# Iterative Repair with Weak Verifiers for Few-shot Transfer in KBQA with Unanswerability

Anonymous ACL submission

## Abstract

Real-world applications of KBQA require models to handle unanswerable questions with a limited volume of in-domain labeled training data. We propose the novel task of few-shot transfer for KBQA with unanswerable questions and contribute two new datasets for performance evaluation. We present FUn-FuSIC – a novel solution for our task that extends FuSIC-KBQA, the state-of-the-art few-shot transfer model for answerable-only KBQA. We first note that FuSIC-KBQA's iterative repair makes a strong assumption that all questions are answerable. As a remedy, we propose Feedback for Unanswerability (FUn), which uses iterative repair using feedback from a suite of strong and weak verifiers, and an adaptation of self-consistency for unanswerabilty to better assess the answerability of a question. Our experiments show that FUn-FuSIC significantly outperforms suitable adaptations of multiple LLM-based and supervised SoTA models on our task, while establishing a new SoTA for answerable few-shot transfer as well.

## 1 Introduction

The semantic parsing formulation of the Knowledge Base Question Answering (KBQA) task takes as input a Knowledge Base (KB) and a natural language question, and outputs a logical form (or program) that produces the answer upon execution over the KB. KBQA has important real-world applications, which can require systems to be low resource (i.e., trained only with a few task-specific labeled examples), and robust, specifically the ability to identify questions that cannot be answered based on existing KB.

Traditional supervised models (e.g., (Ye et al., 2022; Shu et al., 2022; Gu et al., 2023)) and even recent LLM few-shot in-context learning (FS-ICL) architectures (Li et al., 2023; Nie et al., 2024) fall short in both aspects. Limited recent work has addressed these independently – in-domain methods for KBQA with unanswerability trained with large labeled data (Patidar et al., 2023; Faldu et al., 2024), and FuSIC-KBQA for few-shot transfer assuming questions are answerable (Patidar et al., 2024). No existing work simultaneously addresses both desiderata in a single KBQA system.

In response, we propose the novel task of *few-shot transfer learning for KBQA with unanswerability*. Specifically, the target domain may offer only a few labeled examples of answerable and unanswerable questions, while the source domain has thousands of labeled examples, but containing *only* answerable questions.

For KBQA transfer, FuSIC-KBQA uses a retrieve-then-generate framework: retrieval of relevant schema and KB snippets followed by an LLM-based generation and a subsequent iterative error guided repair. Specifically, multiple iterations of execution-guided feedback are run until a *non-empty* answer is obtained – this assumes that *all* questions are answerable. For our task, we first design FuSIC-KBQA-U, which, to address **U**nanswerability, eschews the now-inappropriate repair step, and modifies the LLM prompt to accommodate unanswerable questions, along with relevant in-context exemplars. Unlike in studies for unanswerability in general QA (Slobodkin et al., 2023), this vanilla adaptation mostly generates incorrect logical forms for unanswerable questions.

As a remedy, we design a novel solution: FUn-FuSIC (**F**eedback for **Un**answerability in FuSIC-KBQA). The key idea is to modify iterative repair, which earlier relied on a single strong verifier for the logical form's incorrectness, to rely on a *suite of strong and weak verifiers*, where strong verifiers identify certain errors, whereas weak verifiers identify potential errors in the current logical form.

FUn-FuSIC's verifiers consider both the logical form and the answer. For answers, non-emptiness check is now a weak verifier, given potentially

unanswerable questions. For logical forms, we use strong verifiers to identify obvious syntactic and semantic errors. We also propose a novel verifier: non-equivalence of the original question and the back-translation of the logical form. This is weak, due to potential errors in back-translation as well as in equivalence classification.

Using such iterative weak verification based repair, FUn-FuSIC constructs a set of candidate logical forms. Simply selecting from this set the logical form with majority answer via self-consistency (Wang et al., 2023) breaks down in the face of unanswerability. We introduce *self-consistency for unaswerability*, which additionally checks the *likelihood* of the majority answer, empty or otherwise, to produce the final output.

For evaluation in our novel task, we create two KBQA transfer datasets with unanswerable questions in the target. Our experiments show that FUn-FuSIC comprehensively outperforms different types of SoTA models suitably adapted for this task, including LLM-based and more traditional models. We further find that iterative repair of logical forms using weak verifiers holds promise for KBQA in general. Using experiments over benchmark datasets for answerable-only few-shot KBQA transfer, we show that the restriction of FUn-FuSIC for the answerable setting improves upon the known SoTA on the task.

In summary, our specific contributions are as follows. **(a)** We propose the problem of few-shot transfer for KBQA with unanswerability. **(b)** We present FUn-FuSIC that uses iterative repair with error feedback from a diverse suite of weak verifiers. **(c)** We create new datasets for the proposed task, which we make public.[1] **(d)** We show that FUn-FuSIC outperforms adaptation of SoTA KBQA models for our setting. **(e)** We also show that for standard answerable-only KBQA, FUn-FuSIC outperforms the corresponding SoTA model.

## 2 Related Work

In-domain KBQA using 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 using LLM few-shot approaches (Li et al., 2023; Nie et al., 2024; Shu and Yu, 2024) is well explored in literature. These use high volumes of labeled data, either for training or selecting the most relevant few shot exemplars.

For in-domain KBQA, unanswerability has recently been studied (Patidar et al., 2023; Faldu et al., 2024). Patidar et al. (2023) create the GrailQAbility dataset with different categories of unanswerability, and show the inadequacy of superficial adaptations of answerable-only KBQA models. RetinaQA (Faldu et al., 2024) is the SoTA model for KBQA unaswerability. However, this also requires large volumes of training data.

For KBQA transfer (Cao et al., 2022; Ravishankar et al., 2022), low-resource was originally not a focus. More recently, few-shot transfer for KBQA has been addressed (Patidar et al., 2024). Its FuSIC-KBQA model uses a retrieve-then-generate framework with an LLM-based generation stage with iterative error feedback based repair. However, this formulation assumes answerability of all questions, as described in Section 1.

Simple LLM prompting techniques have been used to address unanswerability outside of KBQA (Slobodkin et al., 2023), but without any notion of feedback or iterative repair. Other approaches (Shinn et al., 2023; Chen et al., 2023b) use execution based refinement for program generation but without any notion of unanswerability.

The extensive recent work on program self-repair using LLMs (Olausson et al., 2024) also diverges strongly from our few-shot goal. It assumes availability of unit tests for every test instance. More importantly, as in the answerable KBQA literature, it assumes that *a valid program exists* for each specification.

To the best of our knowledge, ours is the first work to apply iterative weak repair for KBQA, and to use iterative repair for unanswerability.

