Few-shot Transfer Learning for Knowledge Base Question Answering: Fusing Supervised Models with In-Context Learning

Mayur Patidar[†], **Riya Sawhney**[‡], **Avinash Singh**[†], **Biswajit Chatterjee**[†],
Mausam[‡], **Indrajit Bhattacharya**[†]

[†]TCS Research, [‡]Indian Institute of Technology, Delhi
{patidar.mayur, singh.avinash9, chatterjee.biswajit1, b.indrajit} @tcs.com
riya.sawhney@outlook.com, mausam@cse.iitd.ac.in

Abstract

Existing Knowledge Base Question Answering (KBQA) architectures are hungry for annotated data, which make them costly and time-consuming to deploy. We introduce the problem of few-shot transfer learning for KBQA, where the target domain offers only a few labeled examples, but a large labeled training dataset is available in a source domain. We propose a novel KBQA architecture called FuSIC-KBQA that performs KB-retrieval using multiple source-trained retrievers, re-ranks using an LLM and uses this as input for LLM few-shot in-context learning to generate logical forms These are further refined using execution-guided feedback. Experiments over multiple source-target KBQA pairs of varying complexity show that FuSIC-KBQA significantly outperforms adaptations of SoTA KBQA models for this setting. Additional experiments show that FuSIC-KBQA also outperforms SoTA KBQA models in the in-domain setting when training data is limited.

1 Introduction

A Knowledge-base Question Answering (KBQA) system converts a natural language question to a logical query, which is executed on the KB to return the answer. The prevalence of diverse KBs, in open domain as well as in enterprises, makes KBQA a task of great practical significance. Unfortunately, traditional supervised models (Saxena et al., 2022; Zhang et al., 2022; Mitra et al., 2022; Wang et al., 2022; Das et al., 2022; Ye et al., 2022; Chen et al., 2021; Das et al., 2021; Shu et al., 2022; Gu et al., 2023) and even recent LLM-based few-shot in-context learning (FS-ICL) architectures (Li et al., 2023; Nie et al., 2024) require large volumes of labeled (question, logical form) pairs for model training or for exemplar selection. Such data is costly and time-consuming to annotate in real applications, and this remains the biggest hurdle for rapid and low-cost deployment of KBQA systems.

A more realistic scenario for a new KB is that a domain expert provides a small number of questions and their logical forms as labels. In this paper, we ask whether this small data, in conjunction with a readily available and larger open-domain benchmark KBQA dataset (Gu et al., 2021; Yih et al., 2016; Su et al., 2016) seen as a source domain, can be used to bootstrap a KBQA system for the new KB, which forms the target domain. We propose this as the *few-shot transfer learning* setting for KBQA. To the best of our knowledge, this practically useful setting has not been studied in the KBQA literature. It differs from unsupervised transfer learning (Cao et al., 2022; Ravishankar et al., 2022), which assumes a large volume of unlabeled training questions in the target domain.

We first note that source and target tasks can differ along multiple dimensions. First, the two KBs may not have much overlap in schema or data. Additionally, the distributions of logical forms and natural language questions may also be quite different. This makes it extremely challenging to transfer source-trained models to a target KBQA system, with limited target supervision.

To handle this challenge, we resort to modern day LLMs, which have achieved unprecedented success in language understanding and code generation tasks, with only few-shot in-context learning (FS-ICL). We present FuSIC-KBQA (**Fu**sed **S**upervised and **I**n-**C**ontext Learning for **KBQA**), which uses source-trained models to construct LLM prompts for FS-ICL. In particular, given a target question, multiple source-trained retrievers with complementary strengths retrieve potentially relevant candidate paths, relations and entity types from the KB. The LLM then performs re-ranking of this retrieved information to increase their target relevance. The re-ranked retrieval is provided in the FS-ICL prompt of another LLM call to generate logical forms. To minimize syntax errors, FuSIC-KBQA generates logical forms in a common lan-

guage (SPARQL) instead of niche languages used in KBQA (e.g. s-expressions). To minimize semantic errors, FuSIC-KBQA provides execution-guided feedback to the LLM based on whether the logical form on execution returns an empty answer.

We perform extensive experiments with multiple pairs of source-target datasets of varying complexity. We show that FuSIC-KBQA outperforms adaptations of SoTA supervised models and recent few-shot LLM FS-ICL models for KBQA by large margins. Additionally, we find that even for in-domain KBQA (non-transfer) with limited training data, FuSIC-KBQA outperforms SoTA supervised KBQA models, suggesting that it is robust across various low-data settings.

Our specific contributions are as follows. (a) We propose the important task of few-shot transfer learning for KBQA. (b) We develop a novel KBQA architecture that combines the benefits of supervised retrievers and LLM prompting for generation and refinement. (c) We carefully create multiple KBQA transfer datasets of varying complexity, which we publicly release to aid future research[1]. (d) We demonstrate using extensive experiments that our fusion architecture has strong performance for few-shot transfer. (e) We additionally demonstrate benefits of this architecture for the in-domain KBQA setting with limited training data.

2 Related Work

In-domain supervised KBQA (Ye et al., 2022; Chen et al., 2021; Shu et al., 2022; Gu et al., 2023; Faldu et al., 2024) involves only a single domain with a large volume of in-domain labeled examples, which are used to train or fine-tune LLM-based architectures. Almost all SoTA approaches for this problem use a retrieve-then-generate architecture, where the first stage retrieves KB elements relevant to a question, and then provides these as context to a generation stage that uses an LLM such as T5 to generate the logical form. Pangu (Gu et al., 2023) is different in that it fine-tunes LLMs to discriminate between relevant logical forms, and interleaves retrieval and discrimination step by step. RetinaQA (Faldu et al., 2024) is a contemporaneous KBQA model for answerable as well as unanswerable questions (Patidar et al., 2023) that combines traversal-based KB retrieval with sketch-filling based construction. It also achieves SoTA

performance for in-domain supervised KBQA.

More recent in-domain KBQA approaches use few-shot in-context learning (FS-ICL) with LLMs. These are still dependent on large volumes of in-domain training data for selecting the most relevant few-shots. Different from retrieve-then-generate, KB-Binder (Li et al., 2023) and KB-Coder (Nie et al., 2024) follow a generate-then-ground architecture that first generates KB-independent partial programs using FS-ICL, and then grounds these for the specific KB using unsupervised techniques. Crucially, these do not take any advantage of the labeled in-domain examples using supervised learning. In recent work, Shu and Yu (2024) use TIARA's supervised retriever followed by FS-ICL LLM generation. We improve over this basic architecture in multiple ways – by accommodating multiple retrievers, re-ranking retrieval results and including execution-based feedback.

Unsupervised Transfer Learning for KBQA (Cao et al., 2022; Ravishankar et al., 2022) involves large volumes of labeled training data in the source domain, and additionally a large volume of *unlabeled data* in the target domain. SoTA models for this setting follow the generate-then-ground architecture, again by first generating KB-independent (partial) programs, which are then adapted for the target KB using unlabeled examples. These models do not look to benefit from FS-ICL using LLMs.

LLM prompting techniques related to ours have been used outside of KBQA. Re-ranking of output from supervised or unsupervised retrieval models using LLMs has recently been used for information extraction (Ma et al., 2023) and multi-hop question answering over documents (Khalifa et al., 2023). We are not aware of the use of LLMs for retrieval re-ranking for KBQA or for combining rankings from multiple retrievers. Reflexion (Shinn et al., 2023) is a recent RL framework that transforms environment signals to language-based feedback for an LLM. It explores python function generation as one of the use-cases, where output accuracy on generated test-cases is fed back to the LLM for improvement. Our execution-guided feedback can be imagined as a specific instance of this framework, but we are not aware of any use of similar ideas for generating logical forms for KBs.

3 Problem and Proposed Approach

We begin by defining the few-shot transfer learning setting for Knowledge Base Question Answering

(KBQA), and then present our solution for it.

