KB respectively. The statistics of these data

Method	*WebQSP (30% KB)*				*WebQSP (50% KB)*				*CWQ (30% KB)*				*CWQ (50% KB)*			
	MRR	*Hits@1*	*Hits@3*	*Hits@10*	*MRR*	*Hits@1*	*Hits@3*	*Hits@10*	*MRR*	*Hits@1*	*Hits@3*	*Hits@10*	*MRR*	*Hits@1*	*Hits@3*	*Hits@10*
TransE (NeurIPS'13)	0.032	0.000	0.043	0.090	0.051	0.011	0.068	0.130	0.047	0.011	0.062	0.121	0.052	0.012	0.067	0.133
DistMult (ICLR'15)	0.112	0.079	0.125	0.176	0.166	0.118	0.182	0.260	0.100	0.070	0.111	0.161	0.153	0.107	0.169	0.243
ComplEx (ICML'16)	0.119	0.085	0.132	0.183	0.178	0.127	0.200	0.277	0.106	0.074	0.119	0.166	0.158	0.110	0.177	0.252
RotatE (ICLR'19)	0.055	0.023	0.062	0.124	0.055	0.024	0.062	0.125	0.058	0.027	0.064	0.125	0.047	0.024	0.061	0.125
SimKGC (ACL'22)	0.456	0.377	0.493	0.605	0.480	0.400	0.518	0.632	0.451	0.360	0.493	0.625	0.466	0.375	0.510	0.642
KGT5 (ACL'22)	0.517	0.484	0.536	0.580	0.543	0.509	0.565	0.610	0.503	0.467	0.524	0.569	0.536	0.501	0.559	0.602
BiNet (SIGKDD'22)	0.152	0.123	0.177	0.181	0.166	0.138	0.195	0.195	0.158	0.113	0.170	0.178	0.178	0.139	0.210	0.210
CSProm-KG (ACL'23)	0.416	0.360	0.448	0.518	0.432	0.375	0.466	0.538	0.400	0.343	0.433	0.506	0.421	0.363	0.454	0.529
DIFT (ISWC'24)	0.515	0.457	0.540	0.625	0.536	0.474	0.563	0.649	0.504	0.438	0.530	0.634	0.521	0.454	0.547	0.651
IVR (NeurIPS'24)	0.266	0.201	0.299	0.385	0.343	0.270	0.382	0.479	0.255	0.188	0.290	0.379	0.337	0.264	0.378	0.472
GoldE (ICML'24)	0.162	0.116	0.182	0.251	0.207	0.152	0.232	0.309	0.163	0.116	0.184	0.252	0.200	0.145	0.226	0.302
RAA-KGC (AAAI'25)	0.318	0.245	0.342	0.448	0.321	0.251	0.349	0.457	0.329	0.261	0.355	0.458	0.330	0.260	0.358	0.466
SATKGC (WWW'25)	0.491	0.409	0.531	0.645	0.510	0.427	0.554	0.669	0.476	0.394	0.515	0.630	0.503	0.419	0.546	0.661
JCQL$_{BART-small}$	0.585	0.553	0.609	0.640	0.609	0.577	0.633	0.664	0.559	0.526	0.583	0.616	0.612	0.581	**0.636**	0.665
JCQL$_{Flan-T5-small}$	0.590	0.559	0.612	0.646	0.621	0.591	0.645	0.675	0.583	0.552	0.606	0.637	0.610	0.578	0.633	0.664
JCQL$_{T5-small}$	**0.603**	**0.572**	**0.628**	**0.652**	**0.630**	**0.599**	**0.652**	**0.693**	**0.588**	**0.556**	**0.612**	**0.645**	**0.614**	**0.583**	0.634	**0.670**

Table 1: Performance on the KBC task. The best results are indicated in bold. The second best results are underlined.

sets are available in Appendix. We make the data sets and source code used in this paper publicly available for future research[1].

Evaluation Measures. For KBC, we adopt the evaluation measures mean reciprocal rank (MRR) and Hits@i ($i \in \{1, 3, 10\}$), which are the same as KBC studies (Bordes et al., 2013; Trouillon et al., 2016). For KBQA, following (Sun et al., 2024; Xu et al., 2024), we adopt the same evaluation measure Hits@1. Please refer to Appendix B for more details.

Setting Details. In Section 4.2, we fix GPT-4o-mini as the LLM and show the performance of JCQL with T5-small, Bart-small (Lewis et al., 2020), and Flan-T5-small (Chung et al., 2022) as the SLMs. In Section 4.3, we fix T5-small as the SLM and show the performance of JCQL with Qwen3-30B-A3B (Zhang et al., 2025), LLaMA3.1-70B, and GPT-4o-mini as the LLMs. We adopt GPT-4o-mini as the LLM and T5-small as the SLM in other experiments. The maximum size of $\mathcal{T}$, The maximum token length, LLM's temperature parameter, sampling size, and SLM's temperature are set to 20, 256, 0.7, 500, and 1 respectively. For the KBQA task, the number of triples returned by the SLM k is set to 5. JCQL is optimized by using Adafactor with a batch size of 64. For the learning rate, we use the relative learning rate with the warmup initialization method. These hyperparameters are applied uniformly across all data sets. Inspired by (Sun et al., 2024), we use ReFinED (Ayoola et al., 2022) to perform entity linking for all questions. It is guaranteed that there is at least a path between topic entities and answer entities, which is the same as the previous joint KBC and KBQA study (Liu et al., 2022a). For questions without topic entities, JCQL answers via CoT (Wei et al., 2022), which is the same setting as the previous KBQA study (Sun et al., 2024).

[1]https://github.com/ldp2211479/JCQL

Method	*WebQSP*		*CWQ*	
w/o KB				
GPT-4o-mini	0.677		0.421	
CoT (NeurIPS'22)	0.646		0.446	
SC (ICLR'23)	0.643		0.464	
w/ KB				
	30% KB	*50% KB*	*30% KB*	*50% KB*
DECAF$_{Ans}$ (ICLR'23)	0.419	0.487	0.295	0.312
KGT5 (ACL'22)	0.201	0.208	0.248	0.261
BiNet (SIGKDD'22)	0.166	0.177	0.183	0.208
KG-CoT (IJCAI'24)	0.230	0.265	0.097	0.165
ToG (ICLR'24)	0.457	0.706	0.360	0.328
GoG (EMNLP'24)	0.684	0.693	0.476	0.477
PoG (NeurIPS'24)	0.668	0.688	0.400	0.423
LMP (ACL'25)	0.716	0.725	0.381	0.401
PDRR (AAAI'26)	0.687	0.690	0.469	0.471
JCQL$_{Qwen3-30B-A3B}$	0.668	0.676	0.422	0.427
JCQL$_{LLaMA3.1-70B}$	0.697	0.705	0.460	0.469
JCQL$_{GPT-4o-mini}$	**0.730**	**0.738**	**0.502**	**0.508**

Table 2: Performance on the KBQA task in terms of Hits@1. The best results are indicated in bold. The second best results are underlined.

