KB-Plugin: A Plug-and-play Framework for Large Language Models to Induce Programs over Low-resourced Knowledge Bases

Anonymous ACL submission

Abstract

Program induction (PI) has become a promising paradigm for using knowledge bases (KBs) to help large language models (LLMs) answer complex knowledge-intensive questions. Nonetheless, PI typically relies on a large number of parallel question-program pairs to make the LLM aware of the schema of a given KB, and is thus challenging for many low-resourced KBs that lack annotated data. To this end, we propose **KB-Plugin**, a plug-and-play framework that enables LLMs to induce programs over any low-resourced KB. Firstly, KB-Plugin adopts self-supervised learning to encode the detailed schema information of a given KB into a pluggable module, namely **schema plugin**. Secondly, KB-Plugin utilizes abundant annotated data from a rich-resourced KB to train another pluggable module, namely **PI plugin**, which can help the LLM extract question-relevant schema information from the schema plugin of any KB and utilize the information to induce programs over this KB. Experiments show that KB-Plugin outperforms SoTA low-resourced PI methods with 25× smaller backbone LLM on both large-scale and domain-specific KBs, and even approaches the performance of supervised methods.

1 Introduction

Recently, the usage of knowledge bases (KBs) as external resources to assist large language models (LLMs) (Brown et al., 2020; Zhao et al., 2023) in answering complex knowledge-intensive questions has gained increasing study (Pan et al., 2023; Li et al., 2023b; Jiang et al., 2023). Among various methods, program induction (PI) has emerged as a promising paradigm due to its good interpretability and capacity to support complex reasoning operations (Cao et al., 2022a; Gu et al., 2023; Li et al., 2023b). Given a KB, PI methods employ LLMs to convert a question into a multi-step program (e.g., KoPL (Cao et al., 2022a) and S-expression (Su et al., 2016)), whose execution against the KB produces the answer. Despite strong capacity, most PI methods rely on individual training for each KB using a large number of manually annotated question-program pairs (Xie et al., 2022; Li et al., 2023b; Luo et al., 2023). As for many low-resourced KBs that lack program annotations, how to enable LLMs to utilize their knowledge via PI remains a challenging problem.

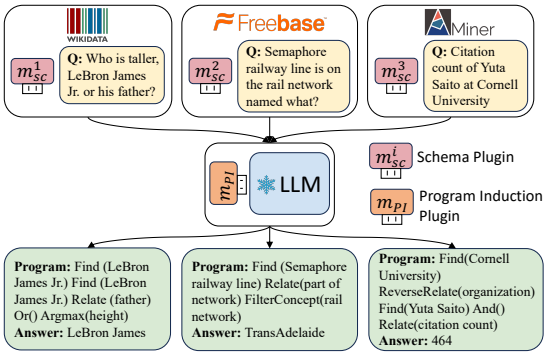

Figure 1: Illustration of KB-Plugin. By simply plugging the schema plugin of a KB and the PI plugin, the LLM is injected with the schema information of this KB and the ability to induce programs over it.

Recent studies (Cao et al., 2022b; Li et al., 2023a) have indicated that the mapping from questions to program sketches (i.e., composed functions without arguments, such as `Find`→ `Relate`→ `FilterConcept`) primarily correlates with language compositional structures and is thus transferable across KBs. Hence the main challenge for PI over low-resourced KBs is to determine the argument for each function (Gu and Su, 2022), which requires LLMs to link natural language in a question to corresponding schema items (i.e., pre-defined relations and concepts) in the KB (e.g., in Fig 1, the relation *"part of network"* and the concept *"rail network"* are arguments of function `Relate` and `FilterConcept`, respectively), so it is impor-

tant to provide LLMs adequate information of each schema item. A straightforward approach is to directly feed all the schema information to the LLM via a prompt. However, the broad schema of KBs and limited context windows of LLMs make this infeasible (Li et al., 2023a).

Regarding the above challenges, we are inspired by recent studies that claim the parameters of LLMs can encode task-specific knowledge (Saxena et al., 2022; Moiseev et al., 2022; Wang et al., 2022). Our basic idea is to encode detailed schema information of a KB into the parameters of a pluggable module (e.g., LoRA (Hu et al., 2022)), namely **schema plugin**, so as not to be hampered by limited context windows like the prompt-based approach. Then we use another pluggable module, namely **PI plugin**, to help the LLM capture question-relevant schema information from the schema plugin and utilize this information to induce programs. As illustrated in Fig. 1, by simply plugging the schema plugin of a KB and the PI plugin, the LLM is injected with the schema information of this KB and the ability to induce programs over it. We name this framework **KB-Plugin**. To implement KB-Plugin, there remain two key problems: (1) By what task can sufficient information about each schema item in a KB be encoded into its schema plugin? (2) Without annotated data from the low-resource KBs, how can the PI plugin learn to extract and utilize question-relevant schema information from their schema plugins to induce programs over these KBs?

To solve the above problems, we propose a novel plugin learning and transfer framework. First, inspired by prior studies (Bordes et al., 2013; Lin et al., 2015) which show that schema items in a KB can be well represented by fact triples involving them, we propose to learn schema plugins via a self-supervised triple completion task. Specifically, given a KB, we plug a schema plugin into the LLM and tune the plugin to enable the LLM to complete relevant triples for each schema item in the KB. In this way, the detailed schema information can be encoded into this schema plugin. As for PI plugin learning, inspired by Cao et al. (2022b), we utilize abundant program annotations from a rich-resourced KB. Specifically, we use this KB to generate multiple KBs with different schemas via alias replacement and train a schema plugin for each of them. Given a training question, we plug these schema plugins alone with the PI plugin into the LLM in turn and train the PI plugin to make the LLM generate the correct program whose arguments conform to the currently plugged schema plugin. In this way, the PI plugin is forced to learn the skills of extracting and utilizing question-relevant schema information from the plugged schema plugin for PI over the corresponding KB. Besides, since the PI plugin is trained to be compatible with different schema plugins, it can be directly transferred to other low-resourced KBs and generalize well with their schema plugins, even if most schema items in these KBs are unseen during its training.

In experiments, we take Wikidata-based KQA Pro as the rich-resourced KB to train the PI plugin, and evaluate our framework on three large-scale Freebase-based datasets (WebQSP, GraphQ, and GrailQA) and two domain-specific datasets (MetaQA for movie domain and SoAyBench for academic domain). The results show that KB-Plugin outperforms SoTA low-resourced PI methods with 25× smaller backbone LLM, demonstrating its scalability to extremely complex schemas and adaptability to various domains. On GraphQ, GrailQA, and MetaQA, KB-Plugin even surpasses the performance of several supervised methods.

Our contributions include: (1) proposing KB-Plugin, a novel plug-and-play framework that enables LLMs to induce programs over any low-resourced KB; (2) empirically validating the efficacy of KB-Plugin for both large-scale and domain-specific KBs through extensive experiments.