Few-shot Transfer Learning for KBQA: A Knowledge Base G consists of a schema with data stored under it. The schema consists of entity types T and binary relations R defined over pairs of types. The data consists of entities E as instances of types T, and triples or facts $F \subseteq E \times R \times E$. We are given a *target Knowledge Base* G^t (consisting of entity types T^t, relations R^t, entities E^t and facts F^t) and a natural language question q^t, and the goal is to generate a structured query or a logical form l^t, which when executed over G^t returns a non-empty answer A^t for the question q^t.

In few-shot transfer learning, we are provided with *target few-shots* D^t containing tens of labeled training examples of questions and logical forms in the target domain. In addition, a related source domain has a *source knowledge base* G^s consisting of its own types T^s, relations R^s, entities E^s and facts F^s, and a much larger *source training set* D^s containing thousands of labeled training examples of questions and corresponding logical forms. A special case of this setting is *zero-shot transfer learning*, where $D^t = \{\}$.

Proposed Approach: *Retrieve-then-generate* architectures for KBQA – used by state-of-the-art models for the supervised in-domain setting – first retrieve relevant KB-elements for the query and then use these as context for generating the logical form. The KB-retrievers used in such models have steadily grown in sophistication in recent years. On the other hand, *generate-then-ground* models for KBQA – used for unsupervised transfer learning and in domain few-shot learning with LLMs – first generate KB-independent incomplete or draft logical forms based on the question, and then ground these for the specific KB.

For our solution, we assume that we have access to a Black-box Large Language Model. We use this term to refer to 100B+ parameter LLMs such as GPT-3 and beyond, which cannot be trained or fine-tuned for all practical purposes, and only offer access through prompting and in-context learning. In the rest of this paper, when we refer to LLMs, unless otherwise specified, we mean such models.

We begin by asking: *which of the two KBQA architectures above is better suited to simultaneously take advantage of a large source training set and an LLM to accurately generate logical forms with limited target supervision?* LLMs excel at

understanding and generating language, but lack knowledge of the target KB. Therefore, we hypothesize that an LLM can generate complete logical forms accurately using in-context exemplars from the target, when provided with high quality context retrieved from the target KB. High quality retrieval can be achieved using one or more sophisticated supervised retrievers – trained using the large source training set, and optionally adapted for the target using few-shot fine-tuning – and then re-ranking for the target using LLM prompting. Based on this hypothesis, we propose FuSIC-KBQA, a retrieve-then-generate architecture for the few-shot transfer learning problem for KBQA. It is depicted at a high-level in Fig. 1.

Retrieval Stage: The retrieval stage makes use of one or more off-the-shelf KB-retrievers. A KB-retriever takes as input a question q and a KB G and returns 3 different aspects: (i) top-K data-paths starting from the linked entities in q, (ii) top-K entity types, and (iii) top-K relations. Not all retrievers may return all three aspects, and K may be aspect-specific for a retriever. We additionally assume that the retrievers are supervised and allow training and fine-tuning. The retrievers of SoTA KBQA models such as TIARA (Shu et al., 2022), RnG-KBQA (Ye et al., 2022) and ReTRaCK (Chen et al., 2021) satisfy these requirements. Though Pangu (Gu et al., 2023) and RetinaQA (Faldu et al., 2024) do not have explicit retrievers, these can be adapted as such. More details on TIARA and Pangu retrievers are in Sec. A.1.1.

We train each retriever on source data D^s and fine-tune using target few-shots D^t. For a target test question, FuSIC-KBQA first performs retrieval using each retriever to get top-K results for each aspect. Note that K may be aspect-specific. Next, the target relevance is improved by *retrieval re-ranking* and sub-selecting using LLM prompting. Specifically, top-K results for each aspect are provided to the LLM along with the question, and it is prompted to select and rank the top-k results ($k \ll K$). Note that k may also be aspect-specific. Aspect-wise top-k retrieval from each retriever is separately passed to the generation stage. Specific prompts are described in Sec. A.1.2.

Generation Stage: We use LLM prompting to generate target logical forms. The prompt includes (i) the test question, (ii) the retrieved KB context for each aspect of each retriever, (iii) an instruc-
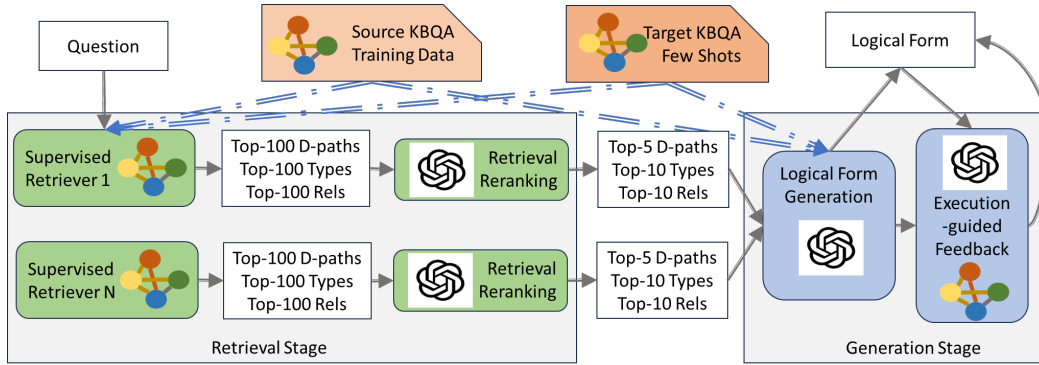
Figure 1: Architecture of FuSIC-KBQA, which fuses supervised learning with LLM few-shot in-context learning for KBQA. Starting from the question, the Retrieval Stage retrieves KB elements using one or more supervised KB retrievers and re-ranks these using LLM prompting. The subsequent Generation Stage uses LLM prompting with the retrieved KB context and labeled exemplars to generate the logical form, which is refined again with LLM prompting, using feedback from execution over the KB. The question is an input to all the components. Blue arrows with compound lines show flow of source training data and target few shots to components.

tion to generate the logical form, and (iv) few-shot exemplars from the target. For zero-shot, target exemplars are replaced with randomly selected exemplars from the source training set D^s. Example prompts are included in Sec. A.1.3. Different retrievers are not only allowed to have different retrieval sizes (k) for the same aspect, but also different output structures. For example, one retriever may return relations while another may not. Including complete retrieval outputs from multiple retrievers in the generation prompt may not be possible due to length constraints when using few-shots, since the retrieval is included for the exemplars as well. While optimal merging is a challenging problem, in this work, we linearly decrease the retrieval size k for aspects of individual retrievers, as the number of retrievers increases. Sec. A.1.3 has more details.

Even with few-shot exemplars and high quality context from multiple retrievers, LLM-generated logical forms contain syntactic and semantic errors, because of unfamiliarity with the target questions and logical forms. Choice of the representation language significantly impacts syntactic correctness of the generated logical form. So far, s-expressions have been used for LLM-based generation (Li et al., 2023; Nie et al., 2024; Shu and Yu, 2024). We expect any LLM to be significantly more familiar with SPARQL than niche languages such as s-expression, and therefore be better able to generate syntactically correct programs. We therefore use SPARQL. SoTA KBQA models represent retrieved KB data paths also as programs. For representational consistency, we use SPARQL to represent

the retrieved KB data paths as well.

Aside from syntactic errors, the generated logical form may have KB-specific semantic errors. Supervised KBQA models commonly use execution-guided checks to filter the ranked list of generated logical forms (Shu et al., 2022; Gu et al., 2023; Faldu et al., 2024) by checking if a logical form upon execution returns an empty answer. Instead, we use LLM-based *execution-guided refinement*. Specifically, if an empty answer is obtained on executing the generated logical form, this feedback is verbally provided to the LLM along with the erroneous logical form, prompting it to correct itself. This is continued until a non-empty answer is obtained, or for a maximum number of iterations, each time providing the sequence of previous generations as context. Examples of the specific prompt are in Sec. A.1.4.

4 Experiments

We now present experimental evaluation of FuSIC-KBQA for few-shot transfer learning which is our main focus. We also evaluate its usefulness for the limited training data in-domain KBQA setting.

