RETINAQA : A Knowledge Base Question Answering Model Robust to both Answerable and Unanswerable Questions

Anonymous ACL submission

Abstract

State-of-the-art KBQA models assume answerability of questions. Recent research has shown that while these can be adapted to detect unaswerability with suitable training and thresholding, this comes at the expense of accuracy for answerable questions. We propose a new model for KBQA named RetinaQA that is robust against unaswerability. It uses discrimination instead of generation to better identify questions that do not have valid logical forms. Additionally, it complements KB-traversal based logical form retrieval with sketch-filling based logical form construction. This helps with questions that have valid logical forms but no data paths in the KB leading to an answer. We demonstrate that RetinaQA significantly outperforms adaptations of state-of-the-art KBQA models across answerable and unanswerable questions. Remarkably, it also establishes a new state-of-the art for answerable KBQA by surpassing existing models.

1 Introduction

The problem of natural language question answering over knowledge bases (KBQA) has received a lot of interest in recent years (Saxena et al., 2022; Zhang et al., 2022; Mitra et al., 2022; Wang et al., 2022; Das et al., 2022; Cao et al., 2022; Ye et al., 2022; Chen et al., 2021; Das et al., 2021; Shu et al., 2022; Gu et al., 2023), where natural language questions are answered over a structured knowledge base, most commonly via producing formal queries or logical forms that are then executed over the knowledge base to retrieve the answers. All existing models for KBQA assume answerability of questions over the given KB. However, this is an unrealistic assumption, since user queries are typically agnostic of the underlying KB, which are typically incomplete. While specialized models for handling unanswerability have been proposed for other question answering tasks (Rajpurkar et al., 2018; Choi et al., 2018; Reddy et al., 2019; Sulem et al., 2022; Raina and Gales, 2022), there is no such model for KBQA.

A recent study proposed a benchmark dataset called GrailQAbility (Patidar et al., 2023), which adapts the GrailQA dataset (Gu et al., 2021) to incorporate different categories of unanswerable questions, and proposes the task of detecting unanswerabilty while answering KB questions. This work also demonstrated the challenges involved in this task. The state-of-the-art KBQA models naturally perform poorly for unanswerable questions. This performance improves with appropriate adaptation for unaswerability, namely (a) training by including unanswerable questions along with answerable ones, and (b) thresholding to separate the two question categories. However, both types of adaptation significantly hurt performance for answerable questions. Additionally, different state-of-the-art models struggle with different categories of unaswerability. Some struggle with questions for which schema elements (i.e. relations or entity types) are missing in the KB and which therefore do not have valid logical forms. Others struggle with questions for which data elements (i.e. entities or facts) are missing in the KB and which therefore have logical forms that are valid but return empty answers. This highlights the importance of rethinking KBQA architectures that are robust against different categories of unanswerability.

Based on our analysis of state-of-the-art KBQA models, we develop a few key insights about KBQA with unanswerability. First, good model calibration is crucial for separating questions that are answerable (having a valid logical form) and those that are unanswerable due to missing schema elements (not having a valid logical form). Secondly, while KB traversal-based retrieval is useful for identifying candidate logical forms for answerable questions, this fails when relevant data elements are missing in the KB, but a valid logical form exists. Detecting this type of unanswerability requires traversal-free logical form construction.

Based on these insights, we propose a new multi-staged **RETrIeve**, ge**N**erate and r**A**nk model for KBQA which is robust against unanswerability named RetinaQA. Based on our observation that discriminative approaches are better calibrated than generative ones, instead of generating logical forms, RetinaQA discriminates between candidate logical forms. This helps in better identification of questions with missing schema elements and therefore no valid logical forms. To identify candidate logical forms, RetinaQA complements KB-traversal based retrieval with sketch-filling based construction, which generates KB-independent sketches and then grounds these by directly retrieving schema elements relevant for the question. This enables identification of logical forms for questions with missing data elements and therefore no connected path in the KB. Interestingly, these architectural choices help for answerable questions as well. Discriminative scoring of syntactically and semantically valid candidates leads to clearer separation between correct and incorrect logical forms.

RetinaQA brings together and adapts ideas from different KBQA architectures for robust KBQA over answerable and unanswerable questions. While traversal based retrieval (Ye et al., 2022; Chen et al., 2021; Shu et al., 2022) and sketch-filling (Cao et al., 2022; Ravishankar et al., 2022; Li et al., 2023) has been in use in KBQA for in-domain and transfer settings respectively, this is the first model that recognizes the simultaneous need for both styles for handling unanswerability and unifies these in a single architecture. Also, while step-by-step discrimination has been recently proposed for KBQA (Gu et al., 2023), this is the first model that discriminates between fully formed logical forms as the final stage.

Using experiments over GrailQAbility, we demonstrate that RetinaQA significantly outperforms adaptations of multiple state-of-the-art KBQA models that assume answerability, not only across different categories of unanswerable questions, but also for answerable questions. We also demonstrate the advantages of RetinaQA for KBQA in general by outperforming existing KBQA models to establish a new state-of-the-art on the GrailQA dataset.

