# Expanding End-to-End Question Answering on Differentiable Knowledge Graphs with Intersection

**Priyanka Sen**
Amazon Alexa AI
Cambridge, UK
sepriyan@amazon.com

**Amir Saffari**
Amazon Alexa AI
Cambridge, UK
amsafari@amazon.com

**Armin Oliya**
Amazon Alexa AI
Cambridge, UK
aooliya@amazon.com

## Abstract

End-to-end question answering using a differentiable knowledge graph is a promising technique that requires only weak supervision, produces interpretable results, and is fully differentiable. Previous implementations of this technique (Cohen et al., 2020) have focused on single-entity questions using a relation following operation. In this paper, we propose a model that explicitly handles multiple-entity questions by implementing a new *intersection* operation, which identifies the shared elements between two sets of entities. We find that introducing intersection improves performance over a baseline model on two datasets, WebQuestionsSP (69.6% to 73.3% Hits@1) and ComplexWebQuestions (39.8% to 48.7% Hits@1), and in particular, improves performance on questions with multiple entities by over 14% on WebQuestionsSP and by 19% on ComplexWebQuestions.

## 1 Introduction

Knowledge graphs (KGs) are data structures that store facts in the form of relations between entities. Knowledge Graph-based Question Answering (KGQA) is the task of learning to answer questions by traversing facts in a knowledge graph. Traditional approaches to KGQA use semantic parsing to parse natural language to a logical query, such as SQL. Annotating these queries, however, can be expensive and require experts familiar with the query language and KG ontology.

End-to-end question answering (E2EQA) models overcome this annotation bottleneck by requiring only weak supervision from question-answer pairs. These models learn to predict paths in a knowledge graph using only the answer as the training signal. In order to train an E2EQA model in a fully differentiable way, Cohen et al. (2020) proposed differentiable knowledge graphs as a way to represent KGs as tensors and queries as differentiable mathematical operations.

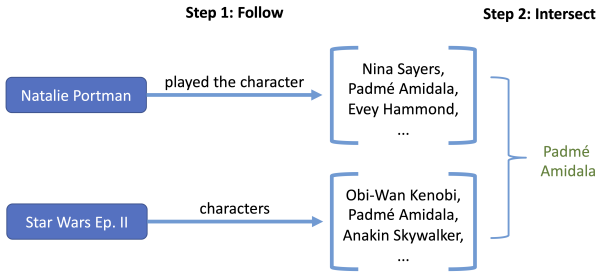


Figure 1: To answer "*Who did Natalie Portman play in Star Wars Episode II?*", we identify all the characters Natalie Portman has played, all the characters in Star Wars Episode II, and intersect the two resulting sets to get to the answer, Padmé Amidala.

Previous implementations of E2EQA models using differentiable knowledge graphs (Cohen et al., 2020) have focused on single-entity questions using a relation following operation. For example, to answer "*Where was Natalie Portman born?*", the model could predict a path starting at the Natalie Portman entity and following a *place of birth* relation to the correct answer.

While this follow operation handles many questions, it often struggles on questions with multiple entities. For example, to answer "*Who did Natalie Portman play in Star Wars Episode II?*", it is not enough to identify all the characters Natalie Portman has played, nor all the characters in Star Wars Episode II. Instead, the model needs to find what character Natalie Portman has played that is also a character in Star Wars. This can be solved through **intersection**. An intersection of two sets $A$ and $B$ returns all elements in $A$ that also appear in $B$. This example is illustrated in Figure 1.

In this paper, we propose to explicitly handle multiple-entity questions in E2EQA by learning intersection in a dynamic multi-hop setting. Our intersection models learn to both follow relations and intersect sets of resulting entities in order to arrive at the correct answer. We find that our mod-



*Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8805–8812
November 7–11, 2021. 

els score 73.3% on WebQuestionsSP and 48.7% on ComplexWebQuestions, and in particular, improve upon a baseline on questions with multiple entities from 56.3% to 70.6% on WebQuestionsSP and 36.8% to 55.8% on ComplexWebQuestions.