4.1 Experimental Set-up

Datasets: A crucial aspect is creating source and target KBQA task pairs that are challenging for few-shot transfer in different ways. Source and target KBQA tasks can differ along multiple dimensions: (i) KB schema and data, (ii) distribution of logical forms, and (iii) distribution of natural language questions. An additional source-independent dimension is (iv) the generalization scenario for

Source→Target	Kn. Base			Logical Forms		NL Questions		Target Gen.
	Dom. JS	Rel. % New	Func. JS	# Relations Src Avg/Tgt Avg	Src-Tgt Sim Cosine	# Tokens Src Avg/Tgt Avg		Scenario
GrailQA→GraphQA	0.25	50.4	0.01	1.36 / 1.53	0.44	10.4 / 9.3		C,Z
GrailQA→WebQSP	0.37	55.4	0.04	1.35 / 1.54	0.31	10.4 / 6.6		I,C,Z
WebQSP→GrailQA-Tech	0.68	99.3	0.11	1.52 / 1.46	0.23	6.6 / 10.9		I,C,Z
WebQSP→GraphQA-Pop	0.66	97.5	0.07	1.52 / 1.73	0.23	6.6 / 9.2		C,Z

Table 1: Statistics for different source and target (test set) KBQA task pairs in terms of the knowledge base, logical forms and natural language questions. 'Dom.(JS)' is Jensen Shannon (JS) divergence between domain distribution of questions, 'Rel.(% New)' is percentage of questions in target with new (unseen) relations, 'Func.(JS)' is JS-divergence between distributions over functions in logical forms, '# Relations' shows the source average and the target average for number of relations per logical form, 'Src-Tgt Sim' is average minimum cosine distance between source and target questions and '# Tokens' shows the source average and target average of number of tokens per question. For the generalization setting of the target test set wrt target few shots, I,C and Z denote IID, compositional generalization and zero-shot respectively.

the target test set versus the target few shots. We prepare source→target pairs using the benchmark KBQA datasets GrailQA (Gu et al., 2021), WebQSP (Yih et al., 2016) and GraphQA (Su et al., 2016). Though all of these have Freebase as the backbone KB, there is large divergence between the domains, relations and types that these span. GrailQA and GraphQA were constructed similarly by first sampling meaningful KB paths and then verbalizing these into questions. However, the GrailQA test set contains largely IID (seen schema elements) and some compositional (unseen combination of seen schema elements) and zero-shot questions (unseen schema elements), while GraphQA test questions are all compositional and therefore harder. WebQSP questions are popular web queries posed by real users, and are therefore significantly different from GrailQA and GraphQA questions. However, all its test questions are IID.

Using these benchmark datasets, we created four source→target pairs in two ways. First, using entire benchmark datasets as sources and targets, we created GrailQA→GraphQA and GrailQA→WebQSP, considering GrailQA's large training dataset size (>44K). Then we took WebQSP as the source and selected as two targets *subsets* of GrailQA domains (in the technical category) and GraphQA domains (in the popular category) that are minimally represented in the WebQSP training data. We name these targets GrailQA-Tech and GraphQA-Pop. Since many functions are not present in WebQSP logical forms, WebQSP→GrailQA-Tech and WebQSP→GraphQA-Pop present significant transfer challenges. Tab. 1 records some key statistics of the four source→target pairs. We can see that while all datasets pose challenges for few-shot transfer,

overall difficulty increases from top to bottom. Because of cost of experimentation with GPT-4, for each target, we create the target test sets by sampling 500 instances uniformly at random from the original test sets (dev set for GrailQA) restricted to selected domains as necessary. 100 target few-shots are similarly sampled from the corresponding training sets. More details about dataset construction are in Sec. A.4.

LLM and Supervised Retrievers: We use `gpt-4-0613` as the LLM, since recent experiments have demonstrated GPT-4's significant superiority across a wide range of NLP tasks (Bubeck et al., 2023). As supervised retrievers, we experiment with TIARA and Pangu. TIARA (Shu et al., 2022) has the most general and sophisticated retriever among SoTA supervised KBQA models. Its retriever generalizes that of RnG-KBQA (Ye et al., 2022) and ReTRaCK (Chen et al., 2021) and returns data paths, types and relations. Though Pangu (Gu et al., 2023) does not have an explicit retriever and generates its logical form by incrementally selecting relations, we adapt it as a retriever by taking its top-5 paths at the final stage and extracting relations and types from these. We use TIARA's entity linker (trained on GrailQA) as our pre-trained off-the-shelf entity-linker for all experiments. We equip all retrieve-then-generate baselines with this same entity linker. KB-Binder's generate-then-ground architecture uses its own LLM-based entity linker.

Baselines: Since there are no existing few-shot transfer KBQA models, we adapt SoTA KBQA models designed for the in-domain setting from the purely supervised and LLM FS-ICL categories.

Supervised KBQA models: Considering the GrailQA leader-board and availability of code, we use **Pangu** (Gu et al., 2023) and **TIARA** (Shu et al., 2022) as supervised KBQA models. We use publicly available code, and pre-trained check-points for our sources - GrailQA and WebQSP. We additionally experiment with **BERT-Ranking**, which has been shown to work well in a different transfer setting for KBQA (non few-shot, with large volumes of target training data) (Gu et al., 2021; Shu et al., 2022) and outperform TIARA (Shu and Yu, 2024). We fine-tune all trainable components of these models on target few-shots for 1 epoch. These models are customized for generating logical forms in s-expression, and we preserve this in our experiments. Sec. A.2 has more details.

LLM Few-shot ICL KBQA models: We use **KB-Binder** (Li et al., 2023) as the SoTA model in this category. In the retrieval (-R) setting, KB-Binder uses BM25 to retrieve the demonstration samples from the entire available training data. We restrict its retrieval to the target few-shots in the few-shot setting and source training data in the zero-shot setting. We use self-consistency and majority voting with 6 examples, as in the paper. Additionally, we experiment with a recent retrieve-then-generate LLM FS-ICL model (Shu and Yu, 2024) which we name as **gf-LLM** (ground-first LLM). As in the paper, we keep TIARA as its retriever, but pre-trained on the source and fine-tuned on target few-shots as in FuSIC-KBQA. The in-context examples are provided in the same way as KB-Binder. KB-Binder and gf-LLM report experiments using `code-davinci-002` and `gpt-3.5-turbo-0613` respectively. We replace these with `gpt-4-0613` as in FuSIC-KBQA. Like Pangu and TIARA, these models also generate logical forms in s-expression, which we preserve. Note that unlike KB-Binder, FuSIC-KBQA and gf-LLM include KB-retrieval in their generation prompts for the test question and the exemplars, requiring many more tokens per exemplar. For all three models, we provide the maximum number of exemplars that fit in the prompt – 100 for KB-Binder, and 5 for FuSIC-KBQA and gf-LLM. Sec. A.3 has more details.

4.2 Results for Few-shot Transfer Setting

For the few-shot transfer learning, we first compare FuSIC-KBQA with SOTA baselines and then analyze different aspects of the model.

4.2.1 Model Comparison

Performances of various models for the few-shot transfer setting are recorded in Tab. 2. Because of mismatch between logical form language of some models and the gold-standards, we directly evaluate the answers obtained upon logical form execution using F1 (Gu et al., 2023; Li et al., 2023).

We first make some key observations about the performance of FuSIC-KBQA. **(a)** FuSIC-KBQA performs the best by a large margin in all datasets for both few-shot and zero-shot. **(b)** Performance varies with the retriever – FuSIC-KBQA with TIARA is mostly better than FuSIC-KBQA with Pangu, but using the two together is always better than both individually, showing the complementary nature of their retrieval and FuSIC-KBQA's ability to take advantage of this. **(c)** While performance improves with few-shots, it also performs remarkably well in the zero-shot setting across datasets. **(d)** Performance varies for all models across datasets, as a reflection of their varying difficulty, and the fourth dataset is the hardest for almost all models. However, FuSIC-KBQA has the most stable performance across datasets.