2 Related Work

Low-resourced Program Induction. Recently, there have emerged three types of PI methods for low-resourced KBs that lack program annotations, but each of them has limitations: (1) Few-shot program generation methods (Gu et al., 2023; Li et al., 2023a) utilize in-context learning ability of LLMs to induce programs with a handful of demonstrations. However, they can only determine function arguments based on the schema item names due to limited context windows, so they face challenges in distinguishing similar schema items. They also suffer from long inference time due to excessive LLM calls or executing a vast number of potential programs; (2) Few-shot data generation methods (Li et al., 2023c) also employ in-context learning with LLMs to convert automatically sampled programs into questions, and train a smaller PI model using the generated question-program pairs.

Nonetheless, the generated questions may not align with programs and often lack diversity due to the limited number of program templates; (3) Similar to us, program transfer methods (Cao et al., 2022b) also leverage program annotations from a rich-resourced KB to aid PI for low-resourced KBs. However, they mainly focus on program sketch transfer and perform poorly without fine-tuning using annotated question-answer pairs from low-resourced KBs to adapt to their schemas. While KB-plugin obviates the reliance on any annotated data from low-resourced KBs, thereby enabling LLMs to easily utilize their knowledge.

Plug-and-Play Modules for LLMs. In recent years, various parameter-efficient modules have been proposed to adapt LLMs to different downstream tasks (Lester et al., 2021; Hu et al., 2022; Li and Liang, 2021; Pfeiffer et al., 2021) . These modules show plug-and-play characteristics and can inject task-specific knowledge and skills into LLMs (Xiao et al., 2023; Zhang et al., 2023). Some researchers also found that pluggable modules for similar tasks encode knowledge and skills into the parametric space in similar ways (Qin et al., 2021; Su et al., 2022), providing basic rationality for the transferability of our PI plugin.

3 Problem Formulation

In this section, we first provide some necessary definitions and then formulate our task.

Knowledge Base. A knowledge base (KB) can be formalized as $\mathcal{KB} = \{\mathcal{C}, \mathcal{E}, \mathcal{R}, \mathcal{T}\}$, where $\mathcal{C}$, $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{T}$ represent the sets of concepts, entities, relations and fact triples, respectively. Specifically, $\mathcal{R} = \{r_e, r_c\} \cup \mathcal{R}_l$, where r_e is *"instance of"*, r_c is *"subclass of"*, and $\mathcal{R}_l$ is the set of other general relations. Correspondingly, $\mathcal{T}$ can be divided into there disjoint subsets: (1) *"instance of"* triples $\mathcal{T}_e = \{(e, r_e, c)|e \in \mathcal{E}, c \in \mathcal{C}\}$; (2) *"subclass of"* triples $\mathcal{T}_c = \{(c_i, r_c, c_j)|c_i, c_j \in \mathcal{C}\}$; (3) relational triples $\mathcal{T}_l = \{(e_i, r, e_j)|e_i, e_j \in \mathcal{E}, r \in \mathcal{R}_l\}$. Elements in $\mathcal{C}$ and $\mathcal{R}$ are also called the schema items of $\mathcal{KB}$.

Program Induction. Given a KB $\mathcal{KB}$ and a natural language question $x = \langle w_1, w_2, \cdots, w_{|x|} \rangle$, program induction (PI) aims to convert x into a program y, which would return the correct answer when executed against $\mathcal{KB}$. Formally, y is composed of functions that take a specific type of arguments, and can be serialized as $y = \langle f_1(arg_1), \cdots, f_t(arg_t), \cdots, f_{|y|}(arg_{|y|}) \rangle, f_t \in \mathcal{F}, arg_t \in \mathcal{E} \cup \mathcal{C} \cup \mathcal{R} \cup \{\emptyset\}$. Here, $\mathcal{F}$ is a set of pre-defined functions that cover basic reasoning operations on KBs. In this work, we use KoPL (Cao et al., 2022a) as our programming language.

Task Formulation. Suppose we have access to (1) source KB $\mathcal{KB}^S$ and source domain data $\mathcal{D}^S = \{(x_i^S, y_i^S)\}_{i=1}^{n^S}$, which are question-program pairs for $\mathcal{KB}^S$; (2) target KB $\mathcal{KB}^T$, which is low-resourced and has no annotated data. The goal is to learn a PI model M_{PI}^T that can translate a question x^T for $\mathcal{KB}^T$ into program y^T, whose execution on $\mathcal{KB}^T$ produces the correct answer.

4 Methodology

As mentioned in the introduction, to enable a LLM M to induce programs over low-resourced $\mathcal{KB}^T$, KB-Plugin learns two types of pluggable modules for M: (1) KB-specific **schema plugin** m_{sc}, which stores information of schema items of a given KB within its parameters; (2) KB-transferable **PI plugin** m_{PI}, which encodes the skill of inducing programs over any KB by extracting and utilizing question-relevant schema information from the schema plugin of this KB. It is trained with $\mathcal{KB}^S$ and $\mathcal{D}^S$ but can be directly transferred to $\mathcal{KB}^T$. The final PI model for $\mathcal{KB}^T$ can be formulated as

$$M_{PI}^T = \text{plug}(M, \{m_{sc}^T, m_{PI}\}), \quad (1)$$

where m_{sc}^T is the schema plugin of $\mathcal{KB}^T$ and $\text{plug}(M, \{\cdot\})$ means plugging the plugins in $\{\cdot\}$ into M. In the following, we will first introduce the architecture of two types of plugins, then present our plugin learning and transfer framework.

4.1 Plugin Architecture

A host of studies have demonstrated that knowledge and skills can be encapsulated within the parameters of LLMs (Saxena et al., 2022; Moiseev et al., 2022; Wang et al., 2022). Inspired by this, we implement both schema plugin and PI plugin with LoRA (Hu et al., 2022), a popular type of pluggable module for LLMs with a few trainable parameters.

Specifically, let L_M be the set of weight matrices in the self-attention modules and MLP modules of a LLM M. For each $W_i \in \mathbb{R}^{d \times k}$ in L_M, LoRA modifies its forward pass from $h = W_i x$ to $h = (W_i + A_i B_i)x$, where $A_i \in \mathbb{R}^{d \times r}$ and $B_i \in \mathbb{R}^{r \times k}$ are two matrices with rank $r \ll \min(d, k)$. A LoRA plugin m_j is thus defined as