2 Related Work

The predominant approach for KBQA is to construct logical forms based on the question which are then executed to retrieve answers (Cao et al., 2022; Ye et al., 2022; Chen et al., 2021; Das et al., 2021). State-of-the-art models involve a KB traversal-based retrieval stage, that retrieves k-hop data paths from linked entities in the question (Ye et al., 2022; Shu et al., 2022). Some models instead (Chen et al., 2021) or additionally (Shu et al., 2022) retrieve schema elements (namely entity types and relations) based on the question. These are used to generate the target logical form. *These architectures are completely dependent on KB-traversal for creating input context for logical form generation.* In contrast to this generative style, Pangu (Gu et al., 2023) uses language models to incrementally evaluate and discriminate between partial logical forms. *We are not aware of any approach that performs one-shot discrimination on fully-formed logical form candidates as the final stage.*

In addition to iid settings, transfer (Cao et al., 2022; Ravishankar et al., 2022) and few-shot (Li et al., 2023) settings has also been studied for KBQA. Here, test questions involve KB relations and entity types unseen during training. These approaches use the notion of generalizable sketches (also called drafts or skeletons) that capture the syntax of the target language. Such sketches are first generated and then filled in with KB-specific arguments to construct complete programs, which are then scored and ranked. *Notably, these transfer architectures do not involve any traversal based component to retrieve logical forms.*

Unanswerability and specialized models for detecting unanswerable questions have been studied for different question answering tasks (Rajpurkar et al., 2018; Choi et al., 2018; Reddy et al., 2019; Sulem et al., 2022; Raina and Gales, 2022). *However, no specialized models have been proposed for detecting unanswerable questions in KBQA.* All existing KBQA models assume that questions have valid logical forms with non-empty answers. Even in the transfer setting for KBQA (Cao et al., 2022; Ravishankar et al., 2022), questions are still assumed to be answerable in the target domain though the logical forms may involve schema elements unseen during training. Recent work (Patidar et al., 2023) has proposed the GrailQAbility benchmark by modifying the popular GrailQA dataset (Gu et al., 2021) to incorporate various cat-

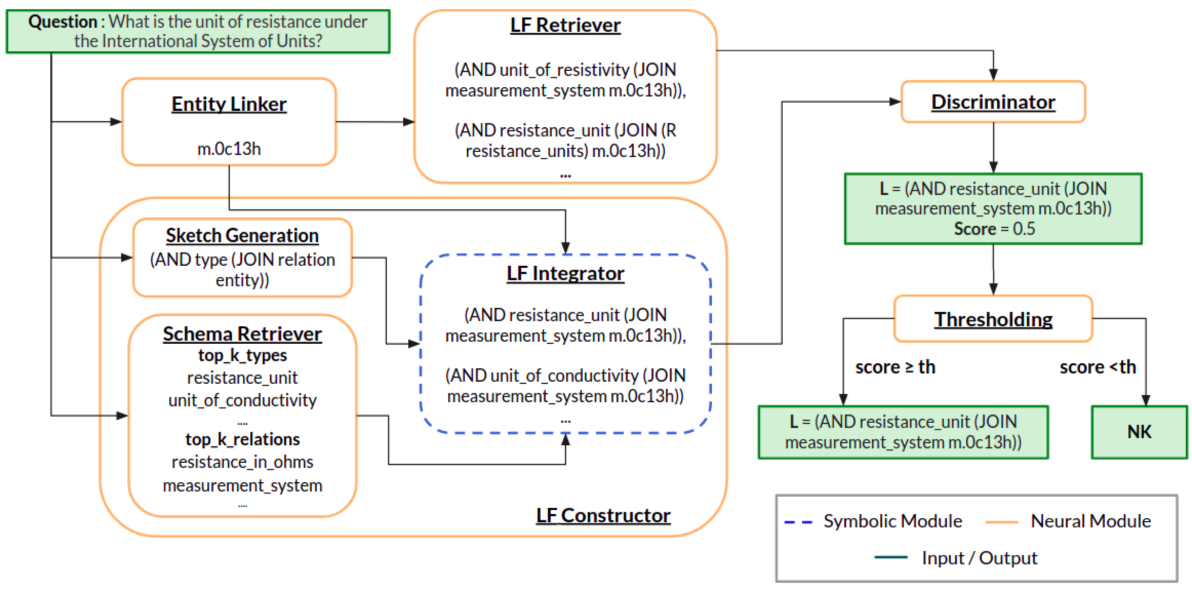
Question : What is the unit of resistance under the International System of Units?
Entity Linker
m.0c13h
LF Retriever
(AND unit_of_resistivity (JOIN measurement_system m.0c13h)),
(AND resistance_unit (JOIN (R resistance_units) m.0c13h))
...
Discriminator
L = (AND resistance_unit (JOIN measurement_system m.0c13h))
Score = 0.5
Sketch Generation
(AND type (JOIN relation entity))
Schema Retriever
top_k_types
resistance_unit
unit_of_conductivity
...
top_k_relations
resistance_in_ohms
measurement_system
...
LF Integrator
(AND resistance_unit (JOIN measurement_system m.0c13h)),
(AND unit_of_conductivity (JOIN measurement_system m.0c13h))
...
LF Constructor
Thresholding
score ≥ th
score < th
L = (AND resistance_unit (JOIN measurement_system m.0c13h))
NK
Symbolic Module
Neural Module
Input / Output

Figure 1: Model Architecture. Here L refers to the predicted logical form.

egories of unanswerable questions. This work also demonstrates the shortcomings of loose adaptions of existing KBQA models that assume answerability for the detecting unanswerable questions.

3 Problem and Solution

We first briefly define the KBQA with unanswerability task and then describe the architecture of our proposed model RetinaQA.