Next, we analyze relative performance of the different models. **(a)** Performances of the supervised models degrade significantly as task complexity increases. While these, especially Pangu, perform very well for the first dataset, performance is significantly lower for the other three. This reflects the challenge of learning to generate logical forms accurately from only target few-shots when the source is less related. **(b)** Models equipped with LLM FS-ICL perform better than purely supervised models, particularly with few-shots. This demonstrates the generative ability of GPT-4 ICL for logical forms. That said, the naive LLM FS-ICL baseline, which simply prompts GPT-4 with a question without any KB retrieval and instructs it to generate the logical form along with 5 question and logical form exemplars, performs extremely poorly, e.g. achieving less than 5.0 F1 on WebQSP→GrailQATech. **(c)** The retrieve-then-generate approach of gf-LLM and FuSIC-KBQA works better than that of KB-Binder's generate-then-ground approach, demonstrating the strength of this architecture for few-shot transfer. **(d)** Within the retrieve-then-generate category, FuSIC-KBQA significantly outperforms gf-LLM, by accommodating multiple retrievers, using SPARQL and employing LLMs for retrieval re-ranking and execu-

	Model	GrailQA→GraphQA	GrailQA→WebQSP	WebQSP→GrailQA-Tech	WebQSP→GraphQA-Pop
Zero-shot Supervised	TIARA (Shu et al., 2022)	52.8	30.0	30.6	37.8
	Pangu (Gu et al., 2023)	60.9	39.3	40.8	25.1
LLM ICL	BERT-Ranking (Gu et al., 2021)	42.4	35.9	21.5	21.1
	KB-Binder (Li et al., 2023)	45.0	39.0	33.6	25.1
	gf-LLM (Shu and Yu, 2024)	53.4	31.3	42.8	38.1
Fusion	FuSIC-KBQA(T)	59.2	55.9	68.4	56.8
	FuSIC-KBQA(P)	63.8	52.5	65.1	41.3
	FuSIC-KBQA(T,P)	**67.5**	**56.6**	**71.4**	**61.7**
Few-shot Supervised	TIARA (Shu et al., 2022)	50.1	40.6	48.2	37.3
	Pangu (Gu et al., 2023)	63.2	45.3	51.0	36.7
LLM ICL	BERT-Ranking (Gu et al., 2021)	44.1	34.4	33.1	23.4
	KB-Binder (Li et al., 2023)	35.2	55.7	68.0	31.5
	gf-LLM (Shu and Yu, 2024)	52.5	53.4	61.2	46.6
Fusion	FuSIC-KBQA(T)	60.3	65.0	70.8	52.3
	FuSIC-KBQA(P)	64.5	61.9	70.3	49.1
	FuSIC-KBQA(T,P)	**69.1**	**65.1**	**74.6**	**61.6**

Table 2: Few-shot transfer learning performance (F1) of different models for different source→target pairs, evaluated over 500 randomly sampled instances over the test sets. FuSIC-KBQA(T), FuSIC-KBQA(P), FuSIC-KBQA(T,P) denote FuSIC-KBQA used with TIARA, Pangu, and TIARA and Pangu together as supervised retrievers.

	Model	Gl→Gp	Gl→W	W→Gl-T	W→Gp-P
Zero-shot	FuSIC-KBQA(T)	59.2	55.9	68.4	56.8
	w/o SPARQL	48.2	41.2	60.5	46.4
	w/o EGF	55.1	53.2	64.4	54.9
	w/o RR	58.0	48.0	59.5	52.6
Few-shot	FuSIC-KBQA(T)	60.3	65.0	70.8	52.3
	w/o SPARQL	53.0	58.7	64.3	53.9
	w/o EGF	56.5	62.0	66.6	44.0
	w/o RR	53.7	61.2	68.4	50.5

Table 3: Ablation of few-shot and zero-shot transfer learning performance using F1 for FuSIC-KBQA(T) for different source→ pairs. Gl, Gl-T, Gp, Gp-P and W denote GrailQA, GrailQA-Tech, GraphQA, GraphQA-Pop and WebQSP, respectively, for brevity. EGF and RR denote execution-guided feedback and retrieval reranking respectively. w/o SPARQL denotes use of s-expression instead of SPARQL for generation.

tion guided refinement in addition to generation.

4.2.2 Analysis of FuSIC-KBQA

Ablations: Tab. 3 records performance for different ablations of FuSIC-KBQA. We can see that the biggest gain comes from using SPARQL as the logical form language, except for the few-shot setting in the fourth dataset. Refinement using execution-guided feedback and reranking of retrieval results make more modest but significant contributions.

We performed additional ablation experiments on the WebQSP→GrailQATech dataset with FuSIC-KBQA(T,P), (i.e., with both TIARA and Pangu as retrievers). We first varied the number of EGF iterations. In the few-shot setting, performance without EGF is 71.6 F1. This improves to

74.4 and 74.6 after the first and second EGF iterations, but does not improve thereafter. In the zero-shot setting, in contrast, performance steadily improves from 66.7 F1 before EGF to 68.6, 70.8, 71.2 and 71.4 over four EGF iterations. This demonstrates that while each EGF iteration has associated cost, these may bring some performance benefits.

For the same WebQSP→GrailQATech dataset again using FuSIC-KBQA(T,P), we studied the effect of the number of few shots. Recall that each shot contributes significantly to prompt length. Unexpectedly, we observed F1 to be 73.6, 70.5, 72.3 and 74.6 for zero, 1, 3 and 5 shots respectively. The poorer performance of 1 and 3 shots can be attributed to low exemplar diversity, biasing GPT-4 to predict simpler logical forms. These suggests that FuSIC-KBQA could be used with fewer shots, but diversity of exemplars is important.

Error Analysis: We performed detailed error analysis of the best performing version of FuSIC-KBQA, that with TIARA and Pangu as retrievers, for the GrailQA→WebQSP and WebQSP→GrailQA-Tech datasets in the few-shot setting. For GrailQA→WebQSP, 41.6% of test questions have errors in answers (i.e., F1 < 1.0). Of these, 17.4% are due to entity linking, 9.8% due to retrieval and 14.4% due to generation. On the other hand, for WebQSP→GrailQA-Tech, there are fewer errors in total (31%), and the corresponding percentages are 10%, 0.4% and 21%, implying that entity linking errors are less, while all other errors are essentially generation errors – there are almost

no retrieval errors. Entity linking errors are caused roughly in equal proportion by errors in mention detection and entity disambiguation – 8.6% and 8.8% in the first dataset, and 4% and 6% in the second. Note that all models other than KB-Binder use the same entity linker. Retrieval errors can be of 3 types – (i) correct KB elements not in the top-K of the original retrieval, (ii) in the top-K but not in the top-k (with $k \ll K$) of the original retrieval, and (iii) originally in the top-k, but incorrectly demoted by LLM re-ranking. For the first dataset, which has significant retrieval errors, these contribute to 3.6%, 5.8% and 0.4% of errors – indicating that re-ranking errors are negligible. Finally, generation errors are potentially of two kinds – syntactic and semantic. However, for both datasets, all generation errors (14.4% and 21%) are semantic, with no syntactic errors at all.

	LLM	Gl→Gp	Gl→W	W→Gl-T	W→Gp-P
Z-s	GPT-4	67.5	56.6	71.4	61.7
	Mixtral	48.9	36.8	59.9	41.4
F-s	GPT-4	69.1	65.1	74.6	61.6
	Mixtral	54.0	48.2	59.0	34.2

Table 4: Performance of FuSIC-KBQA(T,P) with GPT-4 and Mixtral for the four datasets named as in Tab. 3.

Generalization for FuSIC-KBQA: In experiments so far, we have used GPT-4 as the state-of-the-art LLM for FuSIC-KBQA. Further, the source and target in all four datasets, in spite of their significant differences, have the same KB (Freebase) as the backbone. We now report some experiments with a different LLM and a different KB.