## 3 Background & Problem Definition

A Knowledge Base (KB) $G$ consists of a schema and data. The schema consists of entity types (or classes) $T$ and binary relations $R$ defined over pairs of types. The data consists of entities $E$ as instances of types $T$, and triples or facts $F \subseteq E \times R \times E$. Given a *target KB* $G^t$ and a natural language question $q^t$, the goal is to generate a structured query or logical form $l^t$ (in a KB query language, such as SPARQL), which when executed over $G^t$ returns an answer $A^t$ (which is a set in general). *Target few-shots* $D^t$ containing tens of labeled training examples of questions and logical

[1] https://anonymous.4open.science/r/FUn-FuSIC-1704

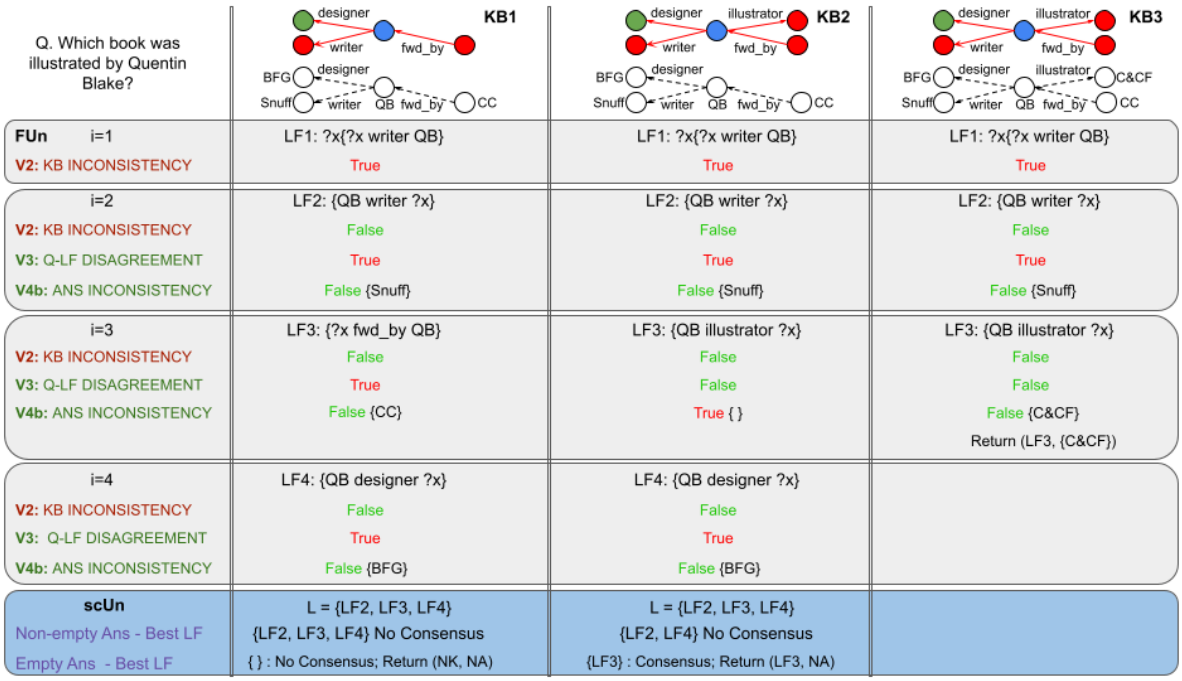


Figure 1: **F**eedback with **Un**answerability (FUn) and **s**elf **c**onsistency for **Un**answerability (scUn) for a question $Q$ when executed over different KBs with $\leq 4$ iterations. $Q$ is answerable for KB3, but unanswerable for KB1 (schema incompleteness) and KB2 (data incompleteness). FUn iterations are shown using gray blocks, with Strong Verifiers in red and Weak Verifiers in green. (V1: Syntax Error omitted for brevity) scUn is shown using the blue block.

forms. A related source domain has a *source KB* $G^s$ (with its own types, relations, entities and facts), and a larger *source training set* $D^s$ with thousands of labeled training examples.

Following Patidar et al. (2023), a question $q$ is **answerable** for a KB $G$ if it has a corresponding logical form $l$ which when executed over $G$ returns a *non-empty answer* $A$. A question is **unanswerable**, if it either (a) does not have a valid logical form for $G$ (schema-level unanswerability), or (b) it has a valid logical form $l$ for $G$, but $l$ returns an empty answer upon execution on $G$ (data-level unanswerability). Schema-level unanswerability arises due to missing types and relations, while missing entities and facts lead to data-level unanswerability. More details are in Appendix A.1.3.

In *KBQA with unanswerability*, given a question $q$, the model needs to output **(a)** a logical form $l$ and a non-empty answer A for answerable $q$, **(b)** $l = \texttt{NK}$ (No Knowledge) for schema-level unanswerable $q$, or **(c)** a valid logical form $l$ and $a = \texttt{NA}$ (No Answer) for data-level unanswerable $q$.

We now define *few-shot transfer learning for KBQA with unanswerability*. A target question $q^t$ may be answerable or unanswerable due to missing schema or data in the target KB $G^t$. Target few shot examples $D^t$ contain both answerable and unanswerable questions of different categories. However, most KBQA datasets contain only answerable questions, we model the source training data $D^s$ as containing *only answerable questions*. Now the unanswerability gap between the source and the target also needs to be bridged. More details are in Appendix A.1.

## 4 Proposed Approach: FUn-FuSIC

Our proposed architecture FUn-FuSIC adapts FuSIC-KBQA for unanswerability. The high-level algorithm is described in Algo. 1. Preserving the retrieve-then-generate framework, the retrieval stage (line 2) performs KB retrieval for $q^t$ using a set $R$ of one or more supervised retrievers. Each retriever $R_i$ is source-trained and then target fine-tuned. The retrieval output $r$ of each $R_i$ consists of relevant schema elements (types and relations) for $q^t$, and data paths emanating from mentioned entities in $q^t$. The union of these, along with $q^t$, is fed to the generation stage, which uses prompting with an LLM $\mathcal{L}$ to generate logical forms using the target few shots $D^t$. More details of the retrieval stage are in the Appendix (Sec. A.6).

FUn-FuSIC makes three modifications to the FuSIC-KBQA architecture. First, the LLM generation instruction is modified to admit the possibility

**Algorithm 1** FUn-FuSIC$(q, G^t, D^t, R, V^s, V^w, \mathcal{L})$

1: $r = \{\}$
2: **for** $i = 1$ to $k$ **do** $r = r \bigcup R_i(q, G^t)$
3: $l = \text{PUn}(\mathcal{L}, I, q, r, D^t)$
4: $(e, l, A, L) = \text{FUn}(\mathcal{L}, q, l, n, V^s, V^w, G^t)$
5: **if** $(e)$ **return**$(l^*, A^*)$
6: **else return**$(\text{scUn}(q, L, t, \mathcal{L}))$

of unanswerability, and the few shots are modified to include examples of unanswerable questions. However, this simple approach is error-prone. So, we bias the instruction towards one type of error. Specifically, when uncertain about answerability of the question, **P**rompting for **Un**answerability (PUn) (line 3) instructs $\mathcal{L}$ to generate a (possibly incorrect) logical form instead of $l = \texttt{NK}$. The detailed prompt $I$ is in the Appendix (Sec. A.8).

We now come to the more significant modifications. First, $l^{(0)}$ is iteratively repaired using feedback as before, but this step is adapted for unanswerability. This iterative repair, which we name **F**eedback for **Un**answerability (FUn) (line 4), either confidently outputs a single logical form $l$ (with corresponding answer $A$) (line 5) or generates a set $L$ of candidate logical forms, which is further analyzed. For this, we introduce **s**elf-**c**onsistency for **Un**answerability (scUn). scUn checks the likelihood of the majority answer in $L$, empty or otherwise, to produce the final output. In the rest of this section, we describe FUn and then scUn. Fig. 1 illustrates flow of FUn and scUn using examples. A real example of FUn execution is in Sec. A.13.