$$m_j = \{(A_i^{m_j}, B_i^{m_j})|W_i \in L_M\}, \quad (2)$$

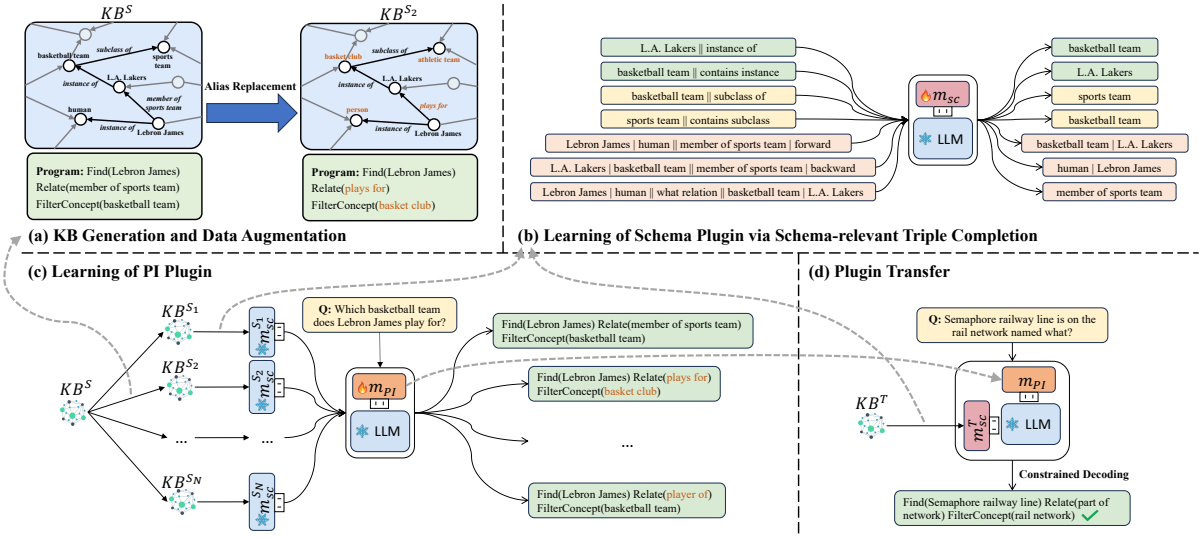

Figure 2: Overview of our plugin learning and transfer framework: (a) Generate multiple source KBs with different schemas and augmented source domain data via alias replacement; (b) Learn an individual schema plugin for each source KB and the target KB via self-supervised schema-relevant triple completion task; (c) Train the PI plugin by inducing program for each source KB when plugging it into the LLM along with the corresponding schema plugin. (d) Transfer the PI plugin by plugging it into the LLM with the schema plugin of the target KB and inducing programs over the target KB with constrained decoding.

and $\text{plug}(M, \{m_1, \ldots, m_N\})$ means replacing all $W_i \in L_M$ with $W_i + \sum_{j=1}^{N} A_i^{m_j} B_i^{m_j}$. If we train $M' = \text{plug}(\text{fz}(M), \{\text{fz}(m_1), \ldots, \text{fz}(m_{N-1}), m_N\})$ on a certain task, where $\text{fz}(\cdot)$ represents parameter freezing, knowledge and skills related to this task will be encoded within m_N. Although other parameter-efficient pluggable modules such as prefix-tuning (Li and Liang, 2021) can also serve as our plugin modules, the advantages of LoRA are that it does not increase input length or inference latency.

4.2 Plugin Learning and Transfer Framework

There are two primary challenges for learning schema plugins and the PI plugin: (1) How to encode sufficient information about each schema item of a KB into a schema plugin? (2) How to ensure that the PI plugin can extract and utilize useful schema information for program induction from schema plugins of different KBs, instead of ignoring the schema plugin entirely, directly learning to induce program over source KB during training, and consequently losing transferability?

To handle these challenges, we propose a novel plugin learning and transfer framework, which is illustrated in Fig. 2 and contains four steps: (1) Generate multiple source KBs $\mathcal{KB}^{S_1}, \ldots, \mathcal{KB}^{S_N}$ with different schemas and augmented data $\mathcal{D}_a^S = \{(x_j^S, y_j^{S_1}, \ldots, y_j^{S_N})\}_{j=1}^{n^S}$ based on $\mathcal{KB}^S$ and D^S via alias replacement, where $y_j^{S_i}$ is the golden program for question x_j^S on $\mathcal{KB}^{S_i}$; (2) Learn individual schema plugin $m_{sc}^{S_i}$ for each $\mathcal{KB}^{S_i}$ via self-supervised schema-relevant triple-completion task; (3) Train PI plugin m_{PI} by requiring $M_{PI}^{S_1}, \ldots, M_{PI}^{S_N}$ to generate $y_j^{S_1}, \ldots, y_j^{S_N}$ given x_j^S, respectively, where $M_{PI}^{S_i} = \text{Plug}(\text{fz}(M), \{\text{fz}(m_{sc}^{S_i}), m_{PI}\})$, so that m_{PI} is forced to extract and utilize schema information from each $m_{sc}^{S_i}$; (4) Learn schema plugin m_{sc}^T for $\mathcal{KB}^T$ using the same method in (2) and take $M_{PI}^T = \text{plug}(M, \{m_{sc}^T, m_{PI}\})$ as the final PI model for $\mathcal{KB}^T$. We will introduce each step in detail in the following.

4.2.1 KB Generation and Data Augmentation

We utilize the aliases of each schema item to generate multiple KBs with different schemas based on $\mathcal{KB}^S = \{\mathcal{C}^S, \mathcal{E}^S, \mathcal{R}^S, \mathcal{T}^S\}$. As shown in Fig. 2(a), for each schema item $v \in \mathcal{C}^S \cup \mathcal{R}^S$, we replace v with v_i, a randomly chosen alias of v, and record $a_i(v) = v_i$. For example, the concept "*basketball team*" can be replaced with "*basket club*" and the relation "*member of sports team*" can be replaced with "*plays for*". Relevant triples in $\mathcal{T}^S$ are also modified with the same alias. In this way, $\mathcal{KB}^{S_i}$ that has a different schema than $\mathcal{KB}^S$ is created. In practice, we let $\mathcal{KB}^{S_1} = \mathcal{KB}^S$ and repeat above

process $N-1$ times to generate $\mathcal{KB}^{S_2}, \ldots, \mathcal{KB}^{S_N}$.

Similarly, for each question-program pair $(x_j^S, y_j^S) \in \mathcal{D}^S$, suppose $y_j^S = \left\langle f_1(arg_1), \cdots, f_t(arg_t), \cdots, f_{|y_j^S|}(arg_{|y_j^S|}) \right\rangle$, we replace every $arg_t \in \mathcal{C}^S \cup \mathcal{R}^S$ with $a_i(arg_t)$ to obtain $y_j^{S_i}$, which is the correct program for x_j^S executable on $\mathcal{KB}^{S_i}$. We repeat the process for $\mathcal{KB}^{S_1}, \ldots, \mathcal{KB}^{S_N}$ to obtain augmented data $\mathcal{D}_a^S = \{(x_j^S, y_j^{S_1}, \ldots, y_j^{S_N})\}_{j=1}^{n^S}$.

4.2.2 Learning of Schema Plugin

Many studies about knowledge graph embedding show that the information of schema items in a KB can be represented by not only their names but also triples containing them (Bordes et al., 2013; Lv et al., 2018). Inspired by this, we propose to encode schema information into schema plugins via a self-supervised triple completion task. As illustrated in Fig. 2(b), to learn the schema plugin m_{sc} for a given KB $\mathcal{KB} = \{\mathcal{C}, \mathcal{E}, \mathcal{R}, \mathcal{T}\}$, where $\mathcal{T} = \mathcal{T}_e \cup \mathcal{T}_c \cup \mathcal{T}_l$, we train $M_{sc} = \text{Plug}(\text{fz}(M), m_{sc})$ to complete relevant triples for each concept and relation in $\mathcal{KB}$ in sequence-to-sequence form as follows.

First, for each concept $c \in \mathcal{C}$, we require M_{sc} to complete relevant *"instance of"* triples to aggregate the semantic features of entities belonging to c. Specifically, we sample K triples $(e_k, \textit{instance of}, c)$ from $\mathcal{T}_e$ (see Appendix B for detailed sampling strategy), and use each sampled triple to construct two pairs of verbalized queries and answer as the inputs and expected outputs for M_{sc}:

- "$\langle e_k \rangle$ || *instance of*" → "$\langle c \rangle$";
- "$\langle c \rangle$ || *contains instance*" → "$\langle e_k \rangle$".