4.2 Knowledge Base Completion

For this task, we compare JCQL with 13 SOTA methods including TransE (Bordes et al., 2013), DistMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016), RotatE (Sun et al., 2019), IVR (Xiao and Cao, 2024), GoldE (Li et al., 2024a), SimKGC (Wang et al., 2022), CSProm-KG (Chen et al., 2023), DIFT (Liu et al., 2024), RAA-KGC (Yuan et al., 2025), SATKGC (Ko et al., 2025), Bi-

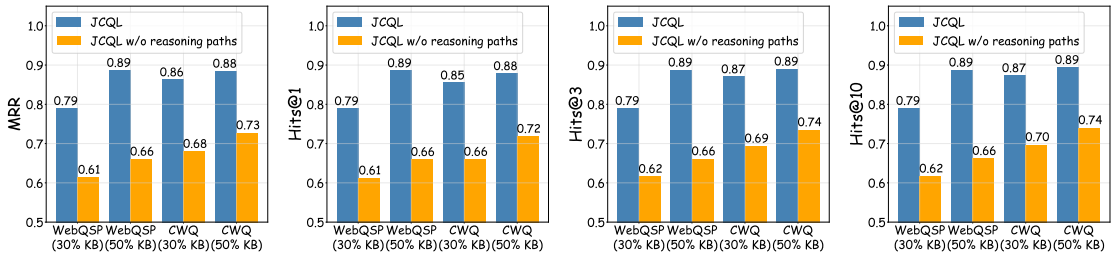


Figure 2: Ablation study on the LLM's reasoning paths for the task of KBC.

Method	*WebQSP*		*CWQ*	
	30% KB	*50% KB*	*30% KB*	*50% KB*
JCQL	**0.730**	**0.738**	**0.502**	**0.508**
JCQL w/o `generate`	0.706	0.722	0.482	0.490
JCQL w/o `complete`	0.691	0.696	0.473	0.475

Table 3: Ablation study on the SLM for the task of KBQA in terms of Hits@1.

Net (Liu et al., 2022a), and KGT5 (Saxena et al., 2022) on predicting tail entities. Their descriptions are presented in Appendix. Each baseline's performance is reproduced via its open-source solution under its default parameters. From the results on both data sets under different settings (i.e., 50% KB and 30% KB) in Table 1, it can be seen that JCQL using different SLMs achieves the best performance compared with 13 baselines in terms of all evaluation measures, demonstrating the effectiveness of JCQL for the KBC task.

4.3 Knowledge Base Question Answering

For this task, in addition to BiNet and KGT5, we use other baselines including GPT-4o-mini, Chain-of-Thought (CoT) (Wei et al., 2022), Self-Consistency (SC) (Wang et al., 2023), KG-CoT (Zhao et al., 2024), ToG (Sun et al., 2024), PoG (Chen et al., 2024), LMP (Wan et al., 2025), PDRR (Zhu et al., 2025), GoG (Xu et al., 2024) and a variant DecAF$_{Ans}$ (i.e., DecAF only using generated answers) of DecAF that does not need SPARQL queries (Yu et al., 2023). The descriptions of them are presented in Appendix. For each baseline, the performance is reproduced via its open-source solution under its default parameters. Note that for all baselines except BiNet and KGT5, we adopt GPT-4o-mini as the LLM. From the results on both data sets under different settings (i.e., 50% KB and 30% KB) in Table 2, it can be seen that JCQL$_{GPT-4o-mini}$ achieves the best performance, outperforming all baselines that also employ GPT-4o-mini. Additionally, the variants based on open-source LLMs (i.e., JCQL$_{LLaMA3.1-70B}$ and JCQL$_{Qwen3-30B-A3B}$) also exhibit highly competitive results.

4.4 Ablation Study

Effect Analysis of the LLM's Reasoning Paths in KBC. To validate the effectiveness of KBQA in enhancing KBC for JCQL, we conduct a statistical analysis of KBC's test triples and KBQA's reasoning paths in Appendix. The analysis results show that the intersection of triples involved in the reasoning paths of the KBQA task and the KBC's test triples is relatively small, which may limit the enhancement of KBQA to KBC. Based on this observation, we first adopt the BFS method to identify the triples that form the shortest path for each KBQA's training question over the complete KB. Then, according to the generated triples of all training questions, we filter the triples that belong to the corresponding KBC training set and use the remaining triples to test the performance of JCQL without reasoning paths. As shown in Figure 2(a)-(d), it can be seen that without reasoning paths, the performance of JCQL declines by at least 8 percentages on both data sets under different settings for the KBC task. Overall, the LLM can improve the SLM's reasoning ability in KBC, especially for KBC test triples existing in the reasoning paths of KBQA training questions.

Effect Analysis of the SLM in KBQA. To verify the importance of the SLM in KBQA, we remove `complete` and `generate` from the LLM agent's action set, respectively. `Generate` prompts the LLM to generate missing triples, while `complete` generates triples with a fine-tuned SLM. As shown in Table 3, we can see that the absence of `complete` causes a more substantial performance decline in JCQL than the absence of `generate`, which vali-

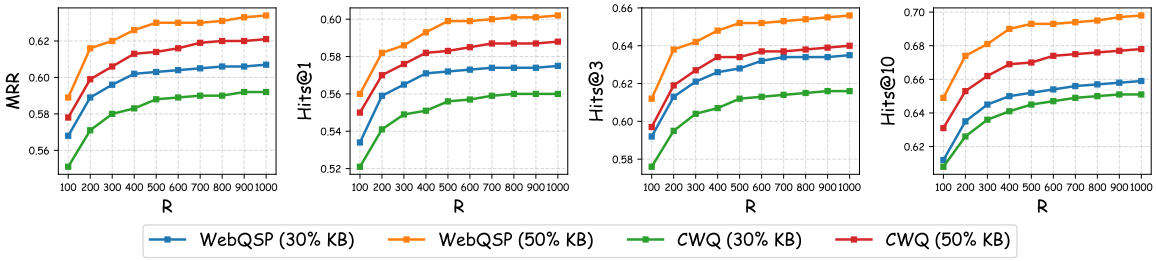


Figure 3: Parameter study over WebQSP and CWQ under different settings.