## 2 Related Works

Traditional approaches to KGQA have used semantic parsing (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005) to parse natural language into a logical form. Collecting semantic parsing training data can be expensive and is done either manually (Dahl et al., 1994; Finegan-Dollak et al., 2018) or using automatic generation (Wang et al., 2015) which is not always representative of natural questions (Herzig and Berant, 2019).

Another line of work in KGQA uses embedding techniques to implicitly infer answers from knowledge graphs. These methods include GRAFT-Net (Sun et al., 2018), which uses a graph convolutional network to infer answers from subgraphs, PullNet (Sun et al., 2019), which improves GRAFT-Net by learning to retrieve subgraphs, and EmbedKGQA (Saxena et al., 2020), which incorporates knowledge graph embeddings. EmQL (Sun et al., 2020) is a query embedding method using set operators, however these operators need to be pretrained for each KB. TransferNet (Shi et al., 2021) is a recent model that trains KGQA in a differentiable way, however it stores facts as an *N* x *N* matrix, where *N* is the number of entities, so it runs into scaling issues with larger knowledge graphs.

Our approach to KGQA based on Cohen et al. (2020) has three main advantages:

- **Interpretability**: Models based on graph convolutional networks (PullNet, GRAFT-Net) get good performance but have weak interpretability because they do not output intermediate reasoning paths. Our approach outputs intermediate paths as well as probabilities.

- **Scaling**: Cohen et al. (2020) show that differentiable KGs can be distributed across multiple GPUs and scaled horizontally, so that different triple IDs are stored on different GPUs, allowing for scaling to tens of millions of facts. Other methods using embedding techniques (EmbedKGQA, EmQL) or less efficient representations (TransferNet) are more memory intensive and not easily distributed.

- **No retraining for new entities**: Models based on latent representations of entities (EmbedKGQA, EmQL) get state-of-the-art performance, however they need to be retrained whenever a new entity is added to the KG (e.g., a new movie) to learn updated embeddings. Our approach can incorporate new entities easily without affecting trained models.

## 3 Models

### 3.1 The Baseline Model

Our baseline model, which we call Rigel-Baseline, is based on differentiable knowledge graphs and the ReifiedKB model (Cohen et al., 2020). We provide an overview here but full details can be found in the original paper.

#### 3.1.1 Differentiable Knowledge Graphs

Assume we have a graph:

$$\mathcal{G} = \big\{(s, p, o) \,|\, s \in E, o \in E, p \in R\big\}, \qquad (1)$$

where $E$ is the set of entities, $R$ is the set of relations, and $(s, p, o)$ is a triple showing that the relation $p$ holds between a subject entity $s$ and an object entity $o$. To create a differentiable knowledge graph, we represent the set of all triples $T = \{t_i\}_{i=1}^{N_T}, \quad t_i = (s_{s_i}, p_{p_i}, o_{o_i})$ in three matrices: a subject matrix ($M_s$), relation matrix ($M_p$), and object matrix ($M_o$). A triple $(s, p, o)$ is represented across all three matrices at a given index.

$$M_s \in \{0,1\}^{N_T \times N_E}, \quad M_s(i,j) = \mathbb{I}\big(e_j = s_{s_i}\big)$$
$$M_p \in \{0,1\}^{N_T \times N_R}, \quad M_p(i,j) = \mathbb{I}\big(p_j = p_{p_i}\big)$$
$$M_o \in \{0,1\}^{N_T \times N_E}, \quad M_o(i,j) = \mathbb{I}\big(e_j = o_{o_i}\big)$$

Since the knowledge graph is represented as matrices, interacting with the knowledge graph is done with matrix operations. ReifiedKB was implemented with a follow operation: Given an entity vector $\mathbf{x}_{t-1} \in \mathbb{R}^{N_E}$ at $t-1$-th time step and a relation vector $\mathbf{r}_t \in \mathbb{R}^{N_R}$, the resulting entity vector $\mathbf{x}_t$ is computed by Equation 2 where $\odot$ is element-wise multiplication.