Here, $\langle e_k \rangle$ and $\langle c \rangle$ means filling in the names of e_k and c, respectively.

Besides, the information of a concept is also related to its sub- and super-concepts. Therefore, for each triple $(c_i, \textit{subclass of}, c_j) \in \mathcal{T}_c$, we also construct two queries with answers for M_{sc}:

- "$\langle c_i \rangle$ || *subclass of*" → "$\langle c_j \rangle$";
- "$\langle c_j \rangle$ || *contains subclass*" → "$\langle c_i \rangle$".

Finally, the information of a relation can be learned from its name and the elements connected by it. Therefore, for each $r \in R_l$, we sample K triples (e_i, r, e_j) from $\mathcal{T}_l$, choose c_i, c_j such that $(e_i, \text{instance\_of}, c_i), (e_j, \text{instance\_of}, c_j) \in T_e$, and use each (e_i, c_i, r, e_j, c_j) to construct three queries with answers:

- "$\langle e_i \rangle$ | $\langle c_i \rangle$ || $\langle r \rangle$ | *forward*" → "$\langle c_j \rangle$ | $\langle e_j \rangle$";
- "$\langle e_j \rangle$ | $\langle c_j \rangle$ || $\langle r \rangle$ | *backward*" → "$\langle c_i \rangle$ | $\langle e_i \rangle$";
- "$\langle e_i \rangle$ | $\langle c_i \rangle$ || *what relation* || $\langle c_j \rangle$ | $\langle e_j \rangle$" → "$\langle r \rangle$".

We empirically find that including c_i, c_j benefits the information encoding for both concepts and relations.

Let the set of all generated queries and answers be $D_{sc} = \{(q_i, a_i)\}_{i=1}^{l}$, then m_{sc} is trained to minimize

$$\mathcal{L}_{sc} = -\sum_{(q_i, a_i) \in D_{sc}} \log P(a_i | q_i), \tag{3}$$

where $P(a_i|q_i)$ is the likelihood of M_{sc} generating a_i given q_i, defined by token-level cross entropy. Note that the learning of m_{sc} does not rely on any additional data except the KB itself, so we can train a schema plugin for any KB.

4.2.3 Learning of PI Plugin

As illustrated in Fig. 2(c), to learn the PI plugin m_{PI}, we first train individual schema plugin $m_{sc}^{S_i}$ for each $\mathcal{KB}^{S_i}$. After that, given $(x_j^S, y_j^{S_1}, \ldots, y_j^{S_N}) \in D_a^S$, where x_i^S is a question and $y_j^{S_i}$ is the golden program for x_j^S on $\mathcal{KB}^{S_i}$, we train m_{PI} by feeding x_i^S to $M_{PI}^{S_1}, \ldots, M_{PI}^{S_N}$ and requiring them to generate $y_j^{S_1}, \ldots, y_j^{S_N}$, respectively. Here, $M_{PI}^{S_i} = \text{Plug}(\text{fz}(M), \{\text{fz}(m_{sc}^{S_i}), m_{PI}\})$. The overall objective can be formulated as:

$$\mathcal{L}_{PI} = -\sum_{(x_j^S, y_j^{S_1}, \ldots, y_j^{S_N}) \in D_a^S} \sum_{i=1}^{N} \log P_i(y_j^{S_i} | x_j^S), \tag{4}$$

where $P_i(y_j^{S_i}|x_j^S)$ is the likelihood of $M_{PI}^{S_i}$ generating $y_j^{S_i}$ given x_j^S, defined by token-level cross entropy. To generate programs conforming to different schemas given the same question, m_{PI} must learn to (1) choose correct functions according to the compositional structure of the question; (2) extract and utilize question-relevant schema information for argument determination from the corresponding schema plugin, because it is the only difference among $M_{PI}^{S_1}, \ldots, M_{PI}^{S_N}$.

4.2.4 Plugin Transfer

Once the PI plugin m_{PI} is trained, we directly transfer it to $\mathcal{KB}^T$ as in Fig 2 (d), and let $M_{PI}^T = \text{plug}(M, \{m_{sc}^T, m_{PI}\})$ be the PI model for $\mathcal{KB}^T$. Here, m_{sc}^T is the trained schema plugin for $\mathcal{KB}^T$

using the method in Sec. 4.2.2. Since m_{sc}^T and $m_{sc}^{S_i}$ are trained with the same tasks, we expect that they encode schema information into their parameters in similar ways (Qin et al., 2021; Su et al., 2022), so m_{PI} can also extract schema information from m_{sc}^T to help PI over $\mathcal{KB}^T$. Besides, to guarantee M_{PI}^T generating valid programs which do not cause execution error or return an empty answer, we adopt constrained decoding, i.e., after M_{PI}^T generates $f_1(arg_1), \ldots, f_t(arg_t)$, we enumerate all the valid $f_{t+1}(arg_{t+1})$ following the method of Gu et al. (2023) and restrict M_{PI}^T to only generate one of them. More details are in Appendix C. We also use beam search to retain top-k programs during decoding to provide M_{PI}^T with a more global view.

5 Experiments

5.1 Datasets

Source Domain. We use KQA Pro (Cao et al., 2022a) as the source domain dataset. It provides 117,970 question-program pairs with diverse compositional structures based on a small subset of Wikidata (Vrandecic and Krötzsch, 2014).

Target Domain. We use WebQSP (Yih et al., 2016), GraphQ (Su et al., 2016), GrailQA (Gu et al., 2021), MetaQA (Zhang et al., 2018) and SoAyBench (Wang et al., 2024) as the target domain datasets. Among them, WebQSP, GraphQ, and GrailQA are based on Freebase (Bollacker et al., 2008). Their KBs contain a large number of schema items and cover various domains, thus can evaluate the effectiveness of KB-Plugin for large-scale KBs. MetaQA and SoAyBench are two datasets in movie and academic domains, respectively, and can evaluate the adaptability to specific domains in detail. For MetaQA, since most of the relations in its KB have been covered by KQA Pro, we remove these relations and relevant question-program pairs from KQA Pro to avoid data leakage. For SoAyBench which is originally a tool-using dataset based on Aminer (Tang et al., 2008) APIs, we construct its KB by collecting relevant data from these APIs. Table 1 shows the statistics of these datasets and their overlap with source KBs generated from KQA Pro. Most schema items in the target KBs are unseen in source KBs and most test cases also involve unseen schema items.

