NEURAL REASONING FOR QUESTION ANSWERING

Haitian Sun

April 2023
CMU-ML-23-100

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee

Ruslan Salakhutdinov, Chair
William W. Cohen
Emma Strubell
Hannaneh Hajishirzi (University of Washington)

Submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

This research was sponsored by the National Science Foundation award IIS1763562 and Office of Naval Research award N000141812861.

Keywords: Machine Learning, Natural Language Processing, Question Answering, Language Models, Knowledge Representation, Knowledge Bases, Neural-Symbolic Machine Learning

To my family, friends, and everyone who helped me

Abstract

Question Answering (QA) is a very challenging task in Natural Language Processing (NLP) because it requires understanding questions, finding relevant information about the questions, and performing various reasoning steps to predict answers. Many types of reasoning are associated with questions people ask everyday. In this thesis, we discuss several methods for the challenging reasoning tasks in Question Answering (QA).

Reasoning tasks commonly seen in Question Answering (QA) include single-hop and multi-hop relation following, intersection and union, negation, and constraint verification. In the first part of this thesis, we study these reasoning tasks with structured or semi-structure queries over symbolic knowledge bases (KBs). We first propose a neural query language for reasoning in symbolic space, and then discuss possibilities to extend it to an embedding space to allow better generalization. Since symbolic KBs are commonly incomplete, we also propose a method to construct virtual KBs (VKBs) from text that support most reasoning tasks as symbolic KBs.

Since most NLP systems are built on language models (LMs), in the next part of the thesis, we propose methods to integrate reasoning methods into language models (LMs) to improve LMs' ability in performing the reasoning steps for more challenging QA tasks. The integration improves LMs' faithfulness to factual knowledge, and also enables updating the knowledge learned by LMs to make updated predictions without any additional training or fine-tuning. The proposed methods apply to both symbolic KBs and virtual KBs.

The reasoning tasks discussed previously, however, mainly focus on questions which are precisely defined, i.e. questions such that a single right answer exists. In the final part of this thesis, we study the QA task with ambiguous questions, i.e. where important information is missing from questions such that multiple answers are possible depending on how the question is interpreted. We developed a new dataset for this task and show it challenges current QA systems. We propose improved methods for the new dataset, that identify conditions which disambiguate the question based on analysis of a document that answers the question. Finally, we consider an "open" version of this task, where no answer document is provided.

Acknowledgments

We started this wonderful journey in Fall 2017. I still remembered the first time when Professor William Cohen introduced his research in NLP to me and lead me into the magical world of research. In the past six years, I strived to become a good NLP researcher. This achievement will not be possible without guidance, support, and love from so many people. I would like to sincerely thank everyone who taught and guided me, studied and learned with me, and now celebrate with me.

I would like to express my sincere gratitude to my advisor, Professor Ruslan Salakhutdinov, for his invaluable guidance, patience, and support throughout my PhD journey. His insightful comments, constructive feedback, and encouragement have been instrumental in shaping this work. I am also grateful to the members of my thesis committee, Professor William W. Cohen, Professor Emma Strubell, and Professor Hanna Hajishirzi, for their time, feedback, and insights. I would also like to thank my wonderful collaborators, Pat Verga, Bhuwan Dhingra, Alex Rofer, Livio Baldini Soares, Yuchen Lin, Wenhu Chen, Tania Bedrax-Weiss, Lidong Bing, and so many more people who have helped me in the past years.

I would like to express my deepest gratitude to my family and friends. I would like to thank my parents Kang Sun and Lanjie Li for their love, support, and encouragement. I would like to thank my partner, Jiachen Wang, for her unwavering support anywhere and anytime. I would like to extend my appreciation to my friends, Congxin Xu, Minying Gao, Xieyang Liu, Bannong Zhang, Jingwen Liao, Ming Yang, Sixu Piao, Xinyu Chen, Lai Wei, Shidong Li, Jinlei Chen, Chang Liu, Haonan Liu, Qiong Guo, Long Xue, Song Lv, and everyone who I met in the past few years. They bring me joy and happiness, and many unforgettable memories. Without their support, I would not have been able to achieve this milestone in my academic career. Last but not the least, I would like to give a special shout-out to my beloved cats, Topper, YY, and MuMu for their constant presence, purring, and playful antics that bring so much joy and comfort to my life.

In conclusion, this thesis would not have been possible without the support and encouragement of so many people. I am deeply grateful to each and every one of them for their contribution to my academic journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Building intelligent systems to understand natural language is a long-standing goal in the natural language processing (NLP) community. Natural language is used for communication in everyday life and storing knowledge about the world. However, developing machine learning technologies to understand natural language is challenging because of the diversity of languages in vocabulary and grammar and because of the rich semantic information in natural language. Fully understanding natural language also may require reasoning about the semantics of documents, or questions about documents.