3.1 KBQA with Unanswerability

A Knowledge Base G consists of a schema S with data D stored under it. The schema consists of entity types T and binary relations R defined over pairs of types. Together we refer to these as schema elements. The data D consists of entities E as instances of types T, and facts $F \subseteq E \times R \times E$. Together, we refer to these as data elements. We follow the definition of Patidar et al. (2023) for defining the task of Knowledge Base Question Answering (KBQA) with unanswerability. A natural language question q is said to be answerable for a KB G if it has a corresponding logical form l which when executed over G returns a non-empty answer A. In contrast, a question q is unanswerable for G, if it either (a) does not have a corresponding logical form that is valid for G, or (b) it has a valid logical form l for G, but which on executing returns an empty answer. The first case indicates that G is missing some schema element necessary for capturing the semantics for q. The second case indicates that the schema S is sufficient for q, but G 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. While different formalisms have been proposed for logical forms, we use *s-expressions* (Gu et al., 2021). These have set-based semantics and functions with arguments and return values as sets.

3.2 The RetinaQA Model

At a high level, RetinaQA has two stages - **logical form enumeration**, followed by **logical form ranking**. For logical form enumeration, RetinaQA follows two complementary approaches and then takes the union. The first is **KB-traversal based retrieval**. Starting from linked entities in the question, RetinaQA traverses KB paths and transforms these to logical forms. The second is **sketch-filling based construction**, which is critical when the KB has missing data elements for the question. Here, RetinaQA first *generates* **logical form sketches** corresponding to the question, and then *enumerates* semantically valid groundings for these by *retrieving* relevant KB schema elements for filling in the sketch arguments. Once candidate logical forms are so identified, RetinaQA uses *discrim-*

inative scoring to rank these logical forms with respect to the question. We next explain each of these components in more detail.

Entity Linker: The pipeline starts with linking mentioned entities in the question with KB entities E. This is required for both logical form retrieval and logical form construction. We use an off-the-shelf entity linker (Ye et al., 2022) previously used in the KBQA literature (Shu et al., 2022; Gu et al., 2023). More details are in the Appendix. If the mentioned entities are missing in the KB, the entity linker returns an empty set.

Logical Form Retriever: As the first approach to enumerating logical forms, RetinaQA uses KB path traversal (Ye et al., 2022). RetinaQA traverses 2-hop paths starting from the linked entities and transform these to logical forms in s-expression. These logical forms are then scored according to their similarity with the question and the top-10 logical forms are selected for the next stage. Following (Ye et al., 2022), we score a logical form l and question q as:

$$s(l, q) = \textsc{Linear}(\textsc{BertCls}([l; q])) \tag{1}$$

and optimize a contrastive objective for ranking:

$$\mathcal{L}_{ret} = -\frac{\exp(s(l^*, q))}{\exp(s(l^*, q)) + \sum_{l \in L \wedge l \neq l^*} \exp(s(l, q))} \tag{2}$$

where l^* is the gold-standard logical form for q and L is the set of logical forms similar to l^*. Note than the transformation to logical forms from KB-paths only covers certain operators (such as *count*), but not some others (such as *argmin*, *argmax*), so that this enumeration approach is not guaranteed to cover all logical forms. Note also that this approach will not retrieve the correct logical form when q has a valid logical form, but no connected data path leading to an answer.

Logical Form Constructor: The second approach used by RetinaQA for logical form enumeration is sketch-filling. Drawing inspiration from the transfer approaches for KBQA (Cao et al., 2022; Ravishankar et al., 2022; Li et al., 2023), RetinaQA uses logical form sketches, which capture KB-independent syntax of s-expressions with functions, operators and literals, and replace KB-specific elements, specifically entities, entity types and relations, with arguments. RetinaQA first generates sketches using a **sketch generator**, then directly retrieves schema relevant elements as candidates for arguments using a **schema retriever**, and finally fills in arguments for each candidate sketch using the retrieved argument candidates in all possible valid ways using a **logical form integrator**. Since this style bypasses data-path based KB-retrieval this can construct valid logical forms when these exist, even when some relevant data element for the question is missing in the KB.

Sketch Generator: The sketch generator takes the question q as input and outputs a sketch s, optimizing a cross-entropy-based objective:

$$L_{sketch} = -\sum_{t=1}^{n} \log(p(s_t | s_{<t}, q))$$

Specifically, we fine-tune T5 (Raffel et al., 2020) - a transformer-based Seq2Seq model. Constraint decoding is performed during inference to ensure the syntactic correctness of the generated sketch.

Schema Retriever: To retrieve candidate arguments for generated sketches, we follow the schema retriever pipeline of TIARA (Shu et al., 2022). (Note that TIARA does not have a sketch generator, but instead uses retrieved schema elements as input to a logical form generator.) It works very similarly to the logical form retriever, only working with schema elements instead of logical forms. It uses the form of Eqn.1 to score a schema element x and the question q, and uses a loss similar to Eqn.2 for optimization. We train two retriever models, one for relations and one for types, and use the top-10 types and top-10 relations as candidate arguments for each question.

Logical form Integrator: This component grounds the generated candidate sketches using the retrieved candidate arguments and also the linked entities to construct complete logical form candidates. Each candidate sketch is grounded using every possible combination of arguments. A symbolic checker is used to ensure type-level validity of the grounded logical forms for the KB G. This also avoids a combinatorial blow-up and restricts the space of logical form candidates. This component does not involve any trainable parameters.