A major strength of FuSIC-KBQA is that it is a framework that allows plugging in different retrievers and also different LLMs. While we have demonstrated its superior performance using GPT-4, GPT-4 is expensive and proprietary. So we experimented with Mixtral 8x7B Instruct v0.1[2], which is a state-of-the-art open-source LLM with a large context length (32k), which is important for FuSIC-KBQA. Results are recorded in Tab. 4. Performance of FuSIC-KBQA(T,P) with Mixtral drops significantly compared to GPT-4, as expected. However, comparing with Tab. 2, we see that FuSIC-KBQA with Mixtral outperforms the supervised baselines for two out of the four datasets in both the few-shot and the zero-shot

settings. This demonstrates that FuSIC-KBQA holds promise even with significantly smaller open-source LLMs. More details are in Sec. A.10

	TIARA	gf-LLM	FuSIC-KBQA
GrailQA→MAKG	5.7	13.8	**45.6**

Table 5: Performance (F1) of FuSIC-KBQA, gf-LLM and TIARA in the few-shot setting with GrailQA as source and Microsoft Academic Graph (MAKG) as target. Since the complete KB schema fits in the prompt, FuSIC-KBQA does not use a retriever.

We also created a small new target dataset using the Microsoft Academic KG[3] This has real questions from students, which we manually annotated with corresponding SPARQL queries. We divided 70 questions into 5 training examples and 65 test questions. We used GrailQA as the source. The results are recorded in Tab. 5. We can see that FuSIC-KBQA outperforms TIARA and also gf-LLM by huge margins in this challenging setting. While this is a small experiment, this demonstrates the FuSIC-KBQA is likely to generalize well for different source and target backbone KBs. More details are in Sec. A.7.

#Train samples	Model	F1	EM
2,217 (5%)	TIARA	64.8	58.2
	Pangu	69.5	62.8
	RetinaQA	62.4	57.0
	FuSIC-KBQA(T)	70.6	60.4
	FuSIC-KBQA(P)	72.9	62.0
	FuSIC-KBQA(T,P)	**74.5**	**63.8**
4,434 (10%)	TIARA	67.2	61.2
	Pangu	72.4	64.6
	RetinaQA	68.3	62.4
	FuSIC-KBQA(T)	74.8	63.0
	FuSIC-KBQA(P)	74.1	63.0
	FuSIC-KBQA(T,P)	**75.6**	**65.2**
44,337 (100%)	TIARA	82.6	76.0
	Pangu	83.5	76.2
	RetinaQA	**84.7**	78.4
	FuSIC-KBQA(T)	79.1	70.2
	FuSIC-KBQA(P)	82.5	72.8
	FuSIC-KBQA(T,P)	83.6	75.0

Table 6: In-domain performance of different models on randomly sampled 500 questions from GrailQA dev set, when trained using different percentages of the GrailQA training set. TIARA, Pangu and RetinaQA generate s-expressions while FuSIC-KBQA generates SPARQL, so that EM is measured differently.

---

[2]https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

[3](https://makg.org/)

4.3 Results for In-domain Setting

Finally, we demonstrate the usefulness of FuSIC-KBQA in the in-domain setting when the volume of training data is limited. In addition to TIARA and Pangu, we also compare against RetinaQA (Faldu et al., 2024) – a contemporaneous model that has achieved SoTA performance for this setting. We train the supervised KBQA models and also the supervised components of FuSIC-KBQA on progressively smaller subsets of the GrailQA training data, and evaluate their performance on 500 randomly chosen instances from the GrailQA dev set.

Here, we additionally evaluate logical form accuracy using exact match (EM). Since the supervised models generate s-expressions, we evaluate EM automatically (Ye et al., 2022). FuSIC-KBQA generates SPARQL. Since automatic equivalence checking of SPARQL programs is challenging, we adopt a partly automated approach, where the checker outputs EQUIVALENT, NON-EQUIVALENT or NO-DECISION, and candidates in the last category are evaluated manually. A sample-based assessment on the test set showed a precision of 1.0 for the first two decisions. Further details are in Sec. A.9.

The results are recorded in Tab. 6. We can see that with 5% and 10% training data, FuSIC-KBQA(T) outperforms TIARA in terms of F1 and EM, FuSIC-KBQA(P) outperforms Pangu in terms of F1 but not EM, but FuSIC-KBQA(T,P) outperforms all the supervised models in terms of EM and also by a large margin in terms of F1. Even with 100% training data, FuSIC-KBQA(T,P) outperforms TIARA and Pangu in terms of F1. However, in this setting, RetinaQA is the best model overall in terms of both F1 and EM. It should be noted that in principle FuSIC-KBQA can be augmented with RetinaQA also as one of the retrievers just as with Pangu, and this could boost its performance. We leave this for future experiments.

In summary, this experiment demonstrates that FuSIC-KBQA can outperform supervised models in the in-domain setting given limited training data, and compares favorably even when large volumes of training data are available.

5 Discussion

Our exploration of LLM for KBQA led to many failed experiments. We discuss these briefly here.

Mention detection and entity disambiguation: We experimented with FS-ICL with GPT-4 for mention detection. However, this performed worse than the supervised version both in the transfer setting (by 1 pct point) and in the limited training data (5%) in-domain setting (by 10 pct points). For entity disambiguation, we explored GPT-4 re-ranking of the entities returned by the supervised model. However, this did not improve performance.

Self-consistency and majority voting: This is reported to improve performance for in-domain KBQA with chatGPT (Li et al., 2023; Nie et al., 2024). However, with GPT-4, generating with higher temperature led to little variance in the generated logical forms in our experiments, naturally rendering majority voting meaningless.

Exemplar selection: KB-Binder (Li et al., 2023) and gf-LLM (Shu and Yu, 2024) benefit from dynamic exemplar selection using BM25. However, we observed only minor improvements beyond random exemplar selection and static exemplars.

6 Conclusions

We have motivated and addressed the few-shot transfer learning setting for KBQA. We propose FuSIC-KBQA as a retrieve-then-generate architecture that benefits from KB-retrieval using one or more source-trained supervised retrievers, which are adapted for the target using LLM-based re-ranking. Such retrieval provides precise and relevant context to an LLM that generates logical forms with few-shot in-context learning, further refined by execution-guided feedback. We demonstrate the usefulness of FuSIC-KBQA using carefully designed experiments over multiple source→target settings of varying complexity. We show that FuSIC-KBQA significantly outperforms adaptations of SoTA KBQA models, both supervised and using LLM few-shot in-context learning, for few-shot transfer. We also demonstrate that FuSIC-KBQA outperforms SoTA supervised KBQA models in their own backyard – the in-domain setting – when the volume of training data is limited. An interesting future direction is addressing unanswerability of questions in the few-shot transfer setting (Patidar et al., 2023; Faldu et al., 2024).

Limitations

FuSIC-KBQA makes more LLM API calls compared to the LLM-based baselines. Both KB-Binder and gf-LLM make only one API call per test instance. FuSIC-KBQA makes 3 additional API calls for re-ranking the 3 different aspects of the retrieval result. Additionally, the genera-

tor makes 0 to 4 additional API calls per test instance for execution-guided feedback. As an example, the average number of EGF-related API calls per instance is 0.7 for GrailQA→WebQSP. So, in aggregate, FuSIC-KBQA makes 4.7 API calls on average for each instance. The number of API calls increases with additional retrievers. The goal of our exploration is to investigate the benefit of LLM API calls for KBQA while limiting the dependence on large volumes of labeled training instances, be it in-domain or few-shot transfer. At one extreme, we have LLM-free supervised only models. KB-Binder and gf-LLM demonstrate improvements based on 1 API call per instance. We have demonstrated that more benefit is obtainable with additional API calls.

While we have verified our individual experimental results by first performing experiments on smaller subsets, the numbers recorded in Tables 2, 3 and 6 are based on single runs for versions of FuSIC-KBQA as well as for KB-Binder and gf-LLM. While this understandably affects the reliability of the results, we are hampered by the cost of GPT-4.

Our main experiments are based on datasets built over Freebase as the underlying KB. Unfortunately, all existing KBQA datasets span only a small set of similar KBs, namely Freebase, Wikidata and DBPedia. Creating larger KBQA datasets spanning different types of KBs and domains is a substantial endeavour and we leave it for future work.

Risks

Our work does not carry any obvious risks.

Acknowledgements

References

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4.

Shulin Cao, Jiaxin Shi, Zijun Yao, Xin Lv, Jifan Yu, Lei Hou, Juanzi Li, Zhiyuan Liu, and Jinghui Xiao. 2022. 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)*.