Data set	*Method*	*30% KB*		*50% KB*	
		LLM call	*Time (s)*	*LLM call*	*Time (s)*
CWQ	ToG	18.5	33.7	18.2	32.7
	GoG	7.9	26.3	7.7	26.1
	PoG	11.7	42.6	9.7	42.8
	LMP	11.4	59.4	11.3	58.8
	PDRR	8.6	49.5	7.6	45.8
	JCQL-SLM	8.2	28.4	8.0	27.2
	JCQL	**7.3**	**25.3**	**7.2**	**20.9**
WebQSP	ToG	17.5	26.6	15.2	23.9
	GoG	5.9	16.8	6.0	16.2
	PoG	6.8	31.0	5.4	26.2
	LMP	5.4	22.1	5.5	21.3
	PDRR	7.5	33.1	7.6	31.6
	JCQL-SLM	5.8	17.4	5.8	17.2
	JCQL	**5.1**	**16.6**	**5.2**	**15.9**

Table 4: Efficiency study on the task of KBQA.

dates the point that the SLM can indeed alleviate the hallucination to enhance the performance by providing much more accurate tail entities to generate more reliable reasoning paths.

4.5 Efficiency Study

We study the efficiency of JCQL, JCQL-SLM (i.e., JCQL w/o `complete`), and some SOTA baselines (i.e., ToG, PoG, LMP, PDRR, and GoG). As shown in Table 4, JCQL achieves the best performance in terms of $LLM\ call$ (i.e., the average number of LLM calls for each question) and $Time$ (i.e., inference time) on both data sets under different settings. Furthermore, JCQL outperforms JCQL-SLM, demonstrating the effectiveness of the SLM for alleviating the LLM's high computational costs issue.

4.6 Parameter Study

In order to comprehensively understand the performance characteristics of JCQL, we conduct the sensitivity analysis to assess how parameter R (the number of sequences from the decoder of the SLM) influences JCQL's performance on the task of KBC. Figure 3 shows the performance of JCQL with parameter $R = \{100, 200, 300, ..., 1000\}$ on WebQSP and CWQ. From the trend shown in Figure 3, we can see that when the number of sequences increases, the performance improves rapidly at lower values, then grows slowly at high values until it stabilizes on both data sets.

5 Related Work

KBC methods often contain two categories of studies: (1) structure-based methods (Bordes et al., 2013; Yang et al., 2015; Trouillon et al., 2016; Sun et al., 2019; Xiao and Cao, 2024; Li et al., 2024a); (2) semantic-based methods (Wang et al., 2022; Chen et al., 2023; Saxena et al., 2022; Liu et al., 2022a, 2024). Note that all above methods often rely on offline training with static KBs, resulting in limited scalability, while JCQL uses LLM's reasoning paths to fine-tune an SLM to update the KB dynamically with the incremental learning technique.

KBQA receives a lot of attention recently. Some studies adopt information retrieval-based methods (Li et al., 2024b; Zhao et al., 2024). ToG (Sun et al., 2024) and PoG (Chen et al., 2024) attempt to treat the LLM as an agent to interactively explore relation paths step by step in the KB and perform reasoning based on the retrieved paths. GoG (Xu et al., 2024) uses the LLM's internal knowledge and the external knowledge in the KB to reason. However, when relations are lacking, GoG often relies on the LLM's internal knowledge, potentially leading to hallucination. In contrast, JCQL embeds a fine-tuned SLM-based KBC model as an action into the LLM agent to generate missing triples, effectively mitigating hallucination issues.

Joint KBC and KBQA methods receive limited attention so far. BiNet (Liu et al., 2022a) converts the question into a relationship path by BERT and jointly deals with KBC and KBQA through a shared embedding space and an answer scoring module. KGT5 (Saxena et al., 2022) converts

triples into a simple input sequence to pre-train T5 for KBC, and fine-tunes it using question and answer pairs for KBQA. Both methods ignore LLM's reasoning ability, while JCQL combines the LLM and the SLM to integrate the knowledge from the LLM and the KB to solve these two tasks jointly.

6 Conclusion

In this paper, we propose a novel framework JCQL, which can resolve KBC and KBQA simultaneously and make the two tasks reinforce each other by combining strengths of the LLM and the SLM. In this framework, LLM's parametric knowledge and KB's structured knowledge can be integrated in an iterative manner. Extensive experiments over two public benchmark data sets have demonstrated the effectiveness of JCQL against many SOTA KBC and KBQA methods.

Limitations

Limitations of JCQL are listed as follows. (1) For KBC, the SLM's context is constructed from the one-hop neighborhoods of the head entity. For some challenging queries, this simple context may not provide sufficient information. (2) For KBQA, JCQL integrates LLM's parametric knowledge and KB's structured knowledge. In future work, we will explore more useful information from the search engine and the integration of multiple KBs to provide more actions for the LLM agent. (3) The current version of JCQL employs a simple and efficient experience replay strategy. We intend to explore more sophisticated incremental learning techniques to further enhance performance.

Acknowledgments

The work was partially supported by the National Key Research and Development Program of China (No. 2024YFF0617702); the National Natural Science Foundation of China (Nos. U22A2025, 62402097, 62232007, and U23A20309); the Joint Funds of the Natural Science Foundation of Liaoning Province (No. 2023-BSBA-132); the 111 Project (No. B16009); the Ant Group Research Program (No. 2025021900003); and the Fundamental Research Funds for the Central Universities (No. N2417007).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Tom Ayoola, Shubhi Tyagi, Joseph Fisher, Christos Christodoulopoulos, and Andrea Pierleoni. 2022. ReFinED: An efficient zero-shot-capable approach to end-to-end entity linking. In *NAACL*, pages 209–220.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *NeurIPS*, pages 2787–2795.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *NeurIPS*, pages 1877–1901.

Chen Chen, Yufei Wang, Aixin Sun, Bing Li, and Kwok-Yan Lam. 2023. Dipping PLMs sauce: Bridging structure and text for effective knowledge graph completion via conditional soft prompting. In *ACL*, pages 11489–11503.

Liyi Chen, Panrong Tong, Zhongming Jin, Ying Sun, Jieping Ye, and Hui Xiong. 2024. Plan-on-graph: Self-correcting adaptive planning of large language model on knowledge graphs. In *NeurIPS*, pages 37665–37691.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, and 16 others. 2022. Scaling instruction-finetuned language models. *Preprint*, arXiv:2210.11416.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186.

Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining small language models and large language models for zero-shot NL2SQL. *Proceedings of the VLDB Endowment*, 17(11):2750–2763.

Zhifeng Jia, Hanmo Liu, Haoyang Li, and Lei Chen. 2025. SIT: Selective Incremental Training for Dynamic Knowledge Graph Embedding . In *ICDE*, pages 1607–1621.

Youmin Ko, Hyemin Yang, Taeuk Kim, and Hyunjoon Kim. 2025. Subgraph-aware training of language models for knowledge graph completion using structure-aware contrastive learning. In *WWW*, page 72–85.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and 1 others. 2015. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*, pages 7871–7880.

Rui Li, Chaozhuo Li, Yanming Shen, Zeyu Zhang, and Xu Chen. 2024a. Generalizing knowledge graph embedding with universal orthogonal parameterization. In *ICML*, pages 28040–28059.

Xingxuan Li, Ruochen Zhao, Yew Ken Chia, Bosheng Ding, Shafiq Joty, Soujanya Poria, and Lidong Bing. 2024b. Chain-of-knowledge: Grounding large language models via dynamic knowledge adapting over heterogeneous sources. In *ICLR*, pages 1–23.

Lihui Liu, Boxin Du, Jiejun Xu, Yinglong Xia, and Hanghang Tong. 2022a. Joint knowledge graph completion and question answering. In *SIGKDD*, pages 1098–1108.

Yang Liu, Xiaobin Tian, Zequn Sun, and Wei Hu. 2024. Finetuning generative large language models with discrimination instructions for knowledge graph completion. In *ISWC*, pages 199–217.

Yinan Liu, Hu Chen, and Wei Shen. 2022b. Personal attribute prediction from conversations. In *WWW*, pages 223–227.

Yinan Liu, Hu Chen, Wei Shen, and Jiaoyan Chen. 2023. Low-resource personal attribute prediction from conversations. In *AAAI*, pages 4507–4515.

Yinan Liu, Wei Shen, and Xiaojie Yuan. 2016. Deola: A system for linking author entities in web document with DBLP. In *CIKM*, pages 2449–2452. ACM.

Yinan Liu, Ziyang Zhang, Bin Wang, and Xiaochun Yang. 2026. SEFEL: A simple yet effective framework for fast event linking. In *AAAI*, pages 15377–15385.

Sigang Luo, Yinan Liu, Dongying Lin, Yingying Zhai, Bin Wang, Xiaochun Yang, and Junpeng Liu. 2025. ETRQA: A comprehensive benchmark for evaluating event temporal reasoning abilities of large language models. In *ACL*, pages 23321–23339.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *Preprint*, arXiv:1312.5602.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21:140:1–140:67.

Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. 2022. Sequence-to-sequence knowledge graph completion and question answering. In *ACL*, pages 2814–2828.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. 2024. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *ICLR*, pages 1–31.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. RotatE: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, pages 1649–1667.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *ICML*, pages 2071–2080.

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

Junhong Wan, Tao Yu, Kunyu Jiang, Yao Fu, Weihao Jiang, and Jiang Zhu. 2025. Digest the knowledge: Large language models empowered message passing for knowledge graph question answering. In *ACL*, pages 15426–15442.

Cunda Wang, Ziying Ma, Po Hu, Weihua Wang, and Feilong Bao. 2026. Debate to align: Reliable entity alignment through two-stage multi-agent debate. *Preprint*, arXiv:2604.13551.

Liang Wang, Wei Zhao, Zhuoyu Wei, and Jingming Liu. 2022. SimKGC: Simple contrastive knowledge graph completion with pre-trained language models. In *ACL*, pages 4281–4294.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, pages 1–24.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, pages 24824–24837.

Changyi Xiao and Yixin Cao. 2024. Knowledge graph completion by intermediate variables regularization. In *NeurIPS*, pages 110218–110245.

Yao Xu, Shizhu He, Jiabei Chen, Zihao Wang, Yangqiu Song, Hanghang Tong, Guang Liu, Jun Zhao, and Kang Liu. 2024. Generate-on-graph: Treat LLM as both agent and KG for incomplete knowledge graph question answering. In *EMNLP*, pages 18410–18430.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*, pages 1–12.

Liang Yao, Chengsheng Mao, and Yuan Luo. 2019. KG-BERT: BERT for knowledge graph completion. *Preprint*, arXiv:1909.03193.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *ICLR*, pages 1–33.

Donghan Yu, Sheng Zhang, Patrick Ng, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Yiqun Hu, William Wang, Zhiguo Wang, and Bing Xiang. 2023. Decaf: Joint decoding of answers and logical forms for question answering over knowledge bases. *Preprint*, arXiv:2210.00063.

Duanyang Yuan, Sihang Zhou, Xiaoshu Chen, Dong Wang, Ke Liang, Xinwang Liu, and Jian Huang. 2025. Knowledge graph completion with relation-aware anchor enhancement. In *AAAI*, pages 15239–15247.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *Preprint*, arXiv:2506.05176.

Ruilin Zhao, Feng Zhao, Long Wang, Xianzhi Wang, and Guandong Xu. 2024. Kg-cot: Chain-of-thought prompting of large language models over knowledge graphs for knowledge-aware question answering. In *IJCAI*, pages 6642–6650.

Zihan Zhou, Yinan Liu, Yuyang Xie, Bin Wang, Xiaochun Yang, and Zezheng Feng. 2026. Diaglink: A dual-user diagnostic assistance system by synergizing experts with llms and knowledge graphs. *Preprint*, arXiv:2601.20311.

Yihua Zhu, Qianying Liu, Akiko Aizawa, and Hidetoshi Shimodaira. 2025. Beyond chains: Bridging large language models and knowledge bases in complex question answering. *Preprint*, arXiv:2505.14099.

A Statistics of KBs

KBC Task. The statistics of KBs are shown in Table 5 and Table 6. Note that isolated nodes refer to nodes that do not have any edges connecting to other nodes, and these nodes maintain self-loops (edges pointing to themselves). We retain those samples which have isolated entities (entities without any neighboring nodes). For the entity e that is an isolated node, we add a triple, represented as $\langle e, noop, e \rangle$.

Data set	*Entities*	*Relations*	*Train*	*Valid*	*Test*	*Isolated nodes*
WebQSP	749,492	1,251	1,714,822	20,000	20,000	130,179
CWQ	672,970	1,232	1,479,924	20,000	20,000	119,654

Table 5: Statistics of the 50% KB for the two data sets.