Logical Form Discriminator: Here, RetinaQA considers the union of logical form candidates from the retriever and constructor components and finally scores and ranks these. A T5 encoder-decoder model is trained to compute scores. We follow (Zhuang et al., 2022), question and logical form with a separator are fed into the encoder and then we use decoding probability over special token [1] as ranking score. It uses a contrastive learning based optimisation objective similar to Eqn.2. We perform random negative sampling. Generally the set of negative candidates is very small hence for most of the questions negative samples cover entire negative candidate set. For a test question, the candidate logical forms are ranked according to the predicted discriminator scores. If the score of top-ranked candidate is below a threshold (tuned on validation set), it classified as unanswerable i.e. l =NK. Otherwise the top ranked candidate is predicted as the logical form. This helps in identifying questions for which valid logical forms do not exist due to missing schema elements. Our experiments suggest that this also helps in separating correct and incorrect logical forms for answerable questions.

4 Experiments

We address the following research questions: (1) How does RetinaQA compare against existing KBQA approaches, in settings that have both answerable and unanswerable questions, and also in those that have only answerable questions? (2) How does RetinaQA perform for different categories of unanswerable questions, i.e., those that are unanswerable due to missing schema elements, and those with missing data elements? (3) What are the individual contributions of various model components in RetinaQA towards its performance in the above two questions?

4.1 Experimental Setup

Datasets: We experiment on two datasets, GrailQA (Gu et al., 2021), and GrailQAbility (Patidar et al., 2023). GrailQA is a popular KBQA dataset, but it contains only answerable questions. It has 64,331 questions and their associated logical forms. The background KB is Freebase. It contains questions at various levels of generalization: iid (seen schema elements), compositional (unseen combination of seen schema elements) and zero-shot (unseen schema elements). The most complex questions can have multiple operators and up to 4 relations. GrailQAbility is a recent dataset that adapts GrailQA to additionally incorporate unanswerable questions. The unanswerable questions are constructed by systematically dropping data and schema elements such as facts, entities, relations and types from the GrailQA KB.

Evaluation Metrics: We primarily focus on evaluating the logical form using the Exact Match (EM) metric, which verifies whether the model-generated logical form is same as the gold logical form (which is NK for unanswerable questions with missing schema element). We also evaluate the answers using the F1 score, which compares the model generated answers with the gold answers. For unanswerable questions, similar to prior work (Patidar et al., 2023), we report two F1 scores – the strict score compares the list of answers based on the given incomplete KB, and the lenient score accepts an answer even if it is absent from the given KB, but was present in the original GrailQA KB.

Baselines: We compare RetinaQA against existing state-of-the-art KBQA models, as per the GrailQA leaderboard and code availability. These are TIARA (Shu et al., 2022), RnG (Ye et al., 2022), and Pangu (Gu et al., 2023). Of these, the first two are shown to the best performing models on GrailQAbility, and Pangu is a SoTA model for GrailQA.[2] For fair comparison, all models use the same entity linker (Ye et al., 2022) and T5-base as base LLM. For GrailQAbility, we adapt all models appropriately for answerability detection. Specifically, we perform thresholding (denoted as "+T") on entity disambiguation and logical form generation to output NK. The thresholds are tuned on the dev set. Additionally, we train the models using both answerable and unanswerable questions (denoted as A+U training vs A training). Further implementation details are in appendix.

4.2 Results on GrailQAbility

Table 1 reports aggregate performance on GrailQAbility. With A+U Training, RetinaQA+T outperforms all models overall and is about 9 pct points ahead of the closest competitor (Pangu+T) in term of EM. For unanswerable questions, RetinaQA achieves a 16 pct points improvement, while being consistently better at answerable questions also. Unsurprisingly, thresholding helps all models for unanswerable questions and hurts slightly for answerable ones. This drop is relatively small for Pangu and RetinaQA, suggesting that they are better calibrated due to their discriminative training.

Table 2 drills down on performance for different categories of unanswerability. First, we observe

[1]We use $< extra\_id\_6 >$ token of T5 for tuning ranking score.

[2]https://dki-lab.github.io/GrailQA/

Train	Model	Overall			Answerable			Unanswerable		
		F1(L)	F1(R)	EM	F1(L)	F1(R)	EM	F1(L)	F1(R)	EM
A	RnG	67.8	65.6	51.6	78.1	78.1	74.2	46.9	40.1	5.7
	RnG+T	67.6	65.8	57	71.4	71.3	68.5	59.9	54.5	33.6
	Tiara	75.05	72.84	53.69	80.03	80	75.63	64.95	58.31	9.2
	Tiara + T	73.26	71.62	55.23	74.08	74.05	70.56	71.6	66.68	24.15
	Pangu	63.09	60.06	54.55	78.72	78.7	74	31.4	22.25	15.13
	Pangu + T	79.14	77.89	66.53	75.52	75.51	72.37	86.48	82.7	54.68
	RetinaQA	76.83	75.24	64.54	**81.22**	**81.2**	**77.41**	67.93	63.16	38.45
	RetinaQA+ T	**83.3**	**82.18**	**73.76**	81.19	81.17	75.01	**87.59**	**84.22**	**71.2**
A+U	RnG	80.5	79.4	68.2	75.9	75.9	72.6	89.7	86.4	59.4
	RnG+T	77.8	77.1	67.8	70.9	70.8	68.1	92	89.8	67.2
	Tiara	78.29	77.43	66.29	71.33	71.32	68.29	92.4	89.82	62.24
	Tiara + T	77.67	76.94	66.87	69.89	69.88	66.98	93.43	91.24	66.65
	Pangu	63.59	60.42	53.79	79.45	79.42	73.49	31.42	21.89	13.85
	Pangu +T	74.84	72.77	66.14	79.44	79.41	71.62	65.52	59.32	55.03
	RetinaQA	77.31	75.71	64.79	**80.98**	**80.97**	**76.95**	69.87	65.04	40.14
	RetinaQA+ T	**83.3**	**82.69**	**77.45**	77.91	77.91	75.16	**94.21**	**92.38**	**82.1**
A+U	RetinaQA - LFR + T	77.36	76.37	65.37	73.4	73.39	70.9	85.38	82.43	54.17
	RetinaQA - LFI + T	74.89	73.53	53.89	70.89	70.85	68.07	83.01	78.95	25.13
	RetinaQA - (SG ∪ SR) + T	64.68	62.58	52.46	72.99	72.95	68.13	47.84	41.54	20.7