We focus on knowledge-intensive tasks in NLP, such as Information Retrieval (IR) and Question Answering (QA), which requires understanding factual knowledge about the world. Knowledge-intensive tasks test NLP systems' ability to find relevant information from a large collection of knowledge and make predictions accordingly. For example, answers to the query "CMU's location" can be found from a piece of text from Wikipedia, e.g. "Carnegie Mellon University (CMU) is a private research university in *Pittsburgh, Pennsylvania*". Some other queries may need more than one piece of information, such as "tech companies near CMU", which requires first finding CMU's location and then companies locate in the same city, filtered by a constraint of being a "tech" company. We refer to the ability of understanding the intention of queries, locating relevant information, and aggregating multiple pieces of information to predict answers, as "*reasoning*". Various types of reasoning procedures are involved in knowledge-intensive NLP tasks.

In this thesis, we consider a few types of reasoning that are commonly seen in Question Answering (QA) tasks:

1. *Relational Following.* Relational following, e.g. "Departments at CMU", is the most commonly seen type of questions in QA. It starts with a topic entity x, e.g. "CMU", and follows a relation

r, e.g. "has_departments", to find answers. We can write relational following questions in first-order logic, e.g.

$$Y = \{y \mid \text{has\_department}(\text{CMU}, y)\}$$

where has_department$(\cdot, \cdot)$ is a predicate with the relation "departments at X" and "CMU" is the topic entity. Y is the set of answers y such that has_department(CMU, y) can be verified with the provided or retrieved information, e.g. $Y = \{\text{MLD}, \text{LTI}, \text{CSD}, \dots\}$.

2. *Multi-hop Relational Following.* Relational following can be chained if questions require more than one reasoning step. For example, "Degrees granted by departments at CMU" requires finding academic departments at CMU and then degrees granted by the departments, i.e. $Y = \{y \mid \exists\ z,\ \text{has\_department}(\text{CMU}, z) \wedge \text{degree}(z, y)\}$, where final answers $y \in Y$ depend on intermediate outputs z.

3. *Intersection and Union.* Two more common types of reasoning are intersection and union, e.g. "cities with CMU or UPitt campuses". To answer the question, we union the set of CMU's and UPitt's locations, i.e. $Y = \{y \mid \text{locate}(\text{CMU}, y) \vee \text{locate}(\text{UPitt}, y)\}$.

4. *Constraints.* Some questions require answers that satisfy some specified constraints, e.g. "cities with CMU's campuses that offer finance degrees". We filter the set of locations by the constraint, i.e. $Y = \{y \mid \text{locate}(\text{CMU}, y) \wedge \text{filter}(y, \text{"finance degrees"})\}$.

5. *Abductive Reasoning.* In abduction, the reasoning process is to derive explanations to a set of observations according to a few pre-determined rules. The explanation is only partially supported by the observations, and, therefore, it makes several assumptions. We consider the abductive reasoning task in a setting of answering ambiguous questions, where information provided in questions (sometimes paired with scenarios when questions are asked) are considered as observations. Since questions are ambiguous, i.e. important information for answering the questions is missing, rules are only partially satisfies. Therefore, multiple answers are possible and some answers may only be correct under certain conditions. We say the combination of plausible answers and their conditions are explanations given the provided information.

   The task of answering ambiguous questions is to find a plausible set of assumptions about the user's intent that will resolve the ambiguity, i.e. questions will have unique answers if the assumptions are added to questions as constraints. For example, the answer to the question "CMU's campuses" is "Pittrburgh" if we assume "campus" means "main campus".

1.2 Our Contributions

Neural-Symbolic Reasoning (Part I)

While many of reasoning process could be expressed as first-order logics and solved with an first-order logic reasoner, first-order logics operate on symbolic knowledge bases and thus are not directly applicable to answer natural language questions. Even though semantic parsing systems [186, 52, 139, 12, 178] have been proposed to parse natural language questions to symbolic queries, the parsing accuracy is limited and is often restricted to a predefined set of query templates and vocabulary. Even worse, symbolic knowledge bases are usually incomplete [101], resulting in the failure of logical inference when the exact knowledge needed is missing. In Part I, we simulate first-order logic with neural operations over symbolic KBs in symbolic and embedding spaces, as well as a virtual KB constructed from a text corpora.

- **Chapter 2**. Since first-order logic has exponential computation complexity [56], it is not possible for neural modules with polynomial complexity to perfectly simulate it. Instead, we implement a subset of logic operations that are commonly seen in real question answering tasks, including relational following, intersection and union, and constraints, and set difference. We introduce novel way of representing a symbolic knowledge base (KB) called a sparse-matrix reified KB (NQL). This representation enables neural KB inference modules that are fully differentiable, faithful to the original semantics of the KB, expressive enough to model multi-hop inferences, and scalable enough to use with realistically large KBs.