**Algorithm 2** FUn$(q, l, n, V^s, V^w, G, \mathcal{L})$

1: $i = 0, k = 0, L = \{\}, F = ""$
2: **while** $++i \leq n$ **do**
3: **for** $j = 1$ to $k_1$ **do**
4: $(e, f) = V^s_j(l, q, G)$
5: $F = \text{Append}(F, f)$
6: **if** $(!e)$ **break**
7: **end for**
8: **for** $j = 1$ to $k_2$ **do**
9: $(e, f) = V^w_j(l, q, G)$
10: $F = \text{Append}(F, f)$
11: **if** $(e)$ $L = L \bigcup \{l\}; k++$
12: **end for**
13: $l = Gen(\mathcal{L}, I, q, F)$
14: **end while**
15: **if** $(k = k_2)$ **return**$(\text{True}, l, \text{Exec}(l, G), L)$
16: **else return**$(\text{False}, l, \{\}, L)$

The FUn algorithm is described in Algo. 2 Starting with the initial logical form $l^{(0)}$, FUn performs at most $n$ verify-and-repair iterations to create a candidate set $L$ of probable logical forms. The example shows 3 FUn iterations for KB1 and KB2, and 2 iterations for KB3. In the $i^{th}$ iteration, FUn generates a new logical form $l^{(i)}$ by prompting $\mathcal{L}$ using $q^t$ and feedback $F$ received from checks in all previous iterations (line 13). $l^{(i)}$ goes through a sequence of verifications. FUn uses two sets of verifiers. The *strong verifiers* $V^s$ are guaranteed to be correct, while the *weak verifiers* $V^w$ are potentially erroneous. The specific verifiers that we use in this paper are defined later in the section. A template-based feedback string $f$ is appended to the generation prompt for $l^{(i+1)}$ based on the specific verifier that $l^{(i)}$ failed. If $l^{(i)}$ fails a strong verifier, it is rejected (line 6). In the example, this happens for all three KBs in iteration 1. If $l^{(i)}$ passes all checks, strong and weak (line 15), FUn terminates by outputting $(l = l^{(i)}, A = A^{(i)})$, where $A^{(i)}$ is the answer obtained by executing $l^{(i)}$. This happens in iteration 3 for KB3. Otherwise, if $l^{(i)}$ passes at least one weak verifier but not all, it is added to candidate logical form set $L$ (line 11). This happens for iterations 2, 3 and 4 for KB1 and KB2.

**Logical form verifiers:** FUn uses a suite verifiers, categorized as strong ($V^s$) and weak ($V^w$). These may be syntactic, semantic, or execution-based, defined using simple rules or complex LLM functions over $l$, $q$ and $G$. Note that unlike unit tests in program synthesis, the verifiers *do not* have knowledge of the gold logical form, the gold answer or answerability of the question.

We now briefly describe the specific verifiers that we use for this paper. Additional details about the verifiers are in the Appendix (Sec. A.8.1 and Sec. A.2). Note however that FUn is a framework that is equally capable of working with a wholly different suite of verifiers.

**(V1) Syntax Error:** As in FuSIC-KBQA, this verifier executes the logical form $l$ over $G$ and checks for syntax error. This is a strong check: a valid logical form cannot have syntax error.

**(V2) KB Inconsistency:** A logical form $l$ may be inconsistent with the schema of $G$. We identify semantic errors of different categories, such as type-incompatibility and schema hallucinations, implemented using rules over $l$ and $G$. These are also strong verifiers.

**(V3) Question - Logical Form Disagreement:** This verifier checks if $l$ is semantically equivalent to the original natural language question $q$. In our example, LF3 for KB1 disagrees with Q. This is a weak verifier. First, $q$ may not have any equivalent logical form for $G$ due to intrinsic ambiguities. For example, $q$ mentions a PERSON *from a* COUNTRY, when $G$ has the relations *born in* and *works in* between these types. Secondly, this is implemented as a probabilistic classifier that can err. We define equivalence check between $l$ and $q$ using a novel multi-stage LLM pipeline, involving naturalization of $l$ to $l^n$, back-translation of $l^n$ to natural language question $q^b$ and semantic equivalence check between $q$ and $q^b$. More details are in Sec.A.2.

**(V4) Answer Inconsistency:** This verifier executes $l$ over $G$ to obtain an answer $A$ and then checks its compatibility with $q$. This may fail for different reasons, such as **(V4a)** $A$ containing an entity mentioned in $q$, **(V4b)** $A$ being empty, and others. Note that V4a is a strong verifier while V4b is weak, since an empty answer is valid for unanswerable questions (as for LF3 for KB2), but invalid for answerable ones.

**Identifying Candidate Logical Forms:** Unless some logical form passes all checks and is therefore returned (Algo. 2 line 6), FUn constructs a candidate set $L$ of logical forms that are potentially flawed but not certainly so. For our specific suite of weak verifiers, $l^{(i)}$ is added to $L$ if it passes one of V4b ($A$ is non-empty) as for LF3 for KB1, or V3 ($l^i$ is equivalent to $q$), as for LF3 for KB2.

**Self Consistency for Unanswerability:** If FUn fails to generate a single confident logical form at the end of $n$ iterations, as for KB1 and KB2, we use scUn to decide if the best candidate $l^* \in L$ has sufficient confidence. If scUn detects sufficient confidence, as for KB2, it outputs ($l = l^*$, $A = A^*$), where $A^*$ is the answer obtained upon executing $l^*$ and may be NA. If on the other hand, scUn decides insufficient confidence, as for KB1, it outputs ($l =$ NK, $A =$ NA). For identifying the consensus choice from $L$, one possibility is self-consistency (Wang et al., 2023; Chen et al., 2023a) i.e., obtaining the answer for each $l \in L$, and returning those with the most common answer. Self-consistency requires *some answer* to accumulate enough probability by aggregation over reasoning paths. However, for unanswerable questions, no single answer accumulates sufficient probability, and self-consistency returns some low probability answer.

To address this, scUn first identifies via execution the most popular *non-empty* answer $A^*$ among logical forms in $L$, and decides using threshold $t$ if it has enough supporters in $L$ (we use $t = \lfloor \frac{|L|}{2} \rfloor$). If so, scUn uses LLM prompting to select the most appropriate supporting logical form $l^* \in L$ considering $q$, and outputs ($l = l^*$, $A = A^*$).

However, for KB1 in the example, the 3 logical forms among the candidates have 3 different answers, and therefore ($\lfloor \frac{|L|}{2} \rfloor = 1$) no consensus emerges for non-empty answers. Then scUn considers logical forms from $L$ that agree on $A =$ NA. If there are multiple such candidates, scUn selects the most suitable candidate $l^*$, again using LLM prompting, and outputs ($l = l^*$, $A =$ NA). If there is no such candidate, scUn outputs ($l =$ NK, $A =$ NA). For KB2 in the example, scUn selects LF3 – the only logical form with empty answer. Further details on scUn are in the Appendix (Sec. A.14).

## 5 Experiments

We now present experimental evaluation of FUn-FuSIC. For few-shot KBQA transfer *with unanswerability*, we address the following research questions. **(R1)** How does FUn-FuSIC compare against SoTA models for KBQA, suitably adapted for this setting? **(R2)** How does FUn-FuSIC perform across different categories of unanswerability? **(R3)** How do the different components of FUn-FuSIC contribute to its performance? In addition, for *answerable* KBQA few-shot transfer, we ask: **(R4)** How does FUn-FuSIC compare against SoTA KBQA models for this setting?

### 5.1 Experimental Setup

**Datasets:** For few-shot KBQA transfer, available datasets have only answerable questions (Patidar et al., 2024), while our task needs the target dataset to contain unanswerable questions as well. So we construct our own transfer datasets, augmenting existing ones. **GrailQAbility** is the only available KBQA dataset with unanswerable questions (Patidar et al., 2023). This was carefully designed starting from GrailQA (Gu et al., 2021), which has only answerable questions, by systematically deleting schema and data elements from the back-end KB to introduce different categories of unanswerability into the queries. We use this as one of our targets.