Table 1: Performance of different models on the GrailQAbility dataset: overall and for answerable and unanswerable questions. A indicates training with answerable questions, A+U with answerable and unanswerable questions, +T indicates thresholding.

Train	Model	Schema Element Missing						Data Element Missing			
		Type		Relation		Mention Entity		Other Entity		Fact	
		F1(R)	EM	F1(R)	EM	F1(R)	EM	F1(R)	EM	F1(R)	EM
A	RnG+T	55.5	49.5	57.1	46.6	44.7	40.3	56	11.5	58.6	13.9
	Tiara + T	66.27	21.7	70.21	28.06	61.01	23.43	68.91	22.97	68.29	23.63
	Pangu + T	**87.97**	**87.5**	**80.07**	**79.63**	90.57	**90.41**	83.19	0	76.48	1.07
	RetinaQA+ T	86.32	80.31	79.41	62.08	**90.72**	77.83	**85.71**	**68.07**	**84.68**	**71.14**
A+U	RnG+T	93.4	86.8	89.7	**85.5**	92.1	89.6	87.1	30.8	86	32.5
	Tiara + T	91.63	83.84	**90.9**	72.37	**94.5**	71.38	91.6	50.42	90.38	52.85
	Pangu+T	90.8	**90.68**	78.66	78.44	90.41	**90.25**	11.76	0	12.59	0.95
	RetinaQA+ T	**94.22**	90.21	88.52	81.91	94.34	86.64	**93.84**	**75.91**	**94.3**	**76.13**

Table 2: Performance of different models for the unanswerable questions in GrailQAbility, grouped by categories of KB incompleteness. Note that missing mention entities result in invalid logical form, while other missing entities lead to valid logical form with no answer.

that for the baselines, performance varies significantly across different categories. Pangu is good for missing schema elements but the worst for missing data elements. TIARA is the best baseline for missing data elements but is not as good for missing schema elements. The reasons for such behaviors are described in appendix:A.2. We observe that RetinaQA performs the best by large margins on questions with missing data elements, and fairly comparably with Pangu for missing schema elements, making it the overall model of choice across different categories of unanswerability. We also note that thresholding on RetinaQA results in minimal or no loss for questions with missing data (which have valid logical forms), and in huge gains for questions with missing schema elements.

Our ablations (Section 4.4) suggest that data drop gains in RetinaQA are primarily due to its enumeration-independent schema retriever. There we further compare logical form generation (RnG, TIARA) and logical form discrimination based approaches (Pangu, RetinaQA) on unanswerability due to missing schema elements. We notice that for RnG and TIARA the performance gain comes mainly from A+U training rather than from thresholding, leading to a drop in performance for answerable questions. However, for Pangu and RetinaQA, thresholding makes a significant contribution to the performance gain, leading to robust performance for answerable questions.

As a testimony to its robustness, in the A Training setting, RetinaQA achieves comparable performance for answerable and unanswerable questions, with a gap of only 4 pct points for EM. This gap is 18 to 45 pct points for other models. Other trends are very similar to the A+U setting. Additionally, we see that RetinaQA largely outperforms existing models across different generalization settings for answerable (Table 3) and unanswerable questions (Table 5 in appendix). For answerable questions, RetinaQA beats previous the best results for IID and compositional generalization, but for zero-shot generalization, RetinaQA has a comparable or slightly worse performance than Pangu. This is mainly because of the traversal dependence trade-off, as we explain further in Section 4.4.

4.3 Results on GrailQA

Since RetinaQA performs the best for answerable questions as well in GrailQAbility, in Table 4, we report results on GrailQA, which is an answerable-only benchmark. Since all questions are answerable, we apply Execution Guided Check (EGC) as the final step for all models including RetinaQA. With EGC, models output the highest-ranked logical form which when executed over the KB returns a non-empty answer. We find that RetinaQA beats previous state of the art by around 1.2 pct points for F1 and 1.8 pct points for EM. Further analysis of types of generalization across answerable questions (Table 4) shows very similar trends as for answerable questions in GrailQAbility.

4.4 Ablation Study

Here we assess the contributions of the different components in RetinaQA. First, we remove (one at a time) the three key components: the logical form integrator (LFI), the logical form retriever (LFR), and the coupled sketch generator (SG) and schema retriever (SR) combined. The last three rows of Table 1 shows that at the aggregate level all components contribute towards RetinaQA's performance on GrailQAbility to different extents for answerable and unanswerable questions.