Data set	*Entities*	*Relations*	*Train*	*Valid*	*Test*	*Isolated nodes*
WebQSP	749,492	1,251	1,217,899	20,000	20,000	267,114
CWQ	672,970	1,232	1,060,601	20,000	20,000	244,439

Table 6: Statistics of the 30% KB for the two data sets.

KBQA Task. The statistics of the KBQA data sets used in this paper are shown in Table 7. Note that QA pairs in the training set without topic entities are removed. It can be seen that simply using the operation edge traverse on the complete KB could achieve nearly 100% accuracy.

Data set	*Answer format*	*Train*	*Test*	*Coverage*	*Licence*
WebQSP	Entity/Number	2777	3531	0.992	CC Licence
CWQ	Entity	23013	1639	0.997	-

Table 7: Statistics of KBQA data sets. Coverage refers to the accuracy of subgraph matching.

Reasoning Paths. The statistics of reasoning paths are shown in Table 8. In both KBQA data sets, the inference paths involve less than 1% of the KBC test triples, indicating that even if the KBQA reasoning path is fully learned, its impact on the KBC test set remains minimal.

Data set	$\textbf{\textit{Number}}_{path}$	$\textbf{\textit{Number}}_{test}$	$\textbf{\textit{Number}}_{test}/\textbf{\textit{Number}}_{path}$
WebQSP	8,623	27	0.31
CWQ	39,776	73	0.18

Table 8: Statistics of the reasoning path. $\textbf{\textit{Number}}_{path}$ denotes the number of triples in the inference paths. $\textbf{\textit{Number}}_{test}$ denotes the number of triples belonging to inference paths within the KBC test set.

B Supplementary Experiment Settings

Evaluation Measures. For KBC, we adopt the evaluation measures mean reciprocal rank (MRR) and Hits@i ($i \in \{1, 3, 10\}$). MRR is calculated by averaging the reciprocal ranks of true tail entities over all test triples. Hits@i evaluates the proportion of true tail entities in the top i predictions. For KBQA, we adopt the evaluation measure Hits@1.(the proportion of true answers in the top 1 predictions) to evaluate all methods.

KBC task. For DIFT, we utilize SimKGC as the pre-trained KBC model. For IVR, we use ComplEx as the backbone. For BiNet, we utilize the BFS method to construct relational paths.

KBQA task. For ToG and GoG, we change the prompt to adapt to Wikidata. For BiNet, we adopt the same setting as introduced above.

C Baselines

C.1 KBC Methods

For this task, we compare JCQL with eleven SOTA approaches as follows:

- TransE (Bordes et al., 2013) computes the candidate tail entity' score by representing relationships as vector translations.
- DistMult (Yang et al., 2015) uses a bilinear scoring function with a diagonal relationship matrix to calculate scores of candidate tail entities.
- ComplEx (Trouillon et al., 2016) extends DistMult by embedding entities and relations in a complex space.
- RotatE (Sun et al., 2019) obtains the score of tail entities by defining the relationship as rotations to capture a wider range of relational patterns.
- IVR (Xiao and Cao, 2024) represents entities and relationships as embedding matrices and computes scores using a tensor decomposition model.
- GoldE (Li et al., 2024a) computes the score of tail entities by applying hyperbolic orthogonal transformations to head entities and calculating the inner product with tail entities to capture hierarchical and geometric structures in knowledge graphs.
- SimKGC (Wang et al., 2022) leverages contrastive learning to optimize query embeddings derived from BERT, and computes the scores of candidate tail entities using the cosine similarity.

$\mathcal{T}$	$\mathcal{D}$	*WebQSP (30% KB)*				*WebQSP (50% KB)*				*CWQ (30% KB)*				*CWQ (50% KB)*			
		MRR	*Hits@1*	*Hits@3*	*Hits@10*	*MRR*	*Hits@1*	*Hits@3*	*Hits@10*	*MRR*	*Hits@1*	*Hits@3*	*Hits@10*	*MRR*	*Hits@1*	*Hits@3*	*Hits@10*
✓	✓	**0.603**	**0.572**	**0.628**	**0.652**	**0.630**	**0.599**	**0.652**	**0.693**	**0.588**	**0.556**	**0.612**	**0.645**	**0.614**	**0.583**	**0.634**	**0.670**
✓	✗	0.566	0.533	0.583	0.626	0.598	0.565	0.620	0.657	0.530	0.496	0.552	0.595	0.582	0.548	0.605	0.643
✗	✓	0.581	0.550	0.603	0.634	0.603	0.572	0.626	0.656	0.570	0.539	0.590	0.625	0.598	0.568	0.620	0.650
✗	✗	0.517	0.485	0.537	0.581	0.544	0.509	0.565	0.610	0.503	0.467	0.524	0.569	0.536	0.501	0.559	0.602

Table 9: Ablation study of the effect of the input sequence form of the SLM.

- CSProm-KG (Chen et al., 2023) uses the embedding of entities and relationships to generate conditional soft prompts, which are then entered into the frozen SLM to predict the tail entity.
- DIFT (Liu et al., 2024) predicts candidate tail entity by a pre-trained KBC model and refines the selection process with LLaMA.
- SATKGC (Ko et al., 2025) proposes a subgraph-aware training framework that incorporates structural inductive biases into SLMs by utilizing random-walk based subgraph sampling as mini-batches and employing contrastive learning to prioritize topologically hard negatives based on shortest path distances.
- RAA-KGC (Yuan et al., 2025) proposes a relation-aware anchor enhancement strategy that utilizes neighbors of the head entity sharing the same relation as context.
- BiNet (Liu et al., 2022a) converts multi-hop questions into relational paths via an encoder-decoder structure and utilizes a shared embedding space with an answer scoring module for joint optimization of KBC and KBQA.
- KGT5 (Saxena et al., 2022) integrates KBC and KBQA tasks into a unified framework by converting the query and the question into a simple input sequence form and fine-tuning the model T5.