GraphQA (Su et al., 2016) is another popular KBQA dataset. This has the same back-end KB

| Model | WebQSP → GrailQAbility | | | | | | | WebQSP → GraphQAbility | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Overall | | Answerable | | Unanswerable | | | Overall | | Answerable | | Unanswerable | | |
| | F1 | EM-s | F1 | EM-s | F1(L) | F1(R) | EM-s | F1 | EM-s | F1 | EM-s | F1(L) | F1(R) | EM-s |
| RetinaQA | 58.4 | 42.2 | 28.7 | 26.0 | **88.0** | **84.8** | 58.4 | 49.7 | 35.8 | 18.7 | 15.2 | 80.7 | 78.7 | 56.4 |
| Pangu | 54.5 | 43.8 | 31.23 | 29.6 | 83.8 | 80.4 | 58.0 | 53.4 | 33.0 | 30.3 | 26.4 | 76.5 | 74.8 | 39.6 |
| FuSIC-KBQA-U | **76.6** | 48.2 | **67.5** | 59.2 | 85.6 | 80.4 | 37.2 | 67.5 | 34.8 | 49.3 | 40.0 | 85.7 | 82.8 | 29.6 |
| KB-Binder | 43.7 | 33.0 | 19.5 | 16.5 | 67.9 | 66.5 | 49.5 | 44.3 | 36.1 | 27.5 | 21.6 | 61.0 | 61.0 | 50.7 |
| FUn-FuSIC | **76.6** | **60.2** | 67.1 | **61.2** | 85.1 | 80.0 | **59.2** | **70.0** | **53.8** | **50.7** | **42.8** | **89.2** | **86.5** | **64.8** |

Table 1: Performance of different models on two datasets for few-shot KBQA transfer with unanswerability.

(Freebase) as GrailQA. We create our second target dataset using GraphQA, by replacing its KB with the modified KB in GrailQAbility. This introduces unanswerability into GraphQA questions and we label these appropriately as schema-level or data-level unanswerable. We name this dataset **GraphQAbility**. We create the two test sets by selecting 250 answerable and 250 unanswerable questions uniformly at random from their test sets. We create few shots by selecting 100 questions (50 answerable and 50 unanswerable) uniformly at random from the dev set and train set respectively.

The source dataset needs only answerable questions. WebQSP (Yih et al., 2016) is the third popular answerable KBQA dataset. We use WebQSP as source and create the following two source→target pairs: **WebQSP→GrailQAbility** and **WebQSP→GraphQAbility**.

WebQSP training set has 2,858 real user questions, which are manually annotated with logical forms. This is quite different from GraphQA and GrailQA which contain algorithmically generated logical forms, verbalized by crowd-workers. Because of this difference in nature of questions, these are challenging transfer datasets, beyond the unaswerability gap.

**Models for comparison:** As few-shot transfer for KBQA with unanswerability is a novel task, there are no existing baselines. For *in-domain KBQA with unanswerability*, **RetinaQA** (Faldu et al., 2024) and **Pangu** (Gu et al., 2023). which has been adapted for unanswerable questions, are the two SoTA models. For these, we use the available code.[2,3] More details are in Appendix A.10.

**FuSIC-KBQA** is the SoTA model for few-transfer for KBQA with only answerable questions. Instead of retrieve-then-generate, *KB-Binder* (Li et al., 2023) follows a generate-then-ground approach. It is the SoTA for in-domain few-shot KBQA, and overall, FuSIC-KBQA and KB-Binder outperform all other supervised and LLM-equipped KBQA models adapted for few-shot transfer (Patidar et al., 2024). We use available code for KB-Binder[4], and our own implementation for FuSIC-KBQA since source code is not available. To adapt these two baselines for unanswerability, for fair comparison, we modify their logical form generation prompt in the same fashion as PUn for FUn-FuSIC. Additionally, for FuSIC-KBQA, we remove execution-guided feedback (EGF) since it fails for unanswerability. We denote this model FuSIC-KBQA-U. Observe that FuSIC-KBQA-U can also be seen as an ablation of FUn-FuSIC, without FUn. More details about use of KB-Binder are in the Appendix A.12.

We use $\mathcal{L}$ =`gpt-4-0613` for all three models. For a fair comparison, we provide all three LLM-equipped models the *same aggregated prompt limit for a question*. To achieve this, to FUn-FuSIC we provide zero-shot generation and $n = 4$ FUn iterations, to FuSIC-KBQA-U 5-shot generation, and to KB-Binder 25-shot generation.

Though FUn-FuSIC and FuSIC-KBQA allow flexible use of multiple supervised retrievers, for meaningful comparison with RetinaQA, we adapt RetinaQA as retriever for FUn-FuSIC and FuSIC-KBQA-U. More details about KB-Binder and FuSIC-KBQA's generation is in Appendix A.12, and details of FuSIC-KBQA's retriever along compute infrastructure are in Sec. A.9.

**Evaluation Measures:** Since our primary task is generation of logical forms, our primary focus in evaluation is on logical forms as well. For logical form evaluation, the existing exact match measure (EM) (Ye et al., 2022) works only for s-expression as the language. Since FuSIC-KBQA-U and FUn-FuSIC output logical forms in SPARQL, and Pangu, RetinaQA and KB-Binder in s-expression, we propose a new measure that works for both languages. This measure, **EM-s**, considers two logical forms to be equivalent if these contain identical sets of

[2] https://github.com/dair-iitd/RetinaQA

[3] https://github.com/dki-lab/Pangu

[4] https://github.com/ltl3A87/KB-BINDER

| Model | WebQSP → GrailQAbility | | | | | | WebQSP → GraphQAbility | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Schema Level | | | Data Level | | | Schema Level | | | Data Level | | |
| | F1(L) | F1(R) | EM-s | F1(L) | F1(R) | EM-s | F1(L) | F1(R) | EM-s | F1(L) | F1(R) | EM-s |
| RetinaQA | **94.1** | **90.9** | **79.4** | 76.3 | 72.9 | 14.1 | 83.2 | 82.0 | 72.3 | 73.7 | 72.7 | 12.1 |
| Pangu | 91.1 | 87.9 | 87.9 | 69.6 | 65.9 | 00.0 | 77.3 | 74.4 | 74.4 | 73.3 | 72.7 | 00.0 |
| FuSIC-U | 85.4 | 80.6 | 30.9 | **86.0** | **80.0** | **49.4** | 86.6 | 82.6 | 19.0 | **83.3** | **83.3** | **51.5** |
| KB-Binder | 75.1 | 73.9 | 70.1 | 53.1 | 51.5 | 09.5 | 67.0 | 65.9 | 60.9 | 41.2 | 41.2 | 06.8 |
| FUn-FuSIC | 85.8 | 81.2 | 70.9 | 83.8 | 77.6 | 36.5 | **92.4** | **87.5** | **75.6** | 80.3 | 80.3 | 34.8 |

Table 2: Model performance for categories of unanswerable questions. FuSIC-U is short hand for FuSIC-KBQA-U.

relations and entities, and additionally return identical answers upon execution. This is a necessary but not sufficient condition for logical form equivalence. So, we compared EM-s with EM in settings where both are applicable, and found $> 98\%$ agreement. More details are in the Appendix A.3.

Since logical form evaluation is not completely accurate (logical forms different from the gold one may still be correct), following existing literature (Patidar et al., 2023; Faldu et al., 2024), we also evaluate answers using regular F1 (F1(R)), also lenient F1 (F1(L)), which does not penalize the original answer for the complete KB.

| | Answerable | | Unanswerable | | |
|---|---|---|---|---|---|
| | F1 | EM-s | F1(L) | F1(R) | EM-s |
| FUn-FuSIC | 57.0 | 46.0 | 92.0 | 90.0 | 62.0 |
| scUn ⇒ sc | 63.0 | 52.0 | 66.0 | 64.0 | 36.0 |
| w/o syntax | 57.0 | 46.0 | 92.0 | 90.0 | 62.0 |
| w/o kb-inc | 51.7 | 42.0 | 92.0 | 88.0 | 44.0 |
| w/o q-lf | 47.7 | 38.0 | 55.8 | 48.0 | 10.0 |
| w/o ans-inc | 55.0 | 44.0 | 92.0 | 90.0 | 62.0 |

Table 3: Ablation performance of FUn-FuSIC (removing individual components with replacement) on subset of WebQSP → GraphQAbility. scUn ⇒ sc denotes replacing scUn with self consistency. Other rows remove verifiers for syntax error (w/o syntax) (V1), KB inconsistency (w/o kb-inc) (V2), question logical form disagreement (w/o q-lf) (V3) and answer incompatibility (w/o ans-inc) (V4).