$$\mathbf{x}_t = \text{follow}(\mathbf{x}_{t-1}, \mathbf{r}_t) = M_o^T (M_s \mathbf{x}_{t-1} \odot M_p \mathbf{r}_t) \qquad (2)$$

#### 3.1.2 Model

The Rigel-Baseline model is composed of an encoder, which encodes the question, and a decoder, which returns a probability distribution over KG

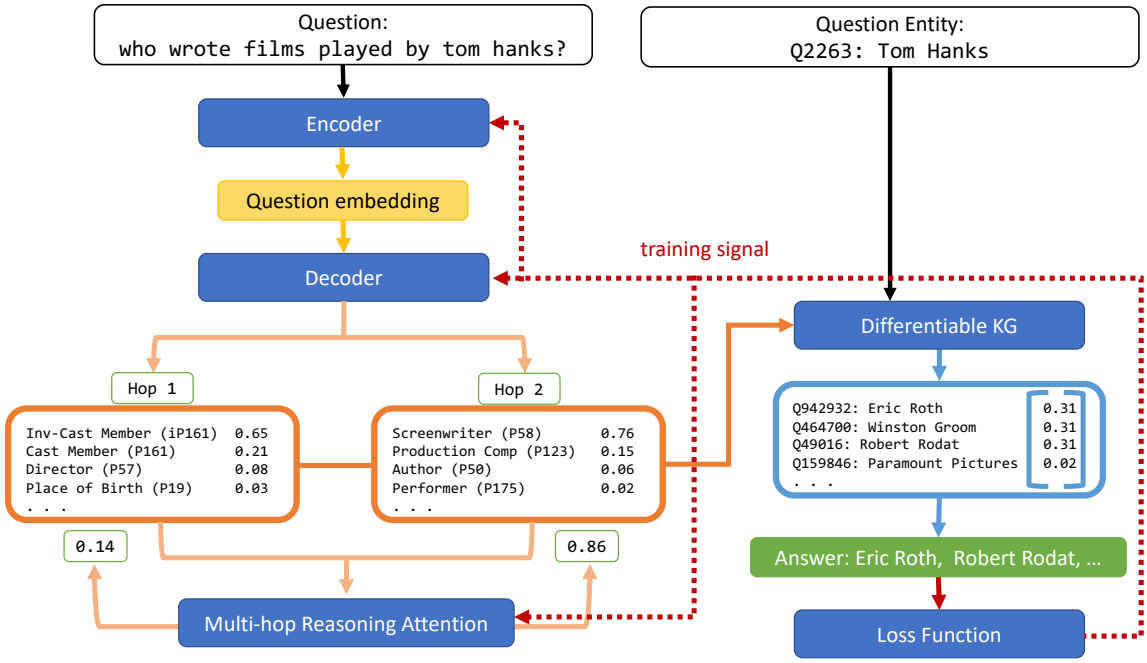


Figure 2: A detailed illustration of the Rigel-Baseline model. The encoder encodes the question, the decoder predicts relations, and attention selects the final hop. The entities and relations are followed in the KG to return predicted answers. The loss between the predicted and actual answers is the training signal for the whole model.

relations. The question entities (which, in our experiments, are provided from the datasets) and predicted relations are followed in the differentiable knowledge graph to return predicted answers. Predicted answers are compared to labeled answers, and the loss is used to update the model. Rigel-Baseline is illustrated in Figure 2.