Dataset	$\lvert\mathcal{DM}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{R}_u\rvert$	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{C}_u\rvert$	$\lvert\mathcal{D}^{\text{test}}\rvert$	$\lvert\mathcal{D}_u^{\text{test}}\rvert$
KQA Pro	-	1209	-	794	-	-	-
WebQSP	56	412	296	446	363	1639	1083
GraphQ	70	9569	8931	7298	7004	2395	2340
GrailQA(dev)	86	3938	3524	2018	1868	6763	6578
GrailQA(test)	86	3938	3524	2018	1868	13231	-
MetaQA	1	9	9	9	3	39093	39093
SoAyBench	1	17	11	5	3	792	756

Table 1: Statistics for source and target domain datasets and their overlaps with 16 source KBs generated from KQA Pro. $|\mathcal{DM}|$ / $|\mathcal{R}|$ / $|\mathcal{C}|$ denotes the number of domains / relations / concepts in their KBs. $|\mathcal{R}_u|$ / $|\mathcal{C}_u|$ denotes the number of relations / concepts unseen in the source KBs. $|\mathcal{D}^{\text{test}}|$ and $|\mathcal{D}_u^{\text{test}}|$ denotes the numbers of test cases and test cases that involve unseen schema items, respectively.

5.2 Baselines

For WebQSP, GraphQ, GrailQA, and MetaQA, we mainly compare KB-Plugin with low-resourced PI methods including (1) few-shot program generation methods **Pangu** (Gu et al., 2023) and **KB-BINDER** (Li et al., 2023a); (2) few-shot data generation method **APS** (Li et al., 2023c); (3) program transfer method **ProgramTrans** (Cao et al., 2022b), where we adopt its results without fine-tuning on target KBs for fair comparison. In addition, we also provide the results of several representative supervised models for comparison.

For SoAyBench, we choose tool-using methods that were evaluated on it as baselines, including **DFSDT** (Qin et al., 2023) and **SoAy** (Wang et al., 2024). These methods solve questions by prompting LLMs to call Aminer APIs in specific orders via in-context learning. Their processes of determining the composition of APIs and filling in arguments for each API can also be viewed as program induction.

We provide detailed descriptions of all the baselines and our evaluation metrics in Appendix D.1.

5.3 Implementation Details

In experiments, we use Llama2-7B (Touvron et al., 2023) as the backbone LLM of KB-Plugin and set the rank r of LoRA to 16. The number of parameters of each plugin is consequently 40M, which is extremely lightweight. The aliases of schema items in KQA Pro are directly obtained from Wikidata. The number of generated source KBs is set to 16 to balance performance and training efficiency. The sampling number K in schema plugin learning is set to be 500, 500, 50, 100, 3000, and 1000 for KQA Pro, WebQSP, GraphQ, GrailQA, MetaQA, and SoAyBench, respectively, to limit the size of the constructed data for schema plugin learning. We use beam size 5 for all experiments. More details can be found in Appendix D.2.

Method	WebQSP	GraphQ	GrailQA Test	GrailQA Dev
Supervised				
QGG	74.0	-	36.7	-
BERT+Ranking	-	25.0	58.0	-
ArcaneQA	75.6	31.8	73.7	76.8
RnG-KBQA	75.6	-	74.4	76.9
Low-resourced				
ProgramTrans	53.8*	-	-	-
APS	51.1	-	57.7	62.1
KB-BINDER	53.2	39.5	56.0	-
Pangu	54.5	43.3	**62.7**	-
KB-Plugin	**57.2 / 61.1***	**49.5**	**62.7**	**65.0**
w/o schema plugin	41.0	42.8	-	57.5
w/ $m_{sc}^{S_0}$	48.0	37.9	-	51.0

Table 2: F1 results on WebQSP, GraphQ, and GrailQA. * means using oracle topic entities.

Method	1-hop	2-hop	3-hop
Supervised			
KV-Mem	96.2	82.7	48.9
PullNet	97.0	99.9	91.4
EmbedKGQA	97.5	98.8	94.8
TransferNet	97.5	100.0	100.0
Low-resourced			
KB-BINDER	93.5	99.6	96.4
KB-Plugin	**97.1**	**100.0**	**99.3**
w/o schema plugin	92.6	99.0	98.9
w/ $m_{sc}^{S_0}$	90.4	93.6	88.6

Table 3: Hit@1 results on MetaQA.

Method	Acc
DFSDT (gpt-3.5-turbo)	45.7
DFSDT (gpt-4)	59.7
SoAy (gpt-3.5-turbo)	67.7
SoAy (gpt-4)	88.7
KB-Plugin	**90.8**
w/o schema plugin	70.8
w/ $m_{sc}^{S_0}$	64.0

Table 4: Accuracy results on SoAyBench.

Dataset	Method	$\mathcal{D}_{\text{seen}}^{\text{test}}$	$\mathcal{D}_{\text{unseen}}^{\text{test}}$
WebQSP	KB-Plugin	**64.9**	**53.3**
	w/o schema plugin	47.6	37.6
	Gain	+17.4	+15.7
GraphQ	KB-Plugin	40.0*	**49.7**
	w/o schema plugin	**70.9***	42.2
	Gain	-30.9*	+7.5
GrailQA-dev	KB-Plugin	**69.0**	**64.8**
	w/o schema plugin	64.9	57.3
	Gain	+4.1	+7.5

Table 5: F1 Results of KB-Plugin with and without schema plugin. $\mathcal{D}_{\text{unseen}}^{\text{test}}$ and $\mathcal{D}_{\text{seen}}^{\text{test}}$ denote the sets of test cases that involve and do not involve schema items unseen in the source KBs, respectively. $*$ means the results may not be indicative since there are only 55 cases in $\mathcal{D}_{\text{seen}}^{\text{test}}$ of GraphQ.

5.4 Main Results

The results are presented in Table 2, 3 and 4. Compared with Pangu, the SoTA PI method for low-resourced KBs, KB-Plugin improves the F1 score by 2.7% and 6.2% on WebQSP and GraphQ, respectively, and achieves comparable performance on GrailQA, despite Pangu using 25× larger model (175B Codex) and 100 annotated examples from each dataset. Moreover, Pangu needs to call Codex hundreds of times for a question to score each candidate program, while our model selects the optimal program via beam search, which is significantly faster and less costly. Besides, since ProgramTrans, KB-BINDER, and Pangu all link questions to schema items according to their names only, the superiority of KB-Plugin also demonstrates the benefits of aggregating additional schema information from relevant triples via schema plugin learning. KB-Plugin even surpasses several supervised models on GraphQ and GrailQA, which demand training using thousands of annotated samples from target KBs, showing the effectiveness of transferring prior knowledge from rich-resourced KBs.

On MetaQA and SoAyBench, KB-Plugin outperforms all the low-resourced baselines even though they use more powerful LLMs (i.e., Codex, gpt-3.5-turbo, and gpt-4), indicating that our framework also performs well for domain-specific KBs. In particular, KB-Plugin achieves strong performance on par with supervised SoTAs on MetaQA even if it does not see any target relations from the source domain.