C.2 KBQA Methods

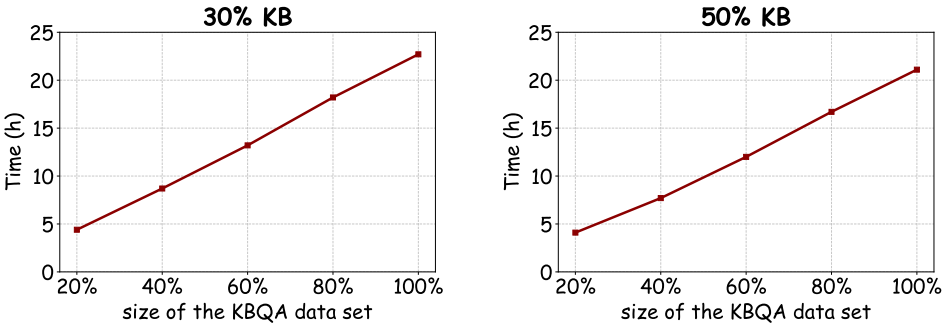
For this task, in addition to BiNet and KGT5, we use other baselines as follows:

- GPT-4o-mini is adopted under the standard prompting(Brown et al., 2020) without requiring the KB.
- Chain-of-Thought (CoT) prompt (Wei et al., 2022) solves questions through step-by-step reasoning without requiring the KB.
- Self-Consistency (SC) (Wang et al., 2023) can generate multiple candidate answers via CoT and select the most consistent answer.
- KG-CoT (Zhao et al., 2024) leverages a step-by-step graph reasoning model to perform reasoning over KBs and generates chains of knowledge with high confidence for LLMs based on the reasoning paths.
- ToG (Sun et al., 2024) explores relation paths within KBs through iterative interactions with the LLM and let the LLM derive final answers based on the retrieved paths.
- PoG (Chen et al., 2024) adaptively explores and self-corrects reasoning paths to efficiently answer questions by leveraging guidance, memory, and reflection.
- GoG (Xu et al., 2024) retrieves triples from the KB and generates novel triples through the LLM to answer complex questions in the incomplete KB.
- LMP (Wan et al., 2025) adapts the message passing paradigm to the textual domain by iteratively condensing neighborhoods into hierarchical semantic facts, enabling LLMs to digest structural knowledge for effective reasoning.
- PDRR (Zhu et al., 2025) employs a semantic parsing-inspired framework to predict question types and decompose them into structured triples, enabling flexible planning for both chain and parallel reasoning over KGs.

D Supplementary Experimental Results

D.1 Effect Analysis of the Input Sequence Form of the SLM

To verify the effectiveness of two kinds of information (i.e., entity description $\mathcal{D}$ and context $\mathcal{T}$) introduced in Section 3.3, we remove them from



(a) (b)

Figure 4: Efficiency study of the training process on WebQSP.

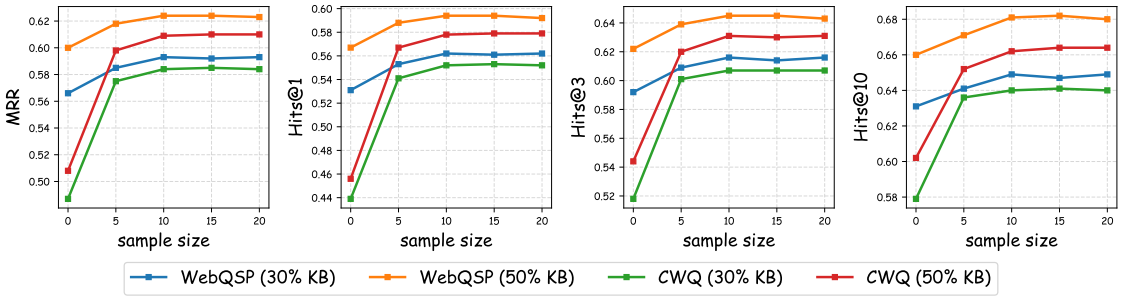


Figure 5: Performance of JCQL with different sample numbers in experience replay.

JCQL. As shown in Table 9, without $\mathcal{D}$ and $\mathcal{T}$, the performance of JCQL declines for KBC task on both data sets under different settings, which demonstrates the effectiveness of them.

D.2 Efficiency Study of Training Process

In this subsection, we study the efficiency of JCQL's training process under different settings. Figure 4(a) and Figure 4(b) plot the total running time on WebQSP under different settings i.e., 30% and 50% of the KB, respectively. From the results, we can see that the total running time is approximately linear to the number of questions in the training data set, which is consistent with calls to the LLM described in Section 3.4. Note that the running time of $\textsc{Parse}(C_q, \mathcal{M}_L)$ and $\textsc{IncFineTune}(\{\langle h, r, t\rangle\}, P_q, \mathcal{M}_S)$ is negligible compared with $\textsc{Agent}(q, a_q, e_q, \mathcal{M}_S)$. Therefore, the training time mainly depends on $\textsc{Agent}(q, a_q, e_q, \mathcal{M}_S)$.

D.3 Performance of JCQL with different sample size in experience replay.

To explore the influence of the sample size in experience replay on JCQL's performance, we conduct experiments with the size set to $\{0, 5, 10, 15, 20\}$. As shown in Figure 5, JCQL's performance improves with the sample size. This suggests that our experience replay strategy can effectively mitigate catastrophic forgetting in SLM. It also demonstrates that performance growth diminishes when the sample size exceeds 10. To balance performance and computational cost, we set the sample size to 10.

E Prompt List

E.1 Prompts for JCQL

As shown in Prompt 1, the prompt is designed to extract the reasoning process of KBQA.

Prompt 1. *The Prompt for the Training Phase of JCQL*

• Instruction:

Solve a question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be four types:
(1) Search[entity1 | entity2 | ...], which searches the exact entities on Wikidata and returns their

one-hop subgraphs. You should extract all concrete entities appeared in your last thought without redundant words, and you should always select entities from topic entities in the first search.
(2) Complete[head entity | relation], which completes tail entities based on the head entity and relationships.
(3) Generate[subquestion], which generates some new triples related to your last thought. The subquestion should be a one-hop question.
(4) Finish[answer1 | answer2 | ...], which summarizes the answers from the triples and returns the true answers. You should reason about the answer based on the known triples. Firstly, you need to summarize all possible answers appeared in the known triples. Secondly, you should compare them with the reference answers. Finally, you should return all the true answers. If you don't know the answer, please return Finish[unknown].
You should generate each step without redundant words.