We make the following key improvements to ReifiedKB. First, we use RoBERTa (Liu et al., 2019) as our encoder instead of word2vec. Second, ReifiedKB used different methods to determine the correct number of hops in multi-hop questions. We implement an attention mechanism using a hierarchical decoder $W_t^{dec}$, which is learned and a unified approach across datasets. Given a question embedding $\mathbf{h}_q$ and relation vector $\mathbf{r}_t$, the resulting entity vector $\mathbf{x}_t$ is computed as:

$$\mathbf{r}_t = \text{softmax}\left(W_t^{dec}\left[\mathbf{h}_q|\mathbf{r}_{t-1}|\cdots|\mathbf{r}_1\right]^T\right) \quad (3)$$

$$\mathbf{x}_t = \text{follow}(\mathbf{x}_{t-1}, \mathbf{r}_t) \quad (4)$$

We compute an attention score across all hops with:

$$c_t = W_t^{att}\left[\mathbf{h}_q|\mathbf{r}_{t-1}|\cdots|\mathbf{r}_1\right]^T \quad (5)$$

$$\mathbf{a} = \text{softmax}([c_1, \cdots, c_{T_h}]) \quad (6)$$

and compute the final estimate $\hat{\mathbf{y}}$ as:

$$\hat{\mathbf{y}} = \sum_{t=1}^{T_h} a_t \mathbf{x}_t. \quad (7)$$

Finally, while ReifiedKB used cross-entropy as its loss function, we instead use a multi-label loss function across all entities. This is because the output space in many samples contains multiple entities, so cross-entropy loss is inadequate.

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N_E}\sum_{i=1}^{N_E} y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i) \quad (8)$$

### 3.2 The Intersection Model

In order to build a model that can handle multiple entities, we expand Rigel-Baseline with a differentiable intersection operation to create Rigel-Intersect. We define intersection as the element-wise minimum of two vectors. While differentiable intersection has previously been implemented as element-wise multiplication (Cohen et al., 2019), we prefer to use minimum since it prevents diminishing probabilities. Given two vectors $a$ and $b$, the element-wise minimum ($\min_{elem}$) returns a minimum (min) at each index where $\min(a_n, b_n)$ will return $a_n$ if $a_n < b_n$, or $b_n$ if $b_n < a_n$. Any element that appears in both vectors returns a non-zero

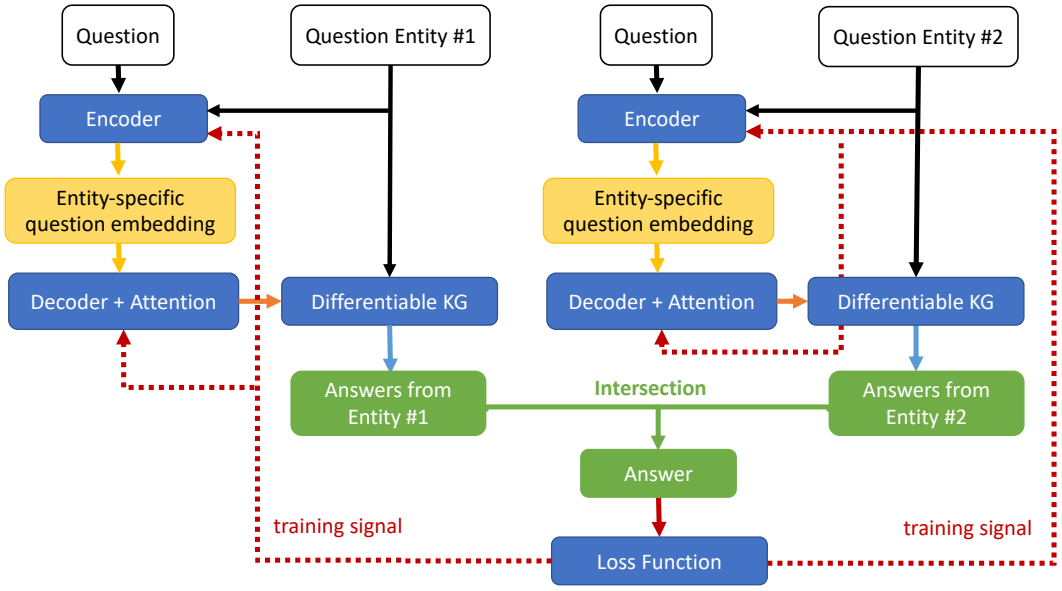


Figure 3: An illustration of the Rigel-Intersect model. Given a question with two entities, we run each question entity in parallel to return intermediate answers. These answers are intersected to return the final answer. The loss between the final answer and the actual answer is the training signal for the whole model.