5.5 Ablation Study

To demonstrate the effect of schema plugins, we remove them from our framework, i.e., we directly train a PI plugin using the source domain data and transfer it to the target KBs without training any schema plugins. According to Table 2, 3, 4, and 5, the performance of KB-Plugin without schema plugins is severely degraded, especially on the test cases that involve schema items unseen in the source KBs. The experimental results illustrate that (1) direct PI transfer is difficult due to the substantial difference between the schemas of source and target KBs; (2) schema plugins of target KBs effectively encode adequate schema information

Question I	*Which airport to fly into Rome?*
Pangu	Find(Rome) Relate(tourist attractions) (✗)
KB-Plugin w/o schema plugin	Find(Rome) Relate(country) FilterConcept(sovereign state) (✗)
KB-Plugin	Find(Rome) Relate(transport terminus) FilterConcept(airport) (✓)
Relevant Triples	(London, transport terminus, Luton airport), (London, instance of, citytown), (Luton airport, instance of, airport)
Question II	*What role did Paul Mccartney play in the Beatles?*
Pangu	Find(Paul Mccartney) Relate(instruments played) (✗)
KB-Plugin	Find(Beatles) Relate(member) Find(Paul Mccartney) ReverseRelate(member) And() Relate(role) (✓)
Source Domain Data Pair	*What is Jane Lynch's role in Glee?* Find(Glee) Relate(starring) Find(Jane Lynch) ReverseRelate(starring) And() Relate(character role)

Table 6: Two typical questions from the test set of WebQSP that KB-Plugin succeeds while Pangu fails. The incorrect functions and arguments are marked as red, while the correct ones are marked as green.

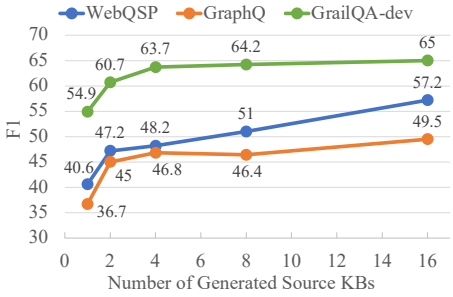

Figure 3: KB-Plugin performance with different numbers of generated source KBs.

via the triple completion task, and the PI plugin can extract and utilize question-relevant schema information from these schema plugins even though it is never trained with them. In addition, if we adopt the schema plugin of a source KB, e.g., $m_{sc}^{S_0}$, for the target KBs, the performance of KB-Plugin also drops heavily, showing the necessity of using matched schema plugin.

To show the rationality of our PI plugin learning method, we evaluate the performance of PI plugins trained with different numbers of generated source KBs on WebQSP, GraphQ, and GrailQA, and present the results in Fig. 3. The PI plugin trained with only one source KB performs poorly, implying that it ignores the schema plugin entirely and directly learns PI over this source KB. Once there emerges a new source KB with a different schema, the performance of the trained PI plugin increases substantially, and there is an apparent trend that the performance will increase with more generated source KBs. These results prove that training the PI plugin over multiple source KBs succeeds in forcing the PI plugin to learn to extract and utilize schema information from different schema plugins, and the learned skill can be transferred to target KBs.

5.6 Case Study

To better showcase the advantages of KB-Plugin over in-context learning PI methods, we present a case comparison between KB-Plugin and Pangu in Table 6. Question I shows the effect of schema plugin learning and utilization. Both Pangu and KB-Plugin without schema plugin struggle to predict the correct relation *"transport terminus"* because it is unseen in the demo examples or source KBs. The complete KB-Plugin, however, effectively encodes the information that *"transport terminus"* is a possible relation between *"citytown"* and *"airport"* into the schema plugin via completing relevant triples, and succeeds in predicting this relation by utilizing above information. Question II demonstrates the benefits of harnessing abundant program annotations from the source domain, where Pangu produces a program with incorrect function composition because none of its demo examples has a similar compositional structure, while KB-Plugin induces the correct program by utilizing prior knowledge learned from the source domain. Further analysis can be found in Appendix E and F.

6 Conclusion

We propose KB-Plugin, a plug-and-play framework that enables LLMs to induce programs over any low-resourced KBs by learning two types of pluggable modules: KB-specific schema plugin and KB-transferable PI plugin. KB-Plugin achieves better or comparable performance on five heterogeneous KBQA datasets with much smaller backbone LLMs compared to SoTA PI methods for low-resourced KBs, demonstrating its effectiveness for both large-scale and domain-specific KBs. Ablation study and case study also prove the rationality and further showcase the advantage of KB-plugin.

7 Limitations

We discuss several limitations of KB-Plugin in this section: (1) In the experiments, we only adopt Llama2-7B as our backbone model due to limited computing resources. Actually, KB-Plugin is model-agnostic and can also be applied to more language models with various sizes and architectures. (2) KB-Plugin requires that the source domain dataset covers questions with diverse various compositional structures, and performs relatively poorly for questions whose compositional structures are unseen in the source domain dataset though they are rare (see Appendix E for details). Future research can focus on improving the transferability of KB-Plugin across compositional structures. In practice, we can also continue to train the PI plugin using some self-training methods such as EGST (Li et al., 2023c) to adapt to these questions. (3) In this work, since both training and evaluation of KB-Plugin require annotated KBQA datasets, we can only take a single dataset KQA Pro as the source dataset and take other datasets as the target datasets, which may limit the upper bounds of KB-Plugin. In the realistic scenario where we need to apply KB-Plugin for a new KB, we can take all these KBQA datasets as the source domain datasets so that the trained source schema plugins would be more diverse and the trained PI plugin would also have stronger transferability and generalizability.

8 Ethical Considerations

Though our framework (as well as other PI methods) can effectively reduce the probability of LLMs generating inaccurate answers when faced with questions involving uncommon knowledge, it may still make mistakes if the induced programs are incorrect. In addition, there is a risk of being hacked through targeted means such as injecting harmful or nonfactual knowledge into the KBs. Hence additional care and protective measures should be taken if our framework is deployed in user-facing applications.

All the datasets and encyclopedias used in this work are publicly published with permissible licenses.

References

Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250. ACM.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2787–2795.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Shulin Cao, Jiaxin Shi, Liangming Pan, Lunyiu Nie, Yutong Xiang, Lei Hou, Juanzi Li, Bin He, and Hanwang Zhang. 2022a. KQA pro: A dataset with explicit compositional programs for complex question answering over knowledge base. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6101–6119. Association for Computational Linguistics.

Shulin Cao, Jiaxin Shi, Zijun Yao, Xin Lv, Jifan Yu, Lei Hou, Juanzi Li, Zhiyuan Liu, and Jinghui Xiao. 2022b. Program transfer for answering complex questions over knowledge bases. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 8128–8140. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan

Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Yu Gu, Xiang Deng, and Yu Su. 2023. Don't generate, discriminate: A proposal for grounding language models to real-world environments. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 4928–4949. Association for Computational Linguistics.

Yu Gu, Sue Kase, Michelle Vanni, Brian M. Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond I.I.D.: three levels of generalization for question answering on knowledge bases. In *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pages 3477–3488. ACM / IW3C2.

Yu Gu and Yu Su. 2022. Arcaneqa: Dynamic program induction and contextualized encoding for knowledge base question answering. In *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*, pages 1718–1731. International Committee on Computational Linguistics.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spacy: Industrial-strength natural language processing in python.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023. Structgpt: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 9237–9251. Association for Computational Linguistics.

Yunshi Lan and Jing Jiang. 2020. Query graph generation for answering multi-hop complex questions from knowledge bases. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 969–974. Association for Computational Linguistics.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 3045–3059. Association for Computational Linguistics.

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

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 4582–4597. Association for Computational Linguistics.

Xingxuan Li, Ruochen Zhao, Yew Ken Chia, Bosheng Ding, Lidong Bing, Shafiq R. Joty, and Soujanya Poria. 2023b. Chain of knowledge: A framework for grounding large language models with structured knowledge bases. *CoRR*, abs/2305.13269.