Next, we drill down into specific question categories. First, we study the *recall* of the correct logical form within the candidate set for unanswerable questions with missing data elements. If we remove SR and SG, the resulting RetinaQA ablation only retrieves candidate logical forms via traversal. We find that removing SR and SG results in a massive 65 pct point drop in recall. In contrast, removing LF retriever does not hurt much (see Table 10 in appendix). This agrees with our intuition that when relevant data is missing, traversal necessarily retrieves irrelevant logical forms.

Next, we study the impact of traversal-dependent logical form retrieval on the recall of the right logical form for answerable questions. Unsurprisingly, removing LFR (and also SR+SG) results in a substantial drop in recall (Table 11 in appendix). Also, LFR has significantly impact for the zero-shot generalization subset of answerable questions. For question forms unseen during training, KB-traversal is the only reliable approach for retrieving logical forms.

Finally, we evaluate the impact of the logical form integrator (LFI) and execution guided checking (EGC) in reducing the space of logical form candidates for the discriminator, by pruning out invalid logical forms, in the answerable setting in GrailQA. By switching off LFI and EGC separately, we see about 4 pct points and 2 pct point performance drops respectively. However, on switching off both together, a 17 pct point drop is observed (Table 9 in appendix). This suggests that these components can compensate for each other, but at least one of them is needed for good performance.

4.5 Error Analysis

We now briefly report a summary of error analysis for RetinaQA on GrailQAbility. For this, we use the A+U training with thresholding version which is the most robust. All errors can be classified into three categories: (1) *thresholding error*, where, due to thresholding, RetinaQA incorrectly predicts "NK" for a question with a valid logical form; (2) *reranking error*, where even though the correct logical form is present in the pool of candidates, the discriminator makes a mistake in scoring; and, (3) *recall error*, where the correct logical form is not in the set of candidates considered by the discriminator due to errors in the earlier stages. This may include errors in entity linking, logical form retrieval or logical form construction (via sketch generation and schema retrieval).

On the subset of answerable questions, thresholding and reranking errors occur in around 37.67%, and 30% of questions, respectively. The most frequent error is however recall error (70%). Among these, entity linking errors occur 80% of the time. This clearly suggests that improving entity linker can significantly improve the overall performance of KBQA models. Unsurprisingly, the majority of the errors of all categories occur for the zero-shot generalization questions. Detailed statistics are in Table 7 in appendix.

For unanswerable questions, we first look at those with missing data elements. We find that

Train	Model	IID			Compositional			Zero-Shot		
		F1(L)	F1(R)	EM	F1(L)	F1(R)	EM	F1(L)	F1(R)	EM
A	RnG	85.5	85.4	83.2	65.9	65.9	60.2	72.7	72.7	67.3
	Tiara	86.53	86.47	84.52	72.02	72.02	64.93	74.24	74.24	67.6
	Pangu	82	81.97	79.09	71.63	71.63	65.95	**77.02**	**77.02**	**70.18**
	RetinaQA	**87.94**	**87.9**	**85.85**	**73.92**	**73.92**	**67.48**	74.84	74.84	69.68
A+U	RnG	85.4	85.3	83.3	65.8	65.8	60.8	66.9	66.9	62.6
	Tiara	82.38	82.36	80.57	65.16	65.16	59.84	58.5	58.5	54.65
	Pangu	81.08	81.01	76.85	**77.43**	**77.43**	**69.52**	**78.01**	**78.01**	**70.42**
	RetinaQA	**89**	**88.98**	**87.06**	71.69	71.69	65.55	73.59	73.59	67.51

Table 3: Performance of different models for answerable questions in the GrailQAbility dataset, for IID, compositional, and zero-shot test scenarios. Names have the same meanings as in Tab. 1.

Model	Overall		IID		Compositional		Zero-Shot	
	F1	EM	F1	EM	F1	EM	F1	EM
RnG	85.5	83.2	85.5	83.2	65.9	60.2	72.7	67.3
Tiara	81.9	75.3	**91.2**	88.4	74.8	66.4	80.7	73.3
Pangu	82.16	75.9	86.38	81.73	76.12	68.82	**82.82**	**76.29**
RetinaQA	**83.33**	**77.84**	**91.22**	**88.58**	**77.49**	**70.48**	82.32	76.2

Table 4: Performance of different models on GrailQA (validation set) (which has only answerable questions) for IID, compositional, and zero-shot test scenarios.Note that we beat previous SOTA on GrailQA.

the vast majority of errors (around 90%) are recall errors, out of which about 72% are attributable to the entity linker. This occurs when mentioned entities in the question are missing in the KB, but entity linker outputs spurious entities. Thresholding error accounts for 45% of errors, while reranking errors only occurs in 5% of questions. This suggests that the discriminator is calibrated well for relative ranking of logical forms, but still makes many mistakes in assigning correct absolute scores to logical forms. See Table 8 in appendix for more details.

Finally, we look at the subset of unanswerable questions with missing schema elements. Since for these, the gold logical form is "NK", thresholding error can be only source of error. This occurs only 14% of time, out of which 90% errors occur for zero-shot generalisation. This indicates that model is largely good in this setting, but makes some mistakes in absolute scoring of logical forms with schema elements not seen during training.