value, and elements that appears in one or neither vector return a 0.

$$\min_{elem}\left(\begin{bmatrix}a_1\\a_2\\...\\a_n\end{bmatrix}, \begin{bmatrix}b_1\\b_2\\...\\b_n\end{bmatrix}\right) = \begin{bmatrix}\min(a_1,b_1)\\\min(a_2,b_2)\\...\\\min(a_n,b_n)\end{bmatrix} \quad (9)$$

Next, we modify the encoder to allow entity-specific question embeddings. The Rigel-Baseline encoder creates one generic question embedding per question, but we may want to follow different relations for each entity. To calculate the question embedding $\mathbf{h_q}$ using an encoder $f_q$, for each question entity, we concatenate the question text $q$ with the entity's mention or canonical name $m$ separated by a separator token [SEP]. We use the embedding at the separator token index $i_{SEP}$ as the entity-specific representation of the question.

$$\mathbf{h_q} = f_q(q \text{ [SEP] } m)[:, i_{SEP}, :] \quad (10)$$

In the decoder, we predict inference chains in parallel for each entity, and follow the entities and relations in the differentiable KG to return intermediate answers. We intersect the two intermediate answers to return the final answer. In multi-hop settings, we weight entities in each vector before intersection based on the attention score. We train using the hyperparameters in Appendix A. Rigel-Intersect is illustrated in Figure 3.

Our implementation of intersection takes a maximum of two entities per question. We use the first two labeled entities per question and ignore subsequent ones. This works well for our datasets where 94% of questions contain a maximum of two entities. Given a dataset with more complex questions, we can extend our implementation in the future to an arbitrary number of entities.

## 4 Datasets

We use two datasets in our experiments:

**WebQuestionsSP** (Yih et al., 2016) is an English question-answering dataset of 4,737 questions (2,792 train, 306 dev, 1,639 test) that are answerable using Freebase (Bollacker et al., 2008). During training, we exclude 30 training set questions with no answer, and during evaluation, we exclude 13 test set questions with no answer and count them as failures in our Hits@1 score. All questions are answerable by 1 or 2 hop chains of inference, so we set our models to a maximum of 2 hops. To create a subset of Freebase, we identify all question entities and relations in WebQuestionsSP and build a subgraph containing all facts reachable from 2 hops of the question entities, as done in previous works (Cohen et al., 2020). This creates a sub-

graph of 17.8 million facts, 9.9 million entities, and 670 relations. We create inverse relations for each relation (e.g., *place of birth* returns a person's birthplace; *inv-place of birth* returns people born in that location), for a total of 1,340 relations.

**ComplexWebQuestions** (Talmor and Berant, 2018) is an extended version of WebQuestionsSP with 34,689 questions (27,649 train, 3,509 dev, 3,531 test) in English requiring complex reasoning. During training, we exclude 163 questions that are missing question or answer entities, and during evaluation, we exclude 21 examples from the test set for the same reasons and count them as failures in the Hits@1 score. We limit our model to 2 hops for training efficiency. To create a subset of Freebase, we identify all question entities and relations in the dataset and build a subgraph containing all facts reachable within 2 hops of the question entities. This results in a subgraph of 43.2 million facts, 17.5 million entities, and 848 relations (1,696 including inverse relations).