Zhenyu Li, Sunqi Fan, Yu Gu, Xiuxing Li, Zhichao Duan, Bowen Dong, Ning Liu, and Jianyong Wang. 2023c. Flexkbqa: A flexible llm-powered framework for few-shot knowledge base question answering. *CoRR*, abs/2308.12060.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2181–2187. AAAI Press.

Haoran Luo, Haihong E, Zichen Tang, Shiyao Peng, Yikai Guo, Wentai Zhang, Chenghao Ma, Guanting Dong, Meina Song, and Wei Lin. 2023. Chatkbqa: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models. *CoRR*, abs/2310.08975.

Xin Lv, Lei Hou, Juanzi Li, and Zhiyuan Liu. 2018. Differentiating concepts and instances for knowledge graph embedding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1971–1979. Association for Computational Linguistics.

Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1400–1409. The Association for Computational Linguistics.

Fedor Moiseev, Zhe Dong, Enrique Alfonseca, and Martin Jaggi. 2022. SKILL: structured knowledge infusion for large language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 1581–1588. Association for Computational Linguistics.

Lunyiu Nie, Shulin Cao, Jiaxin Shi, Jiuding Sun, Qi Tian, Lei Hou, Juanzi Li, and Jidong Zhai. 2022. Graphq IR: unifying the semantic parsing of graph query languages with one intermediate representation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 5848–5865. Association for Computational Linguistics.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2023. Unifying large language models and knowledge graphs: A roadmap. *CoRR*, abs/2306.08302.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 487–503. Association for Computational Linguistics.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *CoRR*, abs/2307.16789.

Yujia Qin, Xiaozhi Wang, YuSheng Su, Yankai Lin, Ning Ding, Zhiyuan Liu, Juanzi Li, Lei Hou, Peng Li, Maosong Sun, and Jie Zhou. 2021. Exploring low-dimensional intrinsic task subspace via prompt tuning. *CoRR*, abs/2110.07867.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. 2022. Sequence-to-sequence knowledge graph completion and question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 2814–2828. Association for Computational Linguistics.

Apoorv Saxena, Aditay Tripathi, and Partha P. Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4498–4507. Association for Computational Linguistics.

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. 2021. Transfernet: An effective and transparent framework for multi-hop question answering over relation graph. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 4149–4158. Association for Computational Linguistics.

Yu Su, Huan Sun, Brian M. Sadler, Mudhakar Srivatsa, Izzeddin Gur, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for QA evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 562–572. The Association for Computational Linguistics.

Yusheng Su, Xiaozhi Wang, Yujia Qin, Chi-Min Chan, Yankai Lin, Huadong Wang, Kaiyue Wen, Zhiyuan Liu, Peng Li, Juanzi Li, Lei Hou, Maosong Sun, and Jie Zhou. 2022. On transferability of prompt tuning for natural language processing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 3949–3969. Association for Computational Linguistics.

Haitian Sun, Tania Bedrax-Weiss, and William W. Cohen. 2019. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2380–2390. Association for Computational Linguistics.

Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 990–998. ACM.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Xiaozhi Wang, Kaiyue Wen, Zhengyan Zhang, Lei Hou, Zhiyuan Liu, and Juanzi Li. 2022. Finding skill

neurons in pre-trained transformer-based language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 11132–11152. Association for Computational Linguistics.

Yuanchun Wang, Jifan Yu, Zijun Yao, Jing Zhang, Yuyang Xie, Shangqing Tu, Huihui Yuan, Jingyao Zhang, Bowen Huang, Yuanyao Li, Juanzi Li, and Jie Tang. 2024. Soay: A service-oriented apis applying framework of large language models.

Chaojun Xiao, Zhengyan Zhang, Xu Han, Chi-Min Chan, Yankai Lin, Zhiyuan Liu, Xiangyang Li, Zhonghua Li, Zhao Cao, and Maosong Sun. 2023. Plug-and-play document modules for pre-trained models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 15713–15729. Association for Computational Linguistics.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. 2022. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 602–631. Association for Computational Linguistics.

Xuchen Yao. 2015. Lean question answering over freebase from scratch. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 66–70. The Association for Computational Linguistics.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 2022. RNG-KBQA: generation augmented iterative ranking for knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6032–6043. Association for Computational Linguistics.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*. The Association for Computer Linguistics.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6069–6076. AAAI Press.

Zhengyan Zhang, Zhiyuan Zeng, Yankai Lin, Huadong Wang, Deming Ye, Chaojun Xiao, Xu Han, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2023. Plug-and-play knowledge injection for pre-trained language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 10641–10658. Association for Computational Linguistics.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A survey of large language models. *CoRR*, abs/2303.18223.

Function	Input×Args → Output	Description
Find	$E \times \emptyset \rightarrow E$	find an entity from the KB
FindAll	$\emptyset \times \emptyset \rightarrow E'$	return all entities in the KB
Relate	$(E \cup E') \times R \rightarrow E'$	a single hop along a relation
ReverseRelate	$(E \cup E') \times R \rightarrow E'$	a reverse hop along a relation
FilterConcept	$E' \times C \rightarrow E'$	return entities in a concept
And/Or	$(E', E') \times \emptyset \rightarrow E'$	intersection/union of two sets
Argmax/Argmin	$E' \times R \rightarrow E'$	superlative aggregations
LT/LE/GT/GE	$E \times R \rightarrow E'$	$< / \leq / > / \geq$
Count	$E' \times \emptyset \rightarrow N$	set cardinality

Table 7: KoPL functions used in this work. E: entity; E': a set of entities; R: relation; C: concept; N: integer.

A Details of KoPL Functions

We list KoPL functions used in this work in Table 7. We make some modifications to the original (Cao et al., 2022a) for conciseness. Except `Find` taking topic entities as the argument, other functions either have no arguments or take schema items (i.e., concepts or relations) as their arguments.

B Triple Sampling Strategy

For WebQSP, GraphQ, and GrailQA, since their KBs are large-scale and relatively sparse, we adopt a popularity-based strategy to sample representative triples for each schema item. Specifically, let the given KB be $\mathcal{KB} = \{\mathcal{C}, \mathcal{E}, \mathcal{R}, \mathcal{T}\}$, where $\mathcal{T} = \mathcal{T}_e \cup \mathcal{T}_c \cup \mathcal{T}_l$. For each $e \in \mathcal{E}$, let $\mathrm{cnt}(e)$ be its popularity (i.e., the number of its occurrences in $\mathcal{KB}$). When sampling *"instance of"* triples for a concept $c \in \mathcal{C}$, we hope the sampled triples contain representative entities belonging to c, so we sort all $(e_k, \textit{instance of}, c) \in T_e$ in descending order of $\mathrm{cnt}(e_k)$ and select the first K triples. When sampling relational triples for a relation $r \in \mathcal{R}_l$, we take both representativeness and diversity into account. Therefore, we sort all $(e_i, r, e_j) \in T_l$ in descending order of $\min(\mathrm{cnt}(e_i), \mathrm{cnt}(e_j))$ and select the first K triples.

On the other hand, the KBs of KQA Pro, MetaQA, and SoAyBench are dense, so we just randomly sample triples for their schema items.