### 5.2 Unanswerability Setting

We first address research question **R1**. Performances of the different models for the few-shot transfer setting with unanswerability are recorded in Tab. 1. Note that the **Overall** columns determine superiority of one model over another, while the **Answerable** and **Unanswerable** columns provide further drill-down for analysis.

First, we observe that FUn-FuSIC significantly outperforms all baselines in terms of EM-s, and performs at par with FuSIC-KBQA-U and significantly better than all other models in F1. However, the other LLM-equipped models do not far surpass supervised models. In fact, all 4 models perform almost at par for GraphQAbility, and KB-Binder performs worse than the other 3 for GrailQAbility. This establishes usefulness of FUn+scUN for few-shot transfer KBQA with unanswerability. Secondly, each model trades off performance differently between answerable and unanswerable questions. RetinaQA, Pangu and also KB-Binder fare better for unanswerable questions, while FUn-FuSIC and FuSIC-KBQA-U fare better for answerable ones. However, FUn-FuSIC achieves the best balance between the two.

We next briefly address research question **R2**. Performance of different models for different categories of unanswerability are recorded in Tab. 2. Models struggle to fare well across both data-level and schema-level unanswerability. FuSIC-KBQA-U performs the best for data-level while performing poorly (in terms of EM-s) for schema-level. Conversely, RetinaQA performs well for schema-level, but has poor data-level EM-s. FUn-FuSIC performs poorer for data-level compared to schema-level but achieves the best balance by far between the two categories among all models.

We next address research question **R3**. Tab. 3 records performance for different ablations of FUn-FuSIC. Note that this experiment is performed on a subset of the test data (50 questions drawn randomly from the each of the answerable and unanswerable categories). The biggest benefit, for both answerable and unanswerable questions, comes from Question - Logical Form Disagreement verifier. KB Inconsistency and Answer Incompatibility verifiers also make significant contributions to the performance. This demonstrates the usefulness of weak verifiers. Replacing scUn with self-consistency, as expected, leads to a drastic drop in unanswerable performance (though this comes with a benefit for answerable questions).

### 5.3 Answerable Setting

We now address research question **R4** (answerable-only KBQA). We use two datasets from existing

| Model | WebQSP → GrailQA-Tech | WebQSP → GraphQA-Pop |
|---|---|---|
| FuSIC-KBQA | 70.8 | 52.3 |
| FUn-FuSIC(sc) | **73.6** | **67.0** |
| FuSIC-KBQA-U | 62.6 | 43.4 |
| FUn-FuSIC(scUn) | **71.2** | **65.0** |

Table 4: Performance using F1 of different models for few-shot KBQA transfer with only answerable questions. The models in the top block have prior knowledge of answerability, while those in the bottom block do not.

| | | |
|---|---|---|
| EM-s $< 1$ | | 46.2 |
| | Retr. Err. | 23.4 |
| | Gen. Err. | 22.8 |
| | $l^* = \mathsf{NK}, \hat{l} \neq \mathsf{NK}$ | 8.4 |
| | $l^* \neq \mathsf{NK}, \hat{l} = \mathsf{NK}$ | 4.6 |
| | $l^* \neq \mathsf{NK}, \hat{l} \neq \mathsf{NK}, l^* \neq \hat{l}$ | 9.8 |

Table 5: FUn error analysis on WebQSP → GraphQA-bility. $l^*$ & $\hat{l}$ denote gold & generated logical forms. Retrieval error means retrieval $r$ is missing $\geq 1$ KB elements (class, relation, entity) necessary for $l^*$. Generation error implies $\hat{l} \neq l^*$ despite correct retrieval.

literature (Patidar et al., 2024), including the hardest one (WebQSP → GraphQA-Pop).[5] For consistency with existing literature, here all models use TIARA (Shu et al., 2022) as the retriever.

This setting admits two sub-settings: **(A)** the models have knowledge that all questions are answerable, and **(B)** though all questions are answerable, the models do not have this prior knowledge.

Setting (A) has been studied for KBQA (Patidar et al., 2024), and FuSIC-KBQA is the established SoTA model, outperforming a host of supervised and LLM-based models adapted for the task. In this setting, FUn-FuSIC requires three simple modifications. (i) PUn is replaced with prompt for answerability, (ii) In FUn, category of V4b (empty answer) changes from weak verifier to strong verifier, and (iii) scUn is replaced by self-consistency.

The first two rows in Tab. 4 record performance for setting (A). FUn-FuSIC significantly outperforms FuSIC-KBQA on both datasets, creating a new SoTA for this setting. This shows the usefulness of iterative repair with a suite of strong and weak verifiers followed by self-consistency for KBQA transfer, even without unanswerability.

In the realistic second setting (B) – not studied before – the models make predictions assuming unanswerability. Here, we evaluate FuSIC-KBQA-U and FUn-FuSIC as in Sec. 5.2. The bottom two rows of Tab. 4 record their performance. We see that FUn-FuSIC outperforms FuSIC-KBQA by a very large margin. This further establishes the usefulness of scUn when guarantees about answerability are not available.

### 5.4 Error Analysis

For WebQSP → GraphQAbility, we analyzed questions whose logical forms are judged incorrect (EM-s $< 1$). See results in Tab. 5. We found three main causes for generation errors. **(1)** Some questions are inherently ambiguous, admitting multiple valid logical forms $l_1$ and $l_2$ in the original complete KB, though only one is recognized as the gold ($l^* = l_1$). Deletion to introduce unanswerability eliminates $l_1$, so we update $l^* = \mathsf{NK}$, and the prediction $\hat{l} = l_2$ is unfairly penalized. **(2)** $l^* = l_1$, $\hat{l} = l_2$, where $l_1 \neq l_2$ while being semantically equivalent, but are incorrectly judged non-equivalent by EM-s. **(3)** FUn is genuinely unable to generate $l^*$ or any semantic equivalent within its iteration limit.

We also estimated the accuracy of specific weak verifiers by manual analysis of ∼100 instances from test. Verifier V3 for *Question - Logical Form Disagreement* has accuracy of 93.6% overall, with its *Query to Natural Language Back-translation* component having 90%. Verifier V4b for *Empty Answer* has 67.8% (this is expected since ∼25% of questions have ($l^* \neq \mathsf{NK}$, $A^* = \mathsf{NA}$).

## 6 Conclusions

For real-world robust, low-resource KBQA, we have proposed the novel task of few-shot transfer learning with unanswerability. We have proposed FUn-FuSIC that augments for unanswerability the SoTA few-shot answerable-only KBQA transfer model by (i) iterative repair using feedback from a suite a strong and weak verifiers – including a novel back-translation based verifier – to create a set of candidate logical forms, and (ii) assessing this candidate set to detect unanswerability (and its category) and/or identify the best logical form using self consistency adapted for unanswerability.

Using two newly created datasets for this novel task, we show that FUn-FuSIC outperforms adaptations of SoTA models for this setting, and also for answerable few-shot transfer KBQA. Error analysis suggests that performing well across categories of unanswerability for few-shot transfer is still a challenge and should be a focus of further research.

[5] https://github.com/dair-iitd/FuSIC-KBQA/

## 7 Limitations

Since LLM inference involves randomness, ideally experiments should be repeated for multiple runs and results should report averages and error bars. Unfortunately, we were not able to do this due to the prohibitive cost of GPT-4, and our results are based on single runs.