Shuang Chen, Qian Liu, Zhiwei Yu, Chin-Yew Lin, Jian-Guang Lou, and Feng Jiang. 2021. ReTraCk: A flexible and efficient framework for knowledge base question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*.

Rajarshi Das, Ameya Godbole, Ankita Naik, Elliot Tower, Manzil Zaheer, Hannaneh Hajishirzi, Robin Jia, and Andrew Mccallum. 2022. Knowledge base question answering by case-based reasoning over subgraphs. In *Proceedings of the 39th International Conference on Machine Learning*.

Rajarshi Das, Manzil Zaheer, Dung Thai, Ameya Godbole, Ethan Perez, Jay Yoon Lee, Lizhen Tan, Lazaros Polymenakos, and Andrew McCallum. 2021. Case-based reasoning for natural language queries over knowledge bases. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Prayushi Faldu, Indrajit Bhattacharya, and Mausam. 2024. RETINAQA : A knowledge base question answering model robust to both answerable and unanswerable questions. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Bangkok, Thailand. Association for Computational Linguistics.

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)*, pages 4928–4949, Toronto, Canada. Association for Computational Linguistics.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond i.i.d.: Three levels of generalization for question answering on knowledge bases. In *Proceedings of the Web Conference 2021*, WWW '21.

Muhammad Khalifa, Lajanugen Logeswaran, Moontae Lee, Honglak Lee, and Lu Wang. 2023. Few-shot reranking for multi-hop QA via language model prompting. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15882–15897, Toronto, Canada. Association for Computational Linguistics.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023. Few-shot in-context learning on knowledge base question answering. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6966–6980, Toronto, Canada. Association for Computational Linguistics.

Yubo Ma, Yixin Cao, Yong Hong, and Aixin Sun. 2023. Large language model is not a good few-shot information extractor, but a good reranker for hard samples! In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10572–10601, Singapore. Association for Computational Linguistics.

Sayantan Mitra, Roshni Ramnani, and Shubhashis Sengupta. 2022. Constraint-based multi-hop question answering with knowledge graph. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*.

Zhijie Nie, Richong Zhang, Zhongyuan Wang, and Xudong Liu. 2024. Code-style in-context learning for knowledge-based question answering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):18833–18841.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Mayur Patidar, Prayushi Faldu, Avinash Singh, Lovekesh Vig, Indrajit Bhattacharya, and Mausam . 2023. Do I have the knowledge to answer? investigating answerability of knowledge base questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10341–10357, Toronto, Canada. Association for Computational Linguistics.

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. *Journal of Machine Learning Research*, 21(140):1–67.

Srinivas Ravishankar, Dung Thai, Ibrahim Abdelaziz, Nandana Mihindukulasooriya, Tahira Naseem, Pavan Kapanipathi, Gaetano Rossiello, and Achille Fokoue. 2022. A two-stage approach towards generalization in knowledge base question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2022*.

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)*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yiheng Shu and Zhiwei Yu. 2024. Distribution shifts are bottlenecks: Extensive evaluation for grounding language models to knowledge bases. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 71–88, St. Julian's, Malta. Association for Computational Linguistics.

Yiheng Shu, Zhiwei Yu, Yuhan Li, Börje Karlsson, Tingting Ma, Yuzhong Qu, and Chin-Yew Lin. 2022. TIARA: Multi-grained retrieval for robust question answering over large knowledge base. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*.

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gür, 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*, pages 562–572, Austin, Texas. Association for Computational Linguistics.

Yu Wang, Vijay Srinivasan, and Hongxia Jin. 2022. A new concept of knowledge based question answering (KBQA) system for multi-hop reasoning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. 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)*.

Wen-tau Yih, Matthew Richardson, Chris 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 (Volume 2: Short Papers)*.

Jing Zhang, Xiaokang Zhang, Jifan Yu, Jian Tang, Jie Tang, Cuiping Li, and Hong Chen. 2022. Subgraph retrieval enhanced model for multi-hop knowledge base question answering. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

A Appendix

A.1 FuSIC-KBQA Implementation Details

A.1.1 TIARA and Pangu Retrievers

TIARA (Shu et al., 2022) uses a multi-grain retriever, consisting of an entity retriever, a logical form retriever and a schema retriever.

The *entity retriever*, consists of a sequence of three steps - mention detection, candidate generation and entity disambiguation. Mention detection is performed using span classification, where candidate spans are represented using tokens via BERT, and classified as a mention or not. The candidate entities for each span are generated using FACC1 alias mapping. Entity disambiguation jointly encodes the question and an entity context (entity label and linked relations) in the KB using a cross encoder to get a matching score. The entity retriever components require large volumes of data to train. For every setting, we use a pre-trained entity retriever the GrailQA training set, and fine-tune it on the source using the target few-shots as the validation set. We use the same entity retriever for Pangu and BERT-Ranking.

The *exemplary logical form retriever* enumerates 2-hop paths in the KB from each candidate entity for the question and represents these using s-expression. Each s-expression are scored by concatenating with the question and encoding using BERT followed by a linear layer. The parameters are trained using a contrastive objective.

The *schema retriever* is independent of the entity retriever. It scores each entity type (class) and relation in the KB with respect to the question and returns the top entity types and top relations separately. Each entity type (or relation) is encoded along with the question using a cross-encoder similar to the s-expressions in the logical form retriever. Entity type and relation classifiers are trained separately.

If a retriever returns data paths in a different representation (e.g. in s-expression as for TIARA), we post-process these and translate to SPARQL using publicly available code.

A.1.2 Retrieval Re-ranking Prompts

We use three different prompts for re-ranking the data paths, entity types and relations in the retrieval result from a supervised retriever.

Re-ranking Data Paths: The prompt for re-ranking data paths uses the following template.

> For a given question please select five relevant candidate paths which might be useful for answering it in ranked order. If the number of candidates are five or less then just rerank the candidates. candidate paths are seperated by | symbol. The output should contain only relevant candidates seperated by "@@".
> **question:** what the language spoken in indonesia?
> **entity:** indonesia m.097kp
> **candidate paths:**
> (AND language.language_dialect (JOIN (R language.human_language.dialects) m.097kp))|
> (AND language.language_dialect (JOIN language.language_dialect.language m.097kp))|
> . . .

Re-ranking Entity Types: The prompt for re-ranking entity types uses the following template.

> For a given question please select ten relevant candidate answer entity types which might be useful for answering it in ranked order. If the number of candidates are ten or less then just rerank the candidates. candidate answer entity types are seperated by | symbol. The output should contain only relevant candidates seperated by "@@".
> **question:** what the language spoken in indonesia?
> **candidate entity types:**
> language.human_language|
> fictional_universe.fictional_language |
> . . .

Re-ranking Relations: The prompt for re-ranking relations uses the following template.

For a given question please select ten relevant candidate relations which might be useful for answering it in ranked order. If the number of candidates are ten or less then just rerank the candidates. candidate relations are seperated by | symbol. The output should contain only relevant candidates seperated by "@@".
question: what the language spoken in indonesia?
candidate Relations:
lang.human_language.countries_spoken_in|
location.country.languages_spoken|
. . .

A.1.3 Logical Form Generation Prompt

The prompt for generating the logical form uses the following template.

Translate the following question to SPARQL for Freebase based on the candidate paths (represented using SPARQL), candidate entities, candidate relations and candidate entity types, which are separated by '|', retrieved from Freebase by one or more retrievers. Please do not include any other relations, entities and entity types.
Your final SPARQL can have three scenarios:
1. When you need to just pick from candidate paths represented using SPARQL.
2. When you need to extend one of candidate paths represented using SPARQL using the candidate relations and entity types.
3. When you need to generate a new SPARQL only using the candidate entities, relations and entity types.
For entity type check please use this relation "type.object.type".
Make sure that the original question can be regenerated only using the identified entity types, specific entities and relations used in the generated SPARQL.
 # FEW-SHOTS
question: when were the texas rangers started
candidate entities: texas rangers m.07l8x
candidate paths from Retriever represented using SPARQL:
SELECT DISTINCT ?x WHERE { ns:m.07l8x ns:sports.sports_team.founded ?x . ?x ns:type.object.type ns:type.datetime . } | . . .
candidate paths from Retriever represented using SPARQL: . . .
candidate relations from Retriever:
sports.sports_team.founded| military.military_unit.formed|
. . .
candidate entity types from Retriever:
. . . media_common.finished_work|
transportation.road_starting_point|
. . .
SPARQL:
SELECT DISTINCT ?x
WHERE {
ns:m.07l8x ns:sports.sports_team.founded ?x . }
. . .