## 5 Results

| Model | WQSP | CWQ |
|---|---|---|
| KVMem (Miller et al., 2016) | 46.7 | 21.1 |
| GRAFT-Net (Sun et al., 2018) | 67.8 | 32.8 |
| PullNet (Sun et al., 2019) | 68.1 | 47.2 |
| ReifiedKB (Cohen et al., 2020) | 52.7 | – |
| EmQL (Sun et al., 2020) | **75.5** | – |
| TransferNet (Shi et al., 2021) | 71.4 | 48.6 |
| Rigel-Baseline (ours) | 69.6 | 39.8 |
| Rigel-Intersect (ours) | 73.3 | **48.7** |

Table 1: Comparison of Hits@1 results on WebQuestionsSP (WQSP) and ComplexWebQuestions (CWQ)

| Dataset | Baseline | Intersect |
|---|---|---|
| WebQSP | 69.6 | 73.3 |
| WebQSP 1 Entity | 75.8 | 75.3 |
| WebQSP >1 Entity | 56.3 | **70.6** |
| CWQ | 39.8 | 48.7 |
| CWQ 1 Entity | 43.1 | 42.7 |
| CWQ >1 Entity | 36.8 | **55.8** |

Table 2: Hits@1 breakdown by number of entities for Rigel-Baseline and Rigel-Intersect

Our results are in Tables 1 and 2. Scores are reported as Hits@1, which is the accuracy of the top predicted answer from the model. Table 1 compares our scores to previous models. The aim of our paper is to show an extension of a promising KGQA technique, not to produce state-of-the-art results, but this table shows that Rigel-Intersect is competitive with recent models. Our improved Rigel-Baseline scores higher than ReifiedKB on WebQuestionsSP at 69.6%, and Rigel-Intersect improves upon that at 73.3%. On ComplexWebQuestions, Rigel-Baseline scores lower than recent methods at 39.8%, but Rigel-Intersect gets competitive results with 48.7%.

The breakdown of results in Table 2 shows that the improved performance of Rigel-Intersect comes from better handling of questions with multiple entities. While Rigel-Baseline and Rigel-Intersect are comparable on questions with one entity, Rigel-Intersect surpasses Rigel-Baseline on questions with more than 1 entity by over 14% on WebQuestionsSP (56.3% vs. 70.6%) and by 19% on ComplexWebQuestions (36.8% vs. 55.8%). Example model outputs are in Appendix C.

Rigel-Baseline is not incapable of handling multiple-entity questions because not all questions require intersection. For example, in "*Who played Jacob Black in Twilight?*", the model can follow *Jacob Black* to the actor, *Taylor Lautner*, without intersecting with *Twilight* because only one actor has played Jacob Black. This is not possible for characters such as James Bond or Batman, who are portrayed by different actors in different movies. Although Rigel-Baseline can spuriously handle multiple-entity questions, Rigel-Intersect uses more accurate inference chains.

## 6 Conclusions

In this paper, we expand an end-to-end question answering model using differentiable knowledge graphs to learn an intersection operation. We show that introducing intersection improves performance on WebQuestionsSP and ComplexWebQuestions. This improvement comes primarily from better handling of questions with multiple entities, which improves by over 14% on WebQuestionsSP, and by 19% on ComplexWebQuestions. In future work, we plan to expand our model to more operations, such as union or difference, to continue improving model performance on complex questions.

## References


Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1247–1250.

William W. Cohen, Matthew Siegler, and Alex Hofer. 2019. Neural query language: A knowledge base query language for Tensorflow. *arXiv preprint arXiv:1905.06209*.

William W. Cohen, Haitian Sun, R. Alex Hofer, and Matthew Siegler. 2020. Scalable neural methods for reasoning with a symbolic knowledge base. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, page 43–48, USA. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Jonathan Herzig and Jonathan Berant. 2019. Don't paraphrase, detect! rapid and effective data collection for semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3810–3820, Hong Kong, China. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

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

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

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. 2021. TransferNet: An effective and transparent framework for multi-hop question answering over relation graph. *arXiv preprint arXiv:2104.07302*.

Haitian Sun, Andrew Arnold, Tania Bedrax Weiss, Fernando Pereira, and William W. Cohen. 2020. Faithful embeddings for knowledge base queries. *Advances in Neural Information Processing Systems*, 33.