While GPT-4 is currently the best performing LLM, it is proprietary as well as expensive. Ideally, evaluation should involve open-source freely accessible LLMs as well. We expect performance of all LLM-based approaches to drop when GPT-4 is replaced by an open LLM. Earlier research has shown that models using Mistral are still able to outperform fully supervised models for answerable few-shot transfer (Patidar et al., 2024). Whether this trend holds for the unanswerable setting is an open question. That said, following current trends, we expect the ability of open LLMs to steadily improve in the coming years.

## 8 Risks

At the highest level, our work reduces risk compared to existing KBQA systems, which when improperly adapted in a low-resource setting, incorrectly answer unanswerable questions, without admitting lack of knowledge. However, can incorrectly inferring unanswerability, citing lack of knowledge when the knowledge is in fact available, generate a new category of risk? While we cannot imagine such a risk at the present time, this may require more thought. In any case, KBQA models for unanswerability should strive to minimize this type of error, along with the other types.

## References


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*.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. 2023a. Universal self-consistency for large language model generation. *CoRR*, abs/2311.17311.

Xinyun Chen, Maxwell Lin, Nathanael Schaerli, and Denny Zhou. 2023b. Teaching large language models to self-debug. In *The 61st Annual Meeting Of The Association For Computational Linguistics*.

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.

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.

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.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. Is self-repair a silver bullet for code generation? In *The Twelfth International Conference on Learning Representations*.

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.

Mayur Patidar, Riya Sawhney, Avinash Kumar Singh, Mausam Biswajit Chatterjee, and Indrajit Bhattacharya. 2024. Few-shot transfer learning for knowledge base question answering: Fusing supervised models with in-context learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Bangkok, Thailand. Association for Computational Linguistics.

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*.

Aviv Slobodkin, Omer Goldman, Avi Caciularu, Ido Dagan, and Shauli Ravfogel. 2023. The curious case of hallucinatory (un)answerability: Finding truths in the hidden states of over-confident large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3607–3625.

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.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

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 KBQA Problem Formulations

We begin by defining the few-shot transfer learning setting for Knowledge Base Question Answering (KBQA) with unanswerability, and then present our solution for it.

### A.1.1 KBQA: Detailed Background

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 $q^t$.

### A.1.2 Few Shot Transfer Learning for KBQA

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, we are given access to a related source domain. This 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. The source and target tasks may differ significantly. First, the data and schema of the knowledge bases $G^t$ and $G^s$ and the domains they cover may be different. Secondly, the distributions of questions and logical forms defined over the KBs may be different in $D^t$ and $D^s$.

### A.1.3 KBQA with Unanswerability

A natural language question q is said to be answerable for a Knowledge Base $G^t$ if it has a corresponding logical form $l$ which when executed over $G^t$ returns a non-empty answer $A$. In contrast, a question $q$ is unanswerable for $G^t$, if it either (a) does not have a corresponding logical form that is valid for $G^t$, or (b) it has a valid logical form $l$ for $G^t$, but which on executing returns an empty answer. The first case indicates that $G^t$ is missing some schema element necessary for capturing the semantics for $q$. The second case indicates that the types $T^s$, relations $R^s$ is sufficient for $q$, but $G^t$ is missing some necessary data elements for answering it. In the KBQA with unanswerability task, given a question $q$, if it is answerable, the model needs to output the corresponding logical form $l$. and the non-empty answer $A$ entailed by it, and if it is unanswerable, the model either needs to output $NK$ (meaning No Knowledge) for the logical form, or a valid logical form $l$ with $NA$ (meaning No Answer) as the answer.

**Different Types of Unanswerability:** Unanswerable questions in KBQA can be categorized into (a) Schema Level Unanswerability : the question does not have a corresponding logical form that is valid for the KB , (b) Data level unanswerability: it has a valid logical form $l$ for the KB, but which on executing returns an empty answer. Schema level unanswerable questions can further be categorized into (1) Missing Class: The class/type required to construct the logical form is not defined for the KB, (2) Missing Relation: The relation required to construct the logical form is not defined for the KB, (3) Missing Topic Entity: The topic entity specified in the question is missing from the KB. Data level unanswerable questions can be categorized into (1) Missing entity: all classes and relations required to construct the logical form are present in the KB , but there exists no path from the topic entity node to the answer node in the KB due to missing intermediary entities (2) Missing Fact: all classes, relations and entities required to answer the question are present in the KB. However, the (subject, relation, object) path is not connected in the KB.

### A.1.4 Few-shot Transfer Learning for KBQA with Unanswerability

This setting is a generalization of few shot transfer learning for KBQA, wherein we no longer assume that the test question is answerable with respect to the Knowledge Base $G^t$. The training set $D^s$ contains thousands of answerable questions along with their corresponding logical form. The target few shots $D^t$ contain both answerable as well as unanswerable questions, along with their corresponding logical forms, if they exist.

## A.2 Error Checks

**(V2) Semantic Error (KB Inconsistency):** A syntactically correct logical form $l$ may still be inconsistent with the schema of $G$. This is a likely

error even for SoTA LLMs since these are unfamiliar with the specific KB $G$. Semantic errors have different categories, such as type-incompatibility, schema hallucinations. **(V2a)** Incompatibility in types: $l$ contains a variable and a connecting relation whose types are incompatible in $G$. This is the case for LF1 for all three KBs in the example. **(V2b)** Schema hallucinations: $l$ contains schema elements (types, relations, entities) absent in $G$. **(V2c)** Type casting errors: Literals in $l$ are not correctly type cast for $G$, e.g. numeric literals as float for Freebase. All of these are certain checks, and are implemented using rules defined over $l$ and $G$. The feedback mentions the type of error and the specifics, e.g., the hallucinated relation, or the incompatible type-relation pair.

**(V3) Question Logical form Disagreement:** FUn performs equivalence check between $l$ and $q$ using a novel multi-stage LLM pipeline. (i) The variable names in $l$ are first naturalized to $l^n$ considering $q$ and preserving semantics, e.g. by replacing *'?x'* with *'?actor'*. (ii) $l^n$ is back-translated into a natural language question $q^b$. (iii) $q^b$ is finally checked for semantic equivalence with $q$. The first two steps are performed using zero-shot prompting, while the last is performed using few-shots constructed using the target few-shots $D^t$. More details about few-shot construction are in Sec. A.8.2. The feedback mentions lack of equivalence as the type of error.

**(V4) Answer Inconsistency:** If the $l$ is syntactically and semantically correct, it is executed over $G$ to obtain an answer $a$. $a$ is then checked for compatibility with $q$. This may fail for different reasons. **(V4a)** $a$ (which is a set in general) contains an entity also in $l$ and therefore mentioned $q$, which is an aberration. **(V4b)** $a$ is empty, as in LF3 for KB2 in the example. All of these checks are implemented using rules defined over $l$ and $G$. Note that while the first two are certain checks, the last is not. An empty answer is valid for unanswerable questions, as for LF3 for KB2, but invalid for answerable ones. As before, the feedback mentions the type of error and the specifics.

### A.3 EM-s: Automated check for equivalence of SPARQL programs

As has been observed in (Patidar et al., 2023), answer evaluation by itself is not a robust measure for evaluation of KBQA models when the dataset contains unanswerability. Traditional KBQA models that generate s-expressions can be evaluated using EM, which checks for logical form equivalence between two logical forms, since it is possible to compare equivalence between two s-expressions efficiently. However, FUn-FuSIC generates sparql queries instead. Directly comparing program equivalence between two sparql queries is an undecideable problem [6]. (Patidar et al., 2024) suggests a semi-automatic strategy for comparison of sparql queries. We propose a completely automatic metric for sparql equivalence check. Two sparql queries are equivalent by the EM-s check if (a) the relations occuring in the two queries are same. (b) the entities occuring in the two queries are the same (c) the answer set obtained by executing the queries over the KB are the same. Note that the EM-s check is necessary, but not sufficient for two sparql queries to be equivalent.