C Details of Constrained Decoding

In constrained decoding, after M_{PI}^T generates t function chunks $f_1(arg_1), \ldots, f_t(arg_t)$, we enumerate all admissible $f_{t+1}(arg_{t+1})$ as the candidate set P_{t+1} following the definition of KoPL functions in Table 7, and constrain M_{PI}^T to continue generating one of these candidate or generating the ⟨`EOS`⟩ token to end the decoding process.

Specifically, let E_{topic} be the set of topic entities in the question obtained using off-the-shelf entity linkers [1]. At $t = 0$, we enumerate `Find`(e) for each $e \in E_{\text{topic}}$ as a candidate in P_1. Specially, around 5% of questions in GraphQ and GrailQA do not have a topic entity (e.g., "Who is the heaviest film director?" from GrailQA, whose target program is `FindAll()` `FilterConcept`(*director*)`SelectAmong`(*weight kg*). For these questions, we follow Pangu (Gu et al., 2023) to start constrained decoding from `FindAll()FilterConcept`(c), where c is a topic concept provided by Gu and Su (2022).

When $t > 0$, we execute the current program $p_t = \langle f_1(arg_1), \ldots, f_t(arg_t) \rangle$ to get its denotation (i.e., a set of entities) and also the concepts, forward relations, and backward relations that are reachable from the denotation. For each concept c, we enumerate `FilterConcept`(c) as a candidate in P_{t+1}. For each forward relation r, we enumerate `Relate`(r) as a candidate. For each backward relation r, we enumerate `ReverseRelate`(r) as a candidate, and also include `LT`(r), `LE`(r), `GT`(r), and `GE`(r) in P_{t+1} if the denotation of p_t is a numerical value such as a quantity or a date. In addition, candidates with superlatives can be enumerated as `Argmax`(r) and `Argmin`(r). Also, `Count()` can always be included to P_{t+1}. If there are multiple topic entities, we enumerate `Find`(e') as a candidate to add a new branch, where $e' \in E_{\text{topic}}$ is a topic entity not in p_t. When p_t contains multiple branches, we enumerate `Or()` and `And()` as candidates to merge the last two branches.

D Experimental Setup

D.1 Details of Baselines and Evaluation Metrics

The details of our baselines are as follows:

Pangu (Gu et al., 2023) utilizes potent LLM Codex (Chen et al., 2021) to produce programs in a step-wise fashion via in-context learning. At each step, it first extends existing programs into new valid candidates by enumerating all possible next functions with arguments, then scores each candidate using Codex with several demonstrations and retains the top-k candidates.

KB-BINDER (Li et al., 2023a) first lets Codex generate several "draft" programs for a given question by imitating a few examples, then grounds the arguments in the drafts to the target KB using similarity

[1] Entity linking is not a major challenge for PI, and exhaustive fuzzy string matching (Yao, 2015) suffices to achieve a reasonable performance.

search to produce hundreds of refined programs. The final answer is decided by the majority vote after executing all these refined programs.

Automatic Program Sampling (APS) (Li et al., 2023c) utilizes gpt-3.5-turbo[2] to translate automatically sampled programs based on a handful of templates into corresponding questions via in-context learning, and subsequently fine-tune a RnG-KBQA (Ye et al., 2022) PI model using the generated question-program pairs.

ProgramTrans (Cao et al., 2022b) is a program transfer method that first uses a seq2seq sketch parser to translate the question into a program sketch, then uses an argument parser to search suitable argument from the KB for each function. We adopt its results without fine-tuning on the target KBs for fair comparison.

DFSDT (Qin et al., 2023) is the SoTA method for general tool using. To solve a question, it employs an LLM to call suitable tool APIs in depth-first order. At each step, the LLM can either (1) call the next API to proceed along a promising path or (2) undo the current call and call another API to expand a new path.

SoAy (Wang et al., 2024) is the SoTA method on SoAyBench. Given a question, it employs LLM to first select the most suitable plan (i.e., API combination) from a candidate pool, then write a Python program with branching and looping structure following the plan to call APIs to get the answer.

Supervised Methods. For WebQSP, GraphQ, GrailQA, and MetaQA, we also provide the fully supervised results of several representative models for comparison, including QGG (Lan and Jiang, 2020), BERT+Ranking (Gu et al., 2021), ArcnaeQA (Gu and Su, 2022), RnG-KBQA (Ye et al., 2022), KV-Mem(Miller et al., 2016), PullNet (Sun et al., 2019), EmbedKGQA (Saxena et al., 2020) and TransferNet Shi et al. (2021).

Evalution Metrics. Following these baselines, we use F1 for WebQSP, GraphQ, and GrailQA, use Hit@1 for MetaQA, and use Accuracy for SoAyBench.

D.2 Implementation Details

We train the schema plugins of the source and target KBs for 3 epochs and 1 epoch, respectively. The batch size and learning rate are set to be 128 and 1e-5, respectively. Besides, we train the PI plugin for 1 epoch with batch size 16 and learning rate 1e-5. For WebQSP, GraphQ, and GrailQA, we use the same off-the-shelf entity-linker as Pangu to find topic entities; For MetaQA, we follow our baselines to use oracle topic entities; For SoAyBench, we find topic entities using spaCy (Honnibal et al., 2020).

[2]https://platform.openai.com/docs/models/gpt-3-5

Dataset	Seen			Unseen		
	Num	EM	F1	Num	EM	F1
GraphQ	2148	71.0	52.8	247	15.4	20.4
GrailQA	6433	79.9	67.4	330	10.0	16.4

Table 8: Performance of KB-Plugin on test cases whose compositional structures are seen and unseen in the source dataset KQA Pro. EM means the exact match of program sketch.

E Analysis about Question Compositional Structures

For GraphQ and GrailQA, we translate their SPARQL programs to KoPL programs using GraphQ Trans (Nie et al., 2022) and analyze the performance of KB-Plugin on the test cases whose question compositional structures (identified by program sketches) are seen and unseen in the source domain dataset KQA Pro, respectively. From the results in Table 8 we can see that (1) KQA Pro covers most of question compositional structures in the target dataset; (2) KB-Plugin correctly predicts the program sketches for over 70% questions whose compositional structures are seen in the source domain dataset, implying that the mapping from questions to program sketches is largely independent of KB schemas and transferable across KBs, which is consistent with the findings of Cao et al. (2022b) and Li et al. (2023a); (3) KB-Plugin performs poorly on the questions with unseen compositional structures though they are relatively rare, indicating that more advanced transfer techniques across compositional structures remains to be explored.

F Error Analysis

We analyze 100 incorrect predictions (i.e., F1<1) randomly sampled from the dev set of GrailQA. The major errors are predicting wrong schema items (36%). Specially, when facing several schema items with only subtle differences, e.g., *"publisher"*(reverse) v.s. *"game version published"*, KB-plugin tends to prefer to choose the shorter one due to the inherent defects of beam search. Besides, 21% errors are due to a wrong termination check where the model misses the last relation or

predicts an additional function. There are also 5% wrong function predictions. Apart from the above errors caused by our model, 27% errors are caused by unidentified or wrongly identified topic entities during entity linking, 9% errors are due to ambiguous or wrong annotations, and the remaining 2% errors are due to the incompletion of KBs.