> **question:** what the language spoken in indonesia?
> **candidate entities:** indonesia m.097kp
> **candidate paths from Retriever represented using SPARQL:**
> SELECT DISTINCT ?x WHERE {
> ?x   ns:lang.language_family.languages ns:m.097kp .   ?x ns:type.object.type ns:language.language_family .
> } | . . .
> **candidate relations from Retriever:**
> lang.human_language.countries_spoken_in|
> location.country.languages_spoken| . . .
> **candidate entity types from Retriever:**
> language.human_language|
> language.language_dialect|
> **SPARQL:**

When using a single retriever, we use 5 shots with top-5 retrieval results for data paths and top-10 for relations and entity types. When multiple retrievers are used, replicating this for each retriever does not fit in the 8192 context length limit of `gpt-4-0613`. We experimented with two different strategies of reducing the context - reducing the number of exemplars (from 5 to 3 for 2 retrievers) and reducing the retrieval length (top-3 paths and top-5 relations and types for 2 retrievers). The second strategy worked marginally better over a small evaluation and we used this for our detailed experiments.

A.1.4   Execution Guided Feedback Prompt

For the execution guided prompt, we use the following template. In each iteration, we add the LLM output from the previous iterations including the generated logical form to the prompt.

> The generated SPARQL gives an empty answer when executed on freebase KG, Please generate again a different executable SPARQL using the same context and constraints.
> **SPARQL:**
> . . .

A.2   Supervised Model Details

All supervised baselines has dataset specific code bases (specifically for data path enumerations,for WebQSP and GrailQA) to handle question types which are not common. For e.g., GrailQA and GraphQA has questions which require comparative and count operators but no such questions are there in the WebQSP dataset. Similarly WebQSP has questions with time-constraints but missing from GrailQA. So GrailQA dataset has significantly broader coverage and more unique canonical logical forms than WebQSP and GraphQA (Gu et al., 2021). So for all supervised baselines, irrespective of the source→target pairs, we use the respective GrailQA code for fine-tuning in few-shot setting and inference in both zero and few-shot settings with models trained on source. We combine schema elements corresponding to the source and target pairs and use that during zero and few-shot setting.

TIARA: It consist of four modules - Entity Retrieval, Schema Retriever, Exemplary Logical Form Retrieval and Generator. We use available code and pre-trained models trained on GrailQA and WebQSP [4] (MIT License). To adapt these pre-trained models for few-shot transfer, we fine-tune them for one epoch on the target dataset. We use learning_rate of 5e-5 and batch size of 64 for fine-tuning the schema retriever. We use learning_rate of 3e-5 and batch size of 2 for fine-tuning the exemplary logical form retriever and generator. For all transfer settings, we use corresponding source trained schema retriever to retrieve top-10 relations in zero-shot and fine-tune it on target 100 samples to retrieve top-10 relations in few-shot setting. To get the classes, we use pre-trained model on the source GrailQA to retrieve top-10 classes for each question in zero-shot setting for GrailQA→GraphQA and GrailQA→WebQSP pairs whereas in few-shot setting we fine-tune pre-trained model with 100 target samples to get classes for GrailQA→GraphQA pair. For GrailQA→WebQSP pair, we use classes predicted by pre-trained model for few-shot setting also. We use relations retrieved by the TIARA schema retriever to get the top-10 classes in zero and few-shot setting for WebQSP→GrailQA-Tech, WebQSP→GraphQA-Pop pairs. During path enumerations for GrailQA→GraphQA and GrailQA→WebQSP pairs we consider all relations whereas for WebQSP→GrailQA-Tech, WebQSP→GraphQA-Pop pairs we ignore relations which starts with "user.","kg.","common." and "type.".

Pangu: We use the existing codebases for Pangu[5]. We use same entity linker as TIARA and fine-tune T5(base) (Raffel et al., 2020) based pre-trained discriminator trained on GrailQA and WebQSP datasets for 1 epoch with learning rate of 1e-4 and batch size of 1. We use pre-trained model[6] on the source GrailQA to get the top-5 answer types for each question in zero-shot setting for GrailQA→GraphQA and GrailQA→WebQSP pairs whereas in few-shot setting we fine-tune pre-trained model with 100 target samples to get answer-types for GrailQA→GraphQA pair. For GrailQA→WebQSP pair, we use answer-types predicted by pre-trained model for few-shot setting also. We use relations retrieved by the TIARA schema retriever to get the top-5 answer types in zero and few-shot setting for WebQSP→GrailQA-Tech, WebQSP→GraphQA-Pop pairs.

BERT-Ranking We use existing codebase for BERT-Ranking (Gu et al., 2021)[7] (Apache License). We fine-tune pre-trained model trained on GrailQA and WebQSP for 1 epoch with learning rate of 1e-3 and batch size of 1. During path enumerations for GrailQA→GraphQA and GrailQA→WebQSP pairs we consider all relations whereas for WebQSP→GrailQA-Tech, WebQSP→GraphQA-Pop pairs we ignore relations which starts with "user.","kg.","common." and "type.".

For all three models, in the zero-shot setting, we use these for the target without any adaptation. For few-shot adaptation of Pangu, we fine-tune the discriminator, which is the only trainable component. TIARA has more trainable components - the logical form ranker and the schema retriever in the retrieval stage and then the generator. Each of these can potentially be fine-tuned or left unadapted. Our experiments show that the schema retriever improves on fine-tuning on the target. Since the generator is trained in pipeline, we fine-tune the generator and the logical-form retriever as well based on this observation. Similar to our model, we fine-tune a baseline component for the target using 1 training epoch, and for all baselines use the same source-trained entity linker without any adaptation.

A.3 KB-Binder and gf-LLM Details

For KB-Binder, we make use of publicly available code[8] (MIT License). We use self-consistency and majority voting with 6 examples, as in the experiments in the paper. In the retrieval(-R) setting, KB-Binder samples demonstration examples by retrieving from the entire available training data. We restrict its retrieval to our target training set D_t with 100 examples.

For gf-LLM (Shu and Yu, 2024), we remove the notion of synthetic data generation proposed in the paper, so that gf-LLM retrieves KB-context for the question using the same TIARA retriever as our model and provides this as KB-grounding to a LLM for logical form generation in addition to in-context examples. Since no code is available for this model, we use our own implementation based on the description in the paper.

KB-Binder and gf-LLM report experiments using `code-davinci-002` and `gpt-3.5-turbo-0613` respectively as the LLM. For consistency and fair comparison, we replace these with `gpt-4-0613` as in our approach. These models generate logical forms in s-expression, which we preserve. As in our approach, for the zero-shot setting we provide exemplars to both models from the source.

For fair comparison, we use `gpt-4-0613` as LLM for KB-Binder, gf-LLM and FuSIC-KBQA. For FuSIC-KBQA we use $temperature = 0$ and $n = 1$.

A.4 Dataset Construction Details

For GrailQA-Tech, we include questions from the following domains from the GrailQA train and dev set: computer, medicine, astronomy, biology, spaceflight.

For GraphQA-Pop, we include questions from the GraphQA test and train sets: cvg, food, business, aviation, comic books, religion, royalty, architecture, exhibitions, distilled-spirits.

A.5 Compute Infrastructure and Time Taken

We use Hugging Face (Wolf et al., 2020), PyTorch (Paszke et al., 2019) for our experiments and use the Freebase setup specified on github [9]. We use NVIDIA A100 GPU with 40 GB GPU memory and 32 GB RAM for fine-tuning of different models.