Since these are a necessary but not sufficient condition for logical form equivalence, we compared EM-s with EM, where both are applicable and found $> 98\%$ agreement.

### A.4 Performance across different categories of unanswerability

We dive deeper into the different categories of unanswerability, as explained in (Patidar et al., 2023). There exist 2 broad categories of unanswerability-schema level unanswerability(absence of knowledge in terms of KB ontology or entities required to construct the logical form) and data level unanswerability(absence of facts or intermediate entities of the logical form path on the KB).

We expect that (a) due to the poor ability of supervised models to generalize in transfer learning settings, RetinaQA will be struggle to generate correct logical forms for data level unanswerable questions (b) due to the strong generalization ability of FuSIC-KBQA, it should be able to perform well for data level unanswerable questions. However, since it is biased to return incorrect logical forms instead of abstaining from returning a logical form, it will perform poorly at identifying schema level unanswerable questions. (c) FUn-FuSIC should be able to maintain the performance of FuSIC-KBQA on data level unanswerable questions to a large extent, while significantly improving the performance on schema level unanswerable questions.

Based on performance on the WebQSP $\rightarrow$ GrailQAbility and WebQSP $\rightarrow$ GraphQAbility

[6] https://users.dcc.uchile.cl/ cgutierr/papers/expPowSPARQL.pdf

datasets, we validate the that the trends are indeed as expected.

### A.5 Cost Analysis

FuSIC-KBQA, as well as the adapted versions of FuSIC-KBQA, such as U-FuSIC and FUn-FuSIC rerank the classes, relations and paths. The total cost for reranking for one question is $0.16.

The cost for generation of logical form from a prompt with 5 in-context examples is $0.16.

Thus, the approximate cost for inference of one question by U-FuSIC is $0.32.

The cost for generation of logical form from a prompt with 0 in-context examples is $0.04. The cost of checking whether two natural language questions are equivalent or not, using few-shot exemplars and chain of thought prompting is also $0.04.

The approximate cost of inference of one question by FUn-FuSIC varies between $0.24 and $0.48. The average cost over 50 randomly sampled questions from the test set is around $0.34.

Hence, the two models are comparable in terms of cost.

### A.6 FuSIC-KBQA Details

Our proposed approach FUn-FuSIC builds upon the the base architecture of FuSIC-KBQA. FuSIC-KBQA has a three step pipeline: (a) Supervised Retrieval: a supervised retriever, trained on the source domain and optionally fine-tuned on the target domain is used to obtain the top-100 classes, relations and paths that are relevant to the question asked, (c) LLM Generation: We provide the top-10 classes, top-10 relations and top-5 paths along with few-shot exemplars to generate the sparql query.

### A.7 FUn-FuSIC Prompts

Here we provide details of various prompts used by FUn-FuSIC.

### A.8 PUn prompt

The first prompt is for Prompting for Unanswerability (PUn).

**Header Prompt**

```
Translate the following question to sparql for Freebase based on the candidate sparql, candidate entities, candidate relations and candidate entity types which are separated by "|" respectively. 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 sparql. 2. When you need to extend one of candidate sparql using the candidate relations and entity types. 3. When you will generate a new sparql only using the candidate entities, relations and entity types. For entity type check please use this relation "type.object.type".D o not use entity names in the query. Use specified mids. If it is impossible to construct a query using the provided candidate relations or types, return "NK". Make sure that the original question can be regenerated only using the identified entity types, specific entities and relations.
```

**NK exemplar**

**Question:** the tv episode segments spam fall under what subject? **Candidate entities:** spam m.04vbm Candidate paths: SELECT DISTINCT ?xWHERE ?x0 ns:tv.tv_segment_performance.segment ns:m.04vbm .?x0 ns:tv.tv_segment_performance.segment ?x .?x ns:type.object.type ns:tv.tv_episode_segment . | ... **Candidate entity types:** tv.tv_series_episode| tv.tv_episode_segment | ... **Candidate relations:** tv.tv_series_episode.segments (type:tv.tv_series_episode R type:tv.tv_episode_segment)| tv.tv_subject.tv_programs (type:tv.tv_subject R type:tv.tv_program)|... **sparql:**NK

**Question Prompt**

**Question:** which school newspaper deals with the same subject as the onion? **Candidate entities:** the onion m.0hpsvmv **Candidate paths:** SELECT DISTINCT ?xWHERE ns:m.0hpsvmv ns:book.newspaper.circulation_areas ?x0 .?x0 ns:periodicals.newspapers ?x .?x ns:type.object.type ns:book.newspaper . |... **Candidate entity types:** education.school_newspaper| type:book.newspaper... **Candidate relations:** education.school_newspaper.school (type:education.school_newspaper R type:education.educational_institution) | book.newspaper_issue.newspaper (type:book.newspaper_issue R type:book.newspaper)|... **sparql:**

### A.8.1 FUN prompt

**Syntax error(V1) Feedback**

Correct the syntax of the following sparql query. Return ONLY the corrected sparql query without any explanation **sparql:** SELECT ?x AND ?y ... **Virtuoso error:** word AND not defined

**KB Inconsistency(V2) Feedback**

The generated sparql has a semantic issue warning: The types of relations don't match for variable ?x in the query. The assigned relation types by ['computer.computer_emulator.computer', 'type.object.type computer.computer_peripheral'] are ['computer.computer', 'computer.computer_peripheral']. These types are mutually incompatible... Please generate again a different executable sparql using the same context and constraints. DO NOT APOLOGIZE - just return the best you can try.

**Question Logical form disagreement(V3) feedback**

The question that you answer is NOT same as what you've been asked for! You have answered the question "Which opera productions has Gino Marinuzzi conducted?" but you were asked to answer "what is the name of the premiere opera production conducted by gino marinuzzi?". Please generate again a different executable sparql using the relations, classes and entities provided earlier. DO NOT APOLOGIZE - just return the best you can try.

The next three prompts fall under the answer incompatibility feedback

**Answer Inconsistency(V4b) feedback**

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.

**Intermediate Node(V4a) feedback**

The generated sparql returns an intermediate type node when executed on the freebase KG. Maybe the answer node is an adjacent node to what we currently query for. Please generate again a different executable sparql using the same context and constraints.

**Answer Inconsistency(V4a) feedback**

The logical form upon execution returns International System of Units, which is not answering the question. Please reconstruct the query using same context and constraints.

### A.8.2 Prompt for Verifier V3 Question Logical Form Agreement

The few shots provided for verifying question logical form agreement come from the few shots provided in the target domain $D^t$. We obtain positive samples from the dataset $D^t$ directly, using the questions and gold logical forms. For obtaining negative samples, we perform zero-shot FuSIC-KBQA inference over the dataset $D^t$. Then we consider those questions for which the predicted logical form is different from the gold logical form.

Firstly, we perform back-translation to obtain natural language question from the logical form

**Naturalization of variable names(V3(i))**

change the sparql query to have variable names representative of what objects they refer to. transform the variable names in this query. Do NOT change the prefix headers and relation names

**Conversion of Logical Form into Natural Language Question(V3(ii))**

Convert this sparql query into a natural language question. Make the question as natural as possible.
SELECT DISTINCT ?unfinishedWork WHERE { Le Moulin de Blute-Fin ns:media.unfinished_work ?unfinishedWork . ?unfinishedWork ns:type.object.type ns:media.unfinished_work . }

We use few-shot LLM prompting to obtain the explanation for why the question and logical form agree or disagree. These few-shots, for obtaining the explanation are dataset independent, and are manually written.