• Example:

Question: where did woodrow wilson go to school?
Topic Entity: [Woodrow Wilson]
Answer: Davidson College | University of Virginia School of Law | Princeton University | Johns Hopkins University
Thought 1: The task requires identifying educational institutions attended by Woodrow Wilson. Based on the question, I will focus on gathering educational institutions linked to him.
Action 1: Search[Woodrow Wilson]
Observation 1: Woodrow Wilson, educated at, Davidson College
Woodrow Wilson, employer, Bryn Mawr College
Woodrow Wilson, educated at, University of Virginia
Thought 2: While Davidson College and the University of Virginia are identified, I suspect there are more institutions. I will use the "Complete" action with "Woodrow Wilson" and the relation "educated at" to find additional schools.
Action 2: Complete[Woodrow Wilson | educated at]
Observation 2: Woodrow Wilson, educated at, Princeton University
Thought 3: The result of Complete Action shows that Princeton University is also the school attended by Woodrow Wilson. I need to conduct further analysis to determine if there are other possible institutions.
Action 3: Generate[which additional educational institutions are linked to Woodrow Wilson?]
Observation 3: Woodrow Wilson, educated at, Johns Hopkins University
Thought 4: Firstly, I have gathered four educational institutions associated with Woodrow Wilson: Davidson College, University of Virginia, Princeton University, and Johns Hopkins University. Secondly, I compare them completely with the reference answers. The University of Virginia School of Law is missing.
Action 4: Finish[Davidson College | University of Virginia | Princeton University | Johns Hopkins University]
Observation 4: Davidson College | Princeton University | Johns Hopkins University

As shown in Prompt 2, the prompt is designed to executive reasoning for KBQA via the agent. The blue font denotes the retrieved answers, the red font denotes the action `generate`, and the green font denotes the action `complete`.

Prompt 2. *The Prompt for the Inference Phase of JCQL*

• Instruction:

Solve a question answering task with interleaving Thought, Action, Observation steps. Thought can reason about the current situation, and Action can be four types:
(1) Search[entity1 | entity2 | ...], which searches the exact entities on Wikidata and returns their one-hop subgraphs. You should extract all concrete entities appeared in your last thought without redundant words, and you should always select entities from topic entities in the first search.
(2) Complete[head entity | relation], which completes tail entities based on the head entity and relationships.
(3) Generate[subquestion], which generates some new triples related to your last thought. The

subquestion should be a one-hop question.
(4) Finish[answer1 | answer2 | ...], which summarizes the answers based on all the known triples. You should reason about the answer based on the known triples. If you don't know the answer, please return Finish[unknown].
You should generate each step without redundant words.

• Example:

Question: Which one of Shakespeare's parents worked as a business person?
Topic Entity: [William Shakespeare]
Thought 1: I need to find Shakespeare's parents firstly.
Action 1: Search[William Shakespeare]
Observation 1: Judith Quiney, father, William Shakespeare
Thought 2: The search results did not provide specific information about William Shakespeare's parents or their occupations. Therefore, I need to identify Shakespeare's father.
Action 2: Generate[who are William Shakespeare's father?]
Observation 2: William Shakespeare, father, John Shakespeare
Thought 3: I could find the father of William Shakespeare is John Shakespeare. I need to identify Shakespeare's mother then.
Action 3: Generate[who are William Shakespeare's mother?]
Observation 3: William Shakespeare, mother, Mary Shakespeare
Thought 4: According to the generate results, I can find the father of William Shakespeare is John Shakespeare and the mother is Mary Shakespeare. So next I will search for some information about their occupations.
Action 4: Search[John Shakespeare | Mary Shakespeare]
Observation 4: John Shakespeare, child, Joan Shakespeare
Mary Shakespeare, spouse, John Shakespeare
Mary Shakespeare, occupation, writer
Thought 5: I find the occupation of Mary Shakespeare is writer, but the search results did not provide specific information about John Shakespeare's occupations. Therefore, I need to identify his business-related activities.
Action 5: Complete[John Shakespeare | occupation]
Observation 5: John Shakespeare, occupation, merchant
John Shakespeare, occupation, politician
Thought 6: John Shakespeare is identified as William Shakespeare's father with occupations as a glove maker and a local government official. This connects him to the business realm. Therefore, I can summarize the answers.
Action 6: Finish[John Shakespeare]

E.2 Supplementary Prompts for JCQL

As shown in Prompt 3, the prompt is designed to select entities that are most related to the last `thought` based on the description of entities. During the reasoning, LLM may generate ambiguous entity names. With Prompt 3, LLM will select the suitable entity.

Prompt 3. *The Prompt for Entity Selection*

• Instruction:

Given entities and their descriptions, please select the entity that is most related to the thought.

• Example:

Thought: Based on the given observation, I need to find out who the leader of America is.
Entity Name: Libya
Candidate Entities:
Q30: country primarily located in North America
Q106315054: vocal track by Donna Fargo; 1974 studio recording
Q99281400: country of the United States of America as depicted in Star Trek
Answer: Q30

As shown in Prompt 4, the prompt is designed to select relations that are most related to the last `thought` in the action `search`.

Prompt 4. *The Prompt for Relation Selection*

• Instruction:

Please select 3 relations that most relevant to the thought and rank them.

• Example:

Thought: The task requires identifying educational institutions attended by Woodrow Wilson. I will focus on gathering educational institutions associated with him.
Entity Name: Woodrow Wilson
Relations: sibling, award received, signatory, owned by, doctoral student, spouse, member of, educated at, child, occupation, described by source, archives at, topic's main category, member of political party, place of burial, successful candidate, father, residence, employer, named after, work location, writing language, position held, sex or gender, candidate, given name, family name, candidacy in election
Answers: educated at, employer, named after

As shown in Prompt 5, the prompt is designed to generate new triples that are most related to the sub-question in the action `generate`. The pink font indicates the supervision in the training phase of the KBQA task.

Prompt 5. *The Prompt for Triple Generation*

• Instruction:

Given the existing triples, please think step by step and generate new triples related to the current question.

• Example:

Question: what are the types of government in the United States?
Hint: federal republic | constitutional republic | presidential system
Known Triples: """
United States of America, instance of, country
United States of America, instance of, constitutional republic
United States of America, has cabinet, United States Cabinet
United States of America, basic form of government, republic
"""
Generated Triples: """
1. United States of America, instance of, federal republic
2. United States of America, instance of, Democratic Republic
3. United States of America, basic form of government, presidential system
"""

We first link the relations within the generated triples to the canonical relations in the background KB using the BM25 algorithm. As shown in Prompt 6, the prompt is then designed to modify the triples generated by LLM. The LLM often frequently generates unreliable triples. For instance, this prompt may allow the LLM invert head-tail entity pairs, such as modifying ⟨Pak Pong-ju, head of government, North Korea⟩ into ⟨North Korea, head of government, Pak Pong-ju⟩. Specifically, as a preprocessing step, for each relation in the background KB, the description and schema are obtained by retrieving the original description from Wikidata and extracting multiple corresponding triples from the background KB. Subsequently, the original description and these triples are input into the LLM to generate refined description and to summarize the schema from the triples.