---

[5]https://github.com/dki-lab/Pangu
[6]https://github.com/dki-lab/ArcaneQA/tree/main/answer_typing
[7]https://github.com/dki-lab/GrailQA/tree/main

[8]https://github.com/ltl3A87/KB-BINDER
[9]https://github.com/dki-lab/Freebase-Setup

Fine-tuning of different models takes less than an hour.

On average, for each test instance, retrieval (with one retriever) takes 14.4 secs, retrieval re-ranking 9.6 secs, logical form generation 14.4 secs and EGF 18.9 secs.

A.6 Few-shot Transfer Experiments: Retrieval Re-ranking analysis

Beyond logical form evaluation, we also directly evaluated retrieval performance for the different versions of FuSIC-KBQA. These clearly demonstrate that while FuSIC-KBQA(P) has much poorer retrieval performance than FuSIC-KBQA(T), FuSIC-KBQA(T,P) is better than both, validating the use of multiple retrievers. Further, LLM re-ranking always significantly boosts retrieval performance.

We use the recall error rate to measure the retrieval performance which is defined as the % of questions having recall <1.0 in terms missing gt KB elements (excluding mentioned entities) from retrieval output i.e.,lower is better). As in Tab. 7, FuSIC-KBQA(T) has low recall error rate as compared to the TIARA across all source and target pairs and in both zero and few-shot setting. FuSIC-KBQA(P) has same recall error rate as Pangu (except for WebQSP→GraphQA-Pop) because number of retrieved relations and types are mostly ≤10. And same is true for data paths≤5. FuSIC-KBQA(T,P) with TIARA and Pangu retrievers has lower recall error rate as compared to FuSIC-KBQA(T) and FuSIC-KBQA(P) in zero-shot and few-shot setting. And FuSIC-KBQA(T) outperforms FuSIC-KBQA(P) in all cases. This also correlates with the KBQA performance mentioned in the Tab. 2 (except for GrailQA→GraphQA where FuSIC-KBQA(P) is better than FuSIC-KBQA(T) for both zero-shot and few-shot.)

A.7 Transfer Experiment with Microsoft Academic KG

Due to smaller number of schema elements in Microsoft Academic KG, a supervised retriever that returns the top-K schema elements is no longer necessary. We instead directly rerank all schema elements to obtain top-k. For gf-LLM, we follow the approach as specified in the paper, selecting top-k schema elements directly from the supervised retriever. We use gpt-4-0613 as LLM for both FuSIC-KBQA and gf-LLM. We do not perform constrained decoding during generation for TIARA.

A.8 In-domain Experiments

TIARA: It consists of four modules - Entity Retrieval, Schema Retriever, Exemplary Logical Form Retrieval and Generator. We use pre-trained models[10] corresponding to 100% training data and use existing codebase[11] to train on 5% and 10% training data. We train a mention detection model for 50 epochs and choose the best model based on the validation set. We use a non-bootstrapped strategy for sampling negative logical forms during the training of Exemplary Logical Form Retrieval.

Pangu: We use the existing codebase for Pangu[12] to train a model on 5% and 10% training data. And use pre-trained model[13] corresponding to 100% training data. We use the same entity linker as TIARA and use T5(base) (Raffel et al., 2020) as our base model.

RetinaQA: We use the existing codebase for RetinaQA[14] to train a sketch generator and discriminator on 5% and 10% training data. And TIARA codebase to train schema and logical form retriever on 5% and 10% training data. We use pre-trained models for sketch generator and discriminator corresponding to the 100% training data. We use the same entity linker as TIARA.

A.9 Equivalence Check for SPARQL

We extract classes, relations (including reverse), entities, literals and functions from predicted and gold SPARQL. We use lookup in ontology files to extract classes, relations and regex to extract literals and entities. And for a given question we perform two types of matching between elements extracted from predicted and gold SPARQL (i) Multi-set based exact match: two multi-sets should be same and (ii) Set based exact match: set of elements excluding literals should be same. So if (i) condition is true and answer F1 is 1.0 then EM is 1.0 whereas if (ii) is false then EM is 0.0 and in all other cases predicted SPARQL is reviewed manually.

---

[10] https://kcpapers.blob.core.windows.net/tiara-emnlp2022/TIARA_DATA.zip
[11] https://github.com/microsoft/KC/tree/main/papers/TIARA
[12] https://github.com/dki-lab/Pangu
[13] https://github.com/dki-lab/Pangu/blob/main/trained_models.md
[14] https://github.com/dair-iitd/RetinaQA

A.10 Experiments with Mixtral

We use `Mixtral 8x7B Instruct v0.1` as open-source llm for FuSIC-KBQA with $temperature = 1e - 5$. We use same prompt as LLM for retrieval re-ranking and execution guided feedback. We also use same prompt and few-shots for logical form generation except we enclose few-shots within "#### Here are some examples: ####" delimiters and indicate the start of test query via "<<<>>>".

Translate the following question **after** $<<<>>>$ to SPARQL for Freebase based on the candidate paths (represented using SPARQL), candidate entities, candidate relations and candidate entity types, which are separated by '|', retrieved from Freebase by one or more retrievers. Please do not include any other relations, entities and entity types.

Your final SPARQL can have three scenarios:

1. When you need to just pick from candidate paths represented using SPARQL.

2. When you need to extend one of candidate paths represented using SPARQL using the candidate relations and entity types.

3. When you need to generate a new SPARQL only using the candidate entities, relations and entity types.

For entity type check please use this relation "type.object.type".

Make sure that the original question can be regenerated only using the identified entity types, specific entities and relations used in the generated SPARQL.

####

FEW-SHOTS

"Here are some examples:"

question: when were the texas rangers started

candidate entities: texas rangers m.07l8x

candidate paths from Retriever represented using SPARQL:
SELECT DISTINCT ?x WHERE { ns:m.07l8x ns:sports.sports_team.founded ?x . ?x ns:type.object.type ns:type.datetime . } | ...

candidate paths from Retriever represented using SPARQL: ...

candidate relations from Retriever:
sports.sports_team.founded| military.military_unit.formed|

...

candidate entity types from Retriever:
... media_common.finished_work| transportation.road_starting_point|

...

9163

SPARQL:
SELECT DISTINCT ?x
WHERE {
ns:m.07l8x ns:sports.sports_team.founded
?x . }
. . .
####
<<<>>>
question: what the language spoken in indonesia?
candidate entities: indonesia m.097kp
candidate paths from Retriever represented using SPARQL:
SELECT DISTINCT ?x WHERE {
?x   ns:lang.language_family.languages
ns:m.097kp .   ?x ns:type.object.type
ns:language.language_family .
} | . . .
candidate relations from Retriever:
lang.human_language.countries_spoken_in|
location.country.languages_spoken| . . .
candidate entity types from Retriever:
language.human_language|
language.language_dialect|
SPARQL:

		Model	GrailQA→ GraphQA	GrailQA→ WebQSP	WebQSP→ GrailQA-Tech	WebQSP→ GraphQA-Pop
Zero-shot	Supervised	TIARA	17.2	35.6	8.6	22.0
		Pangu	37.0	48.0	47.8	67.8
	Fusion	FuSIC-KBQA(T)	12.4	25.2	3.2	13.4
		FuSIC-KBQA(P)	37.0	48.0	47.8	67.4
		FuSIC-KBQA(T,P)	**9.6**	**23.6**	**2.6**	**10.2**
Few-shot	Supervised	TIARA	17.6	30.8	5.0	24.2
		Pangu	31.8	43.8	27.0	56.0
	Fusion	FuSIC-KBQA(T)	10.8	23.8	3.2	13.4
		FuSIC-KBQA(P)	31.8	43.8	27.0	54.8
		FuSIC-KBQA(T,P)	**7.6**	**21.6**	**2.8**	**9.2**

Table 7: Retrieval performance using error rate (lower is better) in few-shot and zero-shot transfer learning settings of different models for different source→target pairs, evaluated over 500 randomly sampled instances over the test sets. FuSIC-KBQA(T), FuSIC-KBQA(P), FuSIC-KBQA(T,P) denote FuSIC-KBQA used with TIARA, Pangu, and TIARA and Pangu together as supervised retrievers.