5 Conclusions

We have presented RetinaQA, the first specialized model for KBQA that is robust for both answerable and unanswerable questions. RetinaQA achieves this robustness by unifying key aspects of KBQA models used for answerable-only iid and transfer settings so that candidate logical forms are arrived at by KB-traversal based retrieval, as well as traversal-independent generation via sketch-filling that bridges over data gaps that break traversal. RetinaQA also discriminates between fully formed candidate logical forms at the final stage instead of generating these. This enables it to better differentiate between valid and invalid logical forms. To demonstrate this robustness, we show that RetinaQA performs well for both answerable and unanswerable questions on the GrailQAbility dataset, with and without unanswerability training. Unanswerability performance improves with thresholding and unanswerability training, and it comes with minimal drop in performance for answerable questions. Performance is stable across IID, zero-shot and compositional splits for answerable questions, as well as IID and zero-shot splits for unanswerable questions. By comparing against state-of-the-art KBQA models adapted for answerability, we show that RetinaQA significantly outperforms these models for unanswerable questions while performing almost at par for answerable ones. Remarkably, RetinaQA also achieves a new state-of-the-art performance on the answerable-only GrailQA dataset, demonstrating the strengths of its architecture for KBQA in general. Further we plan to make our code-base [3] public for the community.

6 Limitations

A sketch, while free of references to the KB, still specifies the length of the path to be traversed in

[3] https://anonymous.4open.science/r/RETINAQA-122B/

the KB. The subsequent grounding step is limited by this and cannot adapt the path length after retrieving schema elements from the KB. RetinaQA inherits this limitation from existing sketch generation approaches (Cao et al., 2022; Ravishankar et al., 2022). We hope to improve this in future work.

For unanswerable questions without valid logical forms for the given KB, RetinaQA only outputs l =NK. However, this does not explain the gap in the schema, which, if bridged, would have make this question answerable. The situation is similar for unanswerable questions with valid logical forms but missing data elements. This is also an important area of future work.

7 Risks

Our work does not have any obvious risks.

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*.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. 2018. QuAC: Question answering in context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

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*.

Evgeniy Gabrilovich, Michael Ringgaard, and Amarnag Subramanya. 2013. Facc1: Freebase annotation of clueweb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0).

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.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization.

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*.

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. *J. Mach. Learn. Res.*, 21(1).

Vatsal Raina and Mark Gales. 2022. Answer uncertainty and unanswerability in multiple-choice machine reading comprehension. In *Findings of the Association for Computational Linguistics: ACL 2022*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

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*.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. CoQA: A Conversational Question Answering Challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266.

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)*.

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*.

Elior Sulem, Jamaal Hay, and Dan Roth. 2022. Yes, no or IDK: The challenge of unanswerable yes/no questions. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

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*.

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)*.

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)*.

Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2022. Rankt5: Fine-tuning t5 for text ranking with ranking losses.

Train	Model	IID		Zero-Shot	
		F1(R)	EM	F1(R)	EM
A+U	RnG	91.9	73.3	81.7	47.1
	RnG+T	94.3	75.9	85.9	59.5
	Tiara	93.76	75.22	86.35	50.84
	Tiara + T	95.1	77.77	87.86	56.88
	Pangu	21.4	12.17	22.32	15.32
	PAngu +T	62.04	57.52	56.94	52.85
	RetinaQA	64.59	36.43	65.44	43.4
	RetinaQA+ T	**97.01**	**89.94**	**88.31**	**75.22**

Table 5: Performance of different models for unanswerable IID and zero-shot test scenarios in GrailQAbility. Names have the same meanings as in Tab. 1.

Train	Model	Full Z-Shot		Partial Z-Shot	
		F1(R)	EM	F1(R)	EM
A+U	RnG	87.2	75.9	78	40
	RnG+T	89.7	86.7	**83.1**	71
	Tiara	90.15	68.97	80.25	40.45
	Tiara + T	**90.64**	78.82	82.64	54.14
	Pangu	24.63	20.69	21.18	15.76
	PAngu +T	89.66	**89.66**	79.94	**79.46**
	RetinaQA	57.64	25.12	43.47	11.31
	RetinaQA+ T	88.67	77.83	80.89	70.54

Table 6: Performance of different models for partial zero-shot and full-zero test scenarios in GrailQAbility. Names have the same meanings as in Tab. 1.

A Appendix

Entity Linker: We use an off-the-shelf entity linker (Ye et al., 2022) previously used in the KBQA literature (Shu et al., 2022; Gu et al., 2023), which uses a standard 3-staged pipeline - Mention Detection, Candidate Generation, and Entity Disambiguation. Mention Detector first identifies span of text from question which corresponds to name of an entity. For each mention a set of candidates entities are generated using alias mapping of FACC1 (Gabrilovich et al., 2013). Final stage is a neural disambiguator which rank candidates given the question and context of entities.