**Explanation Generation Prompt**

**Explain why the two questions are different. Question we answer:** who all like to eat apple or mango? **Question originally asked:** what are the people who enjoy both apple and mango? **explanation:** The question we answer returns people. The question originally asked also returns people. The question we answer finds those people who like eating apple, those people who like eating apple. The question originally asked also finds those people who like eating apple, those people who like eating apple. The question we answer uses logical operator OR. However, the question originally asked uses the logical operator AND Hence, they are different. **[total 3 exemplars] Question we answer:** Which game engines are successors to the Unreal Engine? **Question originally asked:** which video game engine's successor engine is unreal engine? **explanation:**

Some examples of few shots are provided below-

Question Logical Form Sgreement Check(V3(iii))

**Question we answer:** Who are the cricket players who have made exactly 31 stumps in one day internationals? **Question originally asked:** name the cricket player who has 31 odi stumps. **explanation:** The question we answer returns cricket players. The question originally asked also returns cricket players. The question we answer finds cricket players who have made exactly 31 stumps in one day internationals. The question originally asked also finds cricket players who have made 31 stumps in one day internationals. Both questions involve no mathematical or logical operators. **Hence, they are same. Question we answer:** Which game engines are successors to the Unreal Engine? **Question originally asked:** which video game engine's successor engine is unreal engine? **explanation:** The question we answer returns game engines. The question originally asked also returns game engines. The question we answer finds successors to the Unreal Engine. The question originally asked finds the predecessor of the Unreal Engine. The reasoning steps followed by the two questions are different. **Hence, they are different. [total 6 exemplars] Question we answer:** Which cars drive at a speed of 80? **Question originally asked:** name the car with driving speed at least 80? **explanation:**

### A.8.3 Prompt in scUn for Logical Form Selection

We store the back-translated natural language questions along with each predicted logical form. Here, we prompt the LLM to select the closest back-translated natural language question to the original question.

Best Logical Form Selection Prompt

**orig_nl_qn =** which surf films has sarah finn served as the casting director? 1. **pred_nl:** Which surfing films has Sarah Finn directed the casting for? 2. **pred_nl:** Which surfing films has Sarah Finn been the casting director for? of the 2 predicted nl questions, which is closest to the original nl question. Even if none is very close, return the one that is semantically closest? Please explain your answer as well

### A.9 Supervised Models training details

We use Hugging Face (Wolf et al., 2020), PyTorch (Paszke et al., 2019) for our experiments and use the Freebase setup specified on github [7] . We use NVIDIA A100 GPU with 40 GB GPU memory and 32 GB RAM. For training the discriminator module of RetinaQA, we require 2 GPUs. (1) For the answerable experiments, we use the supervised models as specified in (Patidar et al., 2024). (2) For the unanswerability experiments, we train all models from scratch. (a) We use RnG-KBQA entity linker [8] (BSD 3-Clause License) trained on the answerable subset of GrailQAbility for all our experiments. (b) We train the RnG-KBQA path retriever on answerable subset of WebQSP[9] (BSD 3-Clause License). The number of training epochs is determined by the performance of the model over the answerable questions in the dev set. (c) We train the TIARA schema retriever on the answerable subset of WebQSP [10] (MIT License) (d) We train the sketch generator and discriminator of RetinaQA on the answerable subset of WebQSP[11].

### A.10 Supervised Models inference details

We train all components on WebQSP, using the corresponding target domain's dev set as a validation set for early stopping. In the absence of unanswerable questions for training, both models use a threshold fine-tuned on a dev set to detect

[7] https://github.com/dki-lab/Freebase-Setup

[8] https://github.com/salesforce/rng-kbqa/tree/main/GrailQA/entity_linker

[9] https://github.com/salesforce/rng-kbqa/blob/main/WebQSP/scripts/run_ranker.sh

[10] https://github.com/microsoft/KC/tree/main/papers/TIARA/src

[11] https://github.com/dair-iitd/RetinaQA

schema-level unanswerability. We again use the target dev sets for this.

We use the dev set in RetinaQA, during discriminator inference for (a) determining how to best utilize the candidate paths. We might (i) not provide candidate paths (ii) provide candidate paths in GrailQA format (iii) provide candidate paths in WebQSP format. We select the best alternative based upon the performance of the model over the dev set. For the WebQSP → GrailQAbility dataset, we observe (ii) works best, whereas for the WebQSP → GraphQAbility dataset, we observe (i) works best. (b) determining the threshold value. RetinaQA applies a threshold on the scores - for a question, if the highest score candidate logical form has a score less than the threshold, the question is labeled as NK. We choose the optimal value of the threshold to maximize the overall EM-s score over the dev set.

### A.11 Pangu Adaptation Details

Similar to RetinaQA, We train all components on WebQSP, using the corresponding target domain's dev set as a validation set for early stopping. We use 1 GPU for training. Same as RetinaQA, we use the dev set to determine the threshold for schema-level unanswerability. Pangu-T applies a threshold on the scores - for a question, if the highest score candidate logical form has a score less than the threshold, the question is labeled as NK. We choose the optimal value of the threshold to maximize the overall EM-s score over the dev set.

### A.12 KB-Binder and FuSIC-KBQA Adaptation Details

For KB-Binder, we make use of publicly available code [12] (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 25 examples. KB-Binder reports experiments using `code-davinci-002` as the LLM. For consistency and fair comparison, we replace these with `gpt-4-0613` as in our approach. This model generates logical forms in s-expression, which we preserve.

For FuSIC-KBQA (Patidar et al., 2024), since no code is available for this model, we use our own implementation based on the description in the paper. For FuSIC-KBQA, and FUn-FuSIC we use $temperature = 0$.

[12] https://github.com/ltl3A87/KB-BINDER

### A.13 Real Example of FUn

**Question:** Where all were the Olympics held before 2024?

**Logical Form 1**

```
?y{ ?x olympics.venue ?y
?y olympics.year ?z
FILTER(?z = 2024)}
```

**Feedback 1**

**Type:** Logical Form Semantic Error
**Description:** ?y is assigned types ['venue', 'olympics'] by relations ['olympics.venue', 'olympics.year']. These types are mutually incompatible.

**Logical Form 2**

```
?y{ ?x olympics.venue ?y
?x olympics.year ?z
FILTER(?z = 2024)}
```

**Feedback 2**

**Type:** Natural Language Semantic Error
**Description:** The question originally asked returns all places where the Olympics were held before 2024. However, the question we answer returns the places where the Olympics were held in 2024.

**Logical Form 3**

```
?y{ ?x olympics.venue ?y
?x olympics.year ?z
FILTER(?z < 2024)}
```

**Feedback 3**

**Status:** Passes All Checks
**Details:** lf3 satisfies syntax, logical form semantic, natural language semantic and execution checks.

### A.14 Algorithm for Self Consistency with Unanswerability

The high level algorithm for self consistency with Unanswerability (scUn) is described in Algo. 3.

**Algorithm 3** ScUn$(q, L, t, \mathcal{L})$

$(c, l, A) =$ assessConf$(q, L, t, \mathcal{L})$
**if** $(c)$, **return**$(l, A)$
**else** , **return**$(\mathsf{NK}, \mathsf{NA})$

The high level algorithm for assessing confidence in the set of candidate logical forms (assessConf) is described in Algo. 4.

**Algorithm 4** assessConf$(q, L, t, \mathcal{L})$

$(c, L^p, A^p) =$ popAnsNE$(L, t)$
**if** $(c)$ **then**
    $l =$ selectBestNE$(q, L^p, \mathcal{L})$
    **return**$(\text{True}, l, A^p)$
**end if**
$(c, L^p) =$ popAnsE$(L, t)$
**if** $(c)$ **then**
    $l =$ selectBestE$(q, L^p, \mathcal{L})$
    **return**$(\text{True}, l, \mathsf{NA})$
**end if**
**return**$(\text{False}, \mathsf{NK}, \mathsf{NA})$