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

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4231–4242, Brussels, Belgium. Association for Computational Linguistics.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. *arXiv preprint arXiv:1803.06643*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342, Beijing, China. Association for Computational Linguistics.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, page 1050–1055. AAAI Press.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured

classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, page 658–666, Arlington, Virginia, USA. AUAI Press.

## A Model Hyperparameters

We train Rigel-Baseline and Rigel-Intersect using the hyperparameters shown below. For WebQuestionsSP, we train on a single 16GB GPU. Training completes in approximately 12 hours. ComplexWebQuestions is a larger dataset with a larger knowledge graph, so we train on 4 16GB GPUs, with the knowledge graph distributed across 3 GPUs and the model on the fourth GPU. Training completes in approximately 40 hours.

| Hyperparameter | WQSP | CWQ |
|---|---|---|
| Batch Size | 4 | 2 |
| Gradient Accumulation | 32 | 64 |
| Training Steps | 40000 | 80000 |
| Learning Rate | 1e-4 | 1e-4 |
| Max Number of Hops | 2 | 2 |

Table 3: Hyperparameters for training on WebQuestionsSP (WQSP) and ComplexWebQuestions (CWQ)

## B Validation and Test Performance

The table below shows the corresponding validation performances on the dev set for each test result.

| Dataset | Model | Dev | Test |
|---|---|---|---|
| WQSP | Rigel-Baseline | 72.9 | 69.6 |
| | Rigel-Intersect | 77.1 | 73.3 |
| CWQ | Rigel-Baseline | 40.8 | 39.8 |
| | Rigel-Intersect | 48.1 | 48.7 |

Table 4: Validation and test performance in terms of Hits@1 for WebQuestionsSP (WQSP) and ComplexWebQuestions (CWQ)

## C Examples

The table on the following page shows example outputs of Rigel-Baseline and Rigel-Intersect. We only show the top predicted inference chain for each question, but in practice, a probability distribution over all relations is returned. The model also predicts how many hops to take based on an attention score. In our examples, if the model predicts one hop, we show only the top relation from the first hop. If the model predicts two hops, then we show the top relations for both hops.

The first two examples are questions that Rigel-Intersect handles well. In both of these questions, there are multiple probable answers if only one entity is followed (i.e., Russell Wilson attended multiple educational institutions; Michael Keaton has played multiple roles). However only one answer is correct if all entities are considered. The final question is an example where Rigel-Baseline answers correctly even though there are two entities. This is because there is only one winner of the 2014 Eurocup Finals Championship, which Rigel-Baseline can identify without needing to check if the team is from Spain.

**Q:** What educational institution with The Badger Herald newspaper did Russell Wilson go to?
**A:** University of Wisconsin–Madison

**Rigel-Baseline**
Russell Wilson → people.person.education → education.education.institution
→ Collegiate School ✗

**Rigel-Intersect**
Russell Wilson → people.person.education → education.education.institution
The Badger Herald → <inv>-education.educational_institution.newspaper
*Intersection* → University of Wisconsin–Madison ✓

---

**Q:** Who does Michael Keaton play in Cars?
**A:** Chick Hicks

**Rigel-Baseline**
Michael Keaton → film.actor.film → film.performance.character
→ Birdman ✗

**Rigel-Intersect**
Michael Keaton → film.actor.film → film.performance.character
Cars → film.film.starring → film.performance.character
*Intersection* → Chick Hicks ✓

---

**Q:** Which popular sports team in Spain won the 2014 Eurocup Finals Championship?
**A:** Valencia BC

**Rigel-Baseline**
2014 Eurocup Finals Championship → sports.sports_championship_event.champion
→ Valencia BC ✓

**Rigel-Intersect**
Spain → <inv>-sports.sports_team.location
2014 Eurocup Finals Championship → sports.sports_championship_event.champion
*Intersection* → Valencia BC ✓

Table 5: Example outputs of Rigel-Baseline and Rigel-Intersect