  [*This work has been published as "Scalable Neural Methods for Reasoning with a Symbolic Knowledge Base" at ICLR 2020.*]

- **Chapter 3**. Then, we propose EmQL, a query embedding technique, where KB entities and KB queries are represented jointly in an embedding space. Just as in performing reasoning tasks in symbolic space, our reasoning module can accurately execute the logic operations if required facts exist in the knowledge base. We propose a novel method for EmQL to enforce the faithfulness. More importantly, it can also infer plausible results if the required facts do not exist by searching for similar facts in embedding space. This approach improves the reasoning ability of models over traditional first-order logics in handling missing triples in symbolic knowledge base.

  [*This work has been published as "Faithful Embeddings for Knowledge Base Queries" at NeurIPS 2020.*]

- **Chapter 4**. While EmQL shows some generalizability to reason over incomplete KBs, KBs are still highly incomplete due to its predefined vocabulary of entities and relations. On the other hand, most knowledge in the real world is stored in unstructured format as free-form text. Text does not have fixed schema and predefined vocabulary of variables and predicates, and thus is flexible in knowledge representation but hard to query and reason over. In this chapter, we bridge the gap between structure and unstructured knowledge by proposing a strategy to construct a virtual KB [45, 138, 59]. Knowledge in the virtual KB is densely parametrized in embedding space, but is structured as variables and predicates that easily support reasoning operations just as the parameterized symbolic KBs discussed above. Since variables and predicates are represented in embedding space, they are not restricted to any predefined vocabulary.

[*Part of this work has been published as "Reasoning Over Virtual Knowledge Bases With Open Predicate Relations" at ICML 2021.*]

Integrating Reasoning Modules in Language Models (Part II)

Language Models (LM)[39, 92, 23, 5] are pretrained on large text corpora with self-supervision tasks, such as masked language modeling, to learn semantic and syntactic information. A natural way of applying pretrained Language Models for question answering is to convert questions into the pretraining format and finetune LMs to predict the answers. It is hoped that reasoning could be performed within the parameters of LMs, but experiments on a knowledge probing task, LAMA-TREx [117], show that pretrained Language Models have limited performance in QA tasks. The performance is even worse if multi-step reasoning is required. We propose to integrate reasoning modules into language models to improve their reasoning ability to answer complex natural language questions. The reasoning module includes an external memory that is constructed from symbolic or virtual KBs that will be queried multiple times to find relevant information. The language model, in turn, is responsible for question understanding as well as aggregating retrieved information to make final predictions. This integration provides LMs with better reasoning ability and, at the same time, improves the reasoning in understanding natural language questions.

- **Chapter 5.** We integrate EmQL, the embedding-based query language, into a language model to answer natural language questions. Specifically, we consider the parameterized KB in EmQL as an external memory and train a language model to output a query vector to perform the reasoning task. Considering that the parameterized KB is incomplete so answers may not exist in the KB, we mix the reasoning results into the language model to predict the final answers. We call this model Fact Injected Language Model (FILM). More interestingly, the external memory enables FILM to easily use new knowledge at test time to make predictions. This nice property of FILM could significantly reduce the computation cost in re-training LMs to capture up-to-date world knowledge.

  [*This work has been published as "Adaptable and Interpretable Neural Memory over Symbolic Knowledge" at NAACL 2021.*]

- Furthermore, we replace the parameterized symbolic KB in FILM with the virtual KB constructed by OPQL (called OPQL-LM), with minor changes to the original FILM model. OPQL-LM shows strong reasoning abilities in handling multi-hop questions using the virtual KB constructed from the Wikipedia corpus as its only source of knowledge.

  [*Part of this work has been published as "Reasoning Over Virtual Knowledge Bases With Open Predicate Relations" at ICML 2021.*]

Answering Questions with Conditions (Part III)

Many previous question answering tasks [125, 124, 75, 177, 68] assume that the questions have unique answers or lists of answers that are equally correct. We can model such questions with relational following, i.e. $X.\text{follow}(R)$, or other types of reasoning as motivated earlier in this chapter. However, questions asked in the real world miss important information and therefore are ambiguous. For example, the question "which is the first approved Covid vaccine" does not specify the geographical location so there are multiple probable answers, and a probable answer, "Pfizer-BioNTech", is only correct under the condition "in the United States". An important task in answering ambiguous questions is to determine information that has been missing from the questions and identify missing information conditions for answers.

In the logical inference domain, the reasoning task with incomplete information is sometimes called *abductive reasoning*, as opposed to *deductive reasoning* where all information needed to make a deterministic prediction is provided. We frame the task of abductive reasoning as answering questions with conditions. Specifically, for ambiguous questions, we developed models to find conditions associated with answers under which that the answers become unique and deterministic.