Prompt 6. *The Prompt for Triple Modification*

• Instruction:

Please modify the known triples based on the descriptions and schemas of the relations. You should generate triples without redundant words.

• Example:

Question: who is ruling north korea now?
Known Triples: """
North Korea, office held by head of state, Kim Il-sung
North Korea, office held by head of state, Kim Jong-un
Pak Pong-ju, head of government, North Korea
"""
Descriptions: """
1. office held by head of state: This relation links a geographical or political entity...
2. member of political party: This relation connects individuals to the political parties...
3. head of government: This relation links a specific location—such as a city, municipality...
4. head of state: This relation links a geographical

input	⟨ Justin Bieber, father, ? ⟩
sequence	predict tail: Justin Bieber | father entity description: Justin Bieber [Canadian singer (born 1994)] related relationship: father context: ⟨ Justin Bieber | mother | Pattie Mallette ⟩ $\langle SEP \rangle$ ⟨ Justin Bieber | sibling | Jazmyn Bieber⟩ $\langle SEP \rangle$ ...
output	Jeremy Bieber

Table 10: A typical case of input sequences and corresponding output for the SLM.

entity, such as a country or state...
"""
Schemas: """
1. office held by head of state: [Location], office held by head of state, [Political Office]
2. member of political party: [Human], member of political party, [Political Party]
3. head of government: [Location], head of government, [Human]
4. head of state: [Location], head of state, [Human]
"""
New Triples: """
1. North Korea, head of state, Kim Il-sung
2. North Korea, head of state, Kim Jong-un
3. North Korea, head of government, Pak Pong-ju
"""

As shown in Prompt 7, the prompt is designed to select the triples most relevant to the question to construct reasoning paths.

Prompt 7. *The Prompt for Path Generation*

• Instruction:

Considering the known triples and relations, please select the triples and relations that are relevant to the questions and answers.

• Example:

Question: which one of Shakespeare's parents worked as a business person?
Answer: John Shakespeare
Topic entity: William Shakespeare
Known triples: """
William Shakespeare, field of work, acting
William Shakespeare, field of work, poetry
William Shakespeare, field of work, theatre
William Shakespeare, father, John Shakespeare
William Shakespeare, mother, Mary Shakespeare
Mary Shakespeare, spouse, John Shakespeare
Mary Shakespeare, occupation, writer
John Shakespeare, occupation, merchant
John Shakespeare, occupation, politician
"""
Related relations: father, mother, occupation
Related triples: """
William Shakespeare, father, John Shakespeare
William Shakespeare, mother, Mary Shakespeare
Mary Shakespeare, occupation, writer
John Shakespeare, occupation, merchant
John Shakespeare, occupation, politician
"""

F Case Study

Comparison between `generate` and `complete` under 50% KB. JCQL treats LLM as both agent and KB, generating new triples from the internal knowledge of the LLM. However, when triples are missing, action `generate` over-relies on the LLM's internal knowledge, increasing the risk of hallucination. To alleviate this issue, our approach incorporates the SLM, ensuring that predicted relations and entities are assigned unique IDs. As shown in Table 11, action `generate` incorrectly generates relationships and entities that do not exist in the KB. In contrast, JCQL generates valid entities through its action `complete`.

A typical case of input sequences and corresponding output for the SLM. table 10 shows a typical case of input sequences and corresponding output for the SLM. Given the query $\langle Justin\ Bieber, father, ? \rangle$, JCQL obtains triples

with the one-hop neighborhoods of the head entity $Justin\ Bieber$ uniformly, at random and without replacement. Finally, we concatenate them with $\langle SEP \rangle$.

G Supplementary Setting Details

In all experiments, the depth of exploration is set to 10 to avoid endless exploration. The maximum token length for generation is set to 512. All experiments are conducted on a server with an Intel(R) Xeon(R) Silver 4310 CPU (2.10GHz), 256 GB RAM memory, and an NVIDIA RTX A6000 GPU.

H Broader Impact

By performing KBQA and KBC jointly, JCQL enhances the complex reasoning abilities of the KB-augmented LLM and improves the performance of the SLM when faced with incomplete KBs. Furthermore, other broader impacts of JCQL are listed as follows. (1) The proposed framework JCQL is flexible. For KBQA, we can choose any KBC model to complete missing triples for any KB. (2) JCQL can generate reasoning paths based on input questions and update the KB in real time using incremental learning techniques, demonstrating its real-time performance and the ability of lifelong learning. (3) From a long-term perspective, the LLM's reasoning paths generated from continuous increasing of question-answer pairs strengthen the KBC model, which in turn improves the reasoning capability of LLM.

Question	what type of government did soviet union have?
JCQL w/o `complete`	Thought 1: I need to find out what type of government the Soviet Union had. Action 1: Search[Soviet Union] Observation 1: soviet union, history of topic, history of the soviet union soviet union, legislative body, supreme soviet of the soviet union soviet union, official religion, state atheism Thought 2: The Soviet Union had a legislative body called the Supreme Soviet, which implies a type of government. I need to generate more information about the government structure of the Soviet Union based on this observation. Action 2: Generate[type of government of the Soviet Union] Observation 2: soviet union, ideological basis, Marxism-Leninism soviet union, type of government, one-party socialist republic Thought 3: The Soviet Union is characterized as a one-party socialist republic based on its ideological basis of Marxism-Leninism. Action 3: Finish[one-party socialist republic]
JCQL w/o `generate`	Thought 1: The question requires identifying the type of government in the Soviet Union. I should gather information regarding the political structure and governance of the Soviet Union. Action 1: Search[Soviet Union] Observation 1: Soviet Union, foundational text, Declaration of the Creation of the Union of Soviet Socialist Republics Soviet Union, foundational text, Treaty on the Creation of the Union of Soviet Socialist Republics Soviet Union, legislative body, Supreme Soviet of the Soviet Union member of the Supreme Soviet of the Russian Soviet Federative Socialist Republic, country, Soviet Union Thought 2: The current information provides some context about the Soviet Union but lacks specific details about its type of government. I need to find the governance structure of the Soviet Union. Action 2: Complete[Soviet Union | instance of] Observation 2: Soviet Union, instance of, country Soviet Union, instance of, socialist state Thought 3: The results indicate that the Soviet Union was a socialist state. This clarifies the type of government. Therefore, I can summarize the answers. Action 3: Finish[socialist state].

Table 11: The comparison between `generate` and `complete` under the setting i.e., 50% KB.