A.1 Implementation Details

To perform experiments for GrailQAbility, we first update the original Freebase KG using codebase[4]. To test baselines for GrailQAbility, we use the existing codebases[5] [6] [7] and make changes in code to adapt for answer-ability detection. All of the baselines assumes answerability and employs Execution Guide Check i.e. if a logical form returns

[4]https://github.com/dair-iitd/GrailQAbility

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

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

[7]https://github.com/salesforce/rng-kbqa

Components	Overall	IID	Compositional	Zero-shot
#questions	6808	3386	981	2441
#errors	1691	445	347	899
thresholding_error	637	161	113	363
reranking_error	508	49	134	325
coverage_error	1183	396	213	574
entity_linking_error	949	343	136	470
schema_retriever_error	460	61	77	322
sketch_parser_error	420	43	154	22

Table 7: Component wise errors of RetinaQA+ T (A+U) for answerable questions

Components	Overall	IID	Zero-shot
#questions	1196	530	666
#errors	287	127	160
thresholding_error	131	59	72
reranking_error	16	5	11
coverage_error	271	122	149
entity_linking_error	195	77	118
schema_retriever_error	56	29	27
sketch_parser_error	49	27	22

Table 8: Component wise errors of RetinaQA+ T (A+U) for data element missing unanswerable questions

Model	Overall		IID		Compositional		Zero-Shot	
	F1	EM	F1	EM	F1	EM	F1	EM
RetinaQA’	83.33	77.84	91.22	88.58	77.49	70.48	82.32	76.2
RetinaQA’ - EGC	80.62	75.68	89.81	87.7	74.78	68.1	79.03	73.58
RetinaQA’ - LFI	78.65	73.1	88.1	84.81	75	67.31	76.04	70.4
RetinaQA’ - LFR	71.8	68.33	87.33	85.56	69	63.47	66.19	62.83
RetinaQA’ - (SG $\cup$ SR)	73.2	66.78	77.06	72.13	60.63	54.43	76.73	69.56
RetinaQA’ - LFI - EGC	63.29	59.99	79.84	77.84	59.33	54.03	57.73	54.68

Table 9: Ablation experiment on GrailQA dev set. EGC refers to Execution Guided Check and LFI refers to Logical Form Integrator, RetinaQA’ = RetinaQA + EGC

Model	Overall	IID	Zero-shot
RetinaQA	77.34	76.98	77.63
RetinaQA- LFR	77.17	76.79	77.48
RetinaQA- SP - SR	12.29	10	14.11

Table 10: Ablation experiment of Logical Form Coverage(%) on GrailQAbility test set. LFR refers to Logical Form Retriever, SP refers to Sketch Parser and SR refers to Schema Retriever.

Model	Overall	IID	Compositional	Zero-shot
RetinaQA	82.62	88.3	78.29	76.49
RetinaQA- LFR	74.24	85.91	67.38	60.79
RetinaQA- SP - SR	71.94	74.22	65.24	71.49

Table 11: Ablation experiment of Logical Form Coverage(%) on GrailQAbility test set for Answerable questions. LFR refers to Logical Form Retriever, SP refers to Sketch Parser and SR refers to Schema Retriever.

an empty answer upon execution then they select next best logical form. We have removed this constraint while performing experiments for GrailQAbility. Also for A+U training we have made code changes so that models can be trained to predict logical form as NK unanswerable questions. We implement our model using Pytorch (Paszke et al., 2019) and Hugging Face[8]. All the experiments of RetinaQA are performed using an NVIDIA A100 GPU with 80 GB RAM. Above mentioned configurations are the maximum ones, since we have different components and all do not require same compute configurations. For Sketch Generation we fine tune Seq2Seq t5-base model for 10 epochs

[8]https://huggingface.co/

(fixed). We use learning rate of 3e-5 and batch size of 8. We use beam search during decoding with $beamsize = 10$. We also check syntactic correctness while selecting top ranked sketch. Training time for sketch parser is around 3 hours. LF Integrator is a parameter free module and does not require any training. Since, LF Integrator converts logical forms into query-graphs and validates type-level constraints, it is a costly operation. So we employ parallel processing(with cache) for this stage i.e. we use 4-6 CPUs (each with 2 cores) to create pool of valid logical forms. It takes around 5 hours to generate candidates for all train, dev and test data. Finally we train Discriminator which fine-tune t5-base Seq2Seq model. We train Discriminator with learning rate 1e-4 and batch size 4 for 10 epochs. For discriminator training we use AdmaW (Loshchilov and Hutter, 2019) optimizer and linear scheduler with warm up ratio of 0.01. We use 64 negative samples per question for contrastive training. Generally discriminator model converges in 2 epochs of training so we use patience of 2 i.e. if best model does not change for consequent 2 epochs then we assume model has converged and will stop training. It takes around 7-8 hours to train a discriminator. Inference time for discriminator is few minutes.

For A+U training components like Entity Linker, Schema Retriever, LF Retriever are trained only on question where logical form is known. While training for questions with $l =$"NK" is performed only at last step.

All the results presented for single run (however the reproducbility of results is already verified). We also plan to release our code-base[9] for the community.

A.2 In Depth Trade-off Analysis

Sec 4.4 describes how individual components strengthens performance for different types of answerabilties and unanswerabilties. This section discusses an important trade-off i.e. **Traversal dependent Retrieval Vs Traversal independent Retrieval** : Traversal based Retrieval methods perform step by step enumeration over KB to retrieve next possible set of candidates(which is retrieval at data level). While Traversal independent Retrieval based method generate candidates based on semantic similarity with the question(which is at schema level). So for Data Element Missing unanswerability where data paths are missing, Traversal based methods will never find correct path during enumeration and hence will not be able to reach to a correct logical form. While Traversal independent method can generate correct logical form. Hence Traversal independent methods performs well for data element missing.
At the same time the search space for Traversal independent methods is much larger as it lacks KB grounding information. So for zero-shot generalisation where schema elements are unseen Traversal dependent tends to get confused between similar schema elements.

[9]https://anonymous.4open.science/r/RETINAQA-122B