- **Chapter 7.** We start with a simple setting, where a list of candidate conditions is provided for each question and models only need to identify whether candidate conditions have already been satisfied, for example, "The answer is A, if two of the following conditions apply: X, Y, Z". This task is still challenging because conditions may interact with each under some logical relationship, e.g. "any of the following". To reflect this challenge, we propose a new dataset, ConditionalQA, to benchmark the task of identifying missing conditions. ConditionalQA contains questions with user scenarios which partially satisfy the required conditions and thus need *conditional answers*, i.e. the answers are only applicable when certain conditions apply. The dataset is very challenging to existing QA models. The performance of baselines models suggests there is huge room for improvement on the ConditionalQA task.

  [*This work has been published as "ConditionalQA: A Complex Reading Comprehension Dataset with Conditional Answers" at ACL 2022.*]

- **Chapter 8.** We propose a model, T-Reasoner, to tackle one of the challenges in the ConditionalQA dataset – finding missing conditions for answers constrained by multiple conditions which interact under some logical operations. We segment long documents into smaller pieces and input them to T-Reasoner. T-Reasoner consists of an entailment module to check the entailment state of the conditions and a reasoning module to perform the logical reasoning to identify unsatisfied conditions using a Transformer-based model. T-Reasoner is one of the first models focusing on the abductive reasoning tasks in Question Answering.

[*This work has been published as "Scenario-based Question Answering with Interacting Contextual Properties" at ICLR 2023.*]

- **Chapter 9.** However, the list of possible conditions of answers are rarely available for open-domain questions. For example, possible conditions for the question "which is the first approved Covid vaccine", which can be locations, dosage, or restrictions, etc., are not clear until contextual information about the question has been learned. We study the problem of answering ambiguous questions in open domain without a provided list of conditions as candidates. Models should instead predicting any text spans as conditions. To enable finding conditions in open-domain, we build a database of hundreds of millions of probably asked questions, generated from a large text corpus. Contextual information about questions is indirectly retrieved via the generated questions. Specifically, similar questions along with the passages where the questions are generated from are retrieved from the database. Conditions are then obtained through a question revision process using the retrieved passages. Conditions can either be returned directly or revised by mixing them with other retrieved information. This proposed method has been shown effective in the challenging question disambiguation task, achieving a new state-of-the-art.

  [*This work is in submission as "Answering Ambiguous Questions with a Database of Questions, Answers, and Revisions" to ICML 2023.*]

Part I

Neural-Symbolic Reasoning

Chapter 2

Neural Query Language (NQL)

2.1 Overview

Many questions—e.g., *what's the most recent movie that Quentin Tarantino directed?* or *which nearby restaurants have vegetarian entrees and take reservations?*—are best answered by *knowledge-based question-answering* (KBQA) methods, where an answer is found by accessing a KB. Within KBQA, a common approach is *neural semantic parsing*—i.e., using neural methods to translate a natural-language question into a structured query against the KB [186, 52, 139], which is subsequently executed with a symbolic KB query engine.

While this approach can be effective, it requires training data pairing natural-language questions with structured queries, which is difficult to obtain. Even worse, the parsing accuracy is sometimes low, especially if the questions are complex. Hence researchers have also considered *learning semantic parsers from denotations* [12, 178], where training data consists of pairs (q, A), where q is a natural-language question and A is the desired answer. Typically A is a set of KB entities—e.g., if q is the first sample question above, A would be[1] the singleton set containing *Once Upon a Time in Hollywood*.

Learning semantic parsers from denotations is difficult because the end-to-end process to be learned includes a non-differentiable operation—i.e., reasoning with the symbolic KB that contains the answers. To circumvent this difficulty, prior systems have used three different approaches. Some have used heuristic search to infer structured queries from denotations [114, 33]: this works in some cases but often an answer could be associated with many possible structured queries, introducing noise. Others have supplemented gradient approaches with reinforcement learning (e.g., [105]). Some systems have also "neuralized" KB reasoning, but before the work described here only over *small* KBs: this approach is natural when answers are naturally constrained to depend on a small set of facts (e.g., a single table [186, 57]), but more generally requires coupling a learner with some

[1] At the time of this writing.

(non-differentiable) mechanism to retrieve an appropriate small question-dependent subset of the KB as in [150, 144].

In this chapter, we introduce a simple yet novel scheme for incorporating reasoning on a large question-independent KB into a neural network, by representing a symbolic KB with an encoding called a *sparse-matrix reified KB*. A sparse-matrix reified KB is very compact, can be distributed across multiple GPUs if necessary, and is well-suited to modern GPU architecture. For KBs with many relations, a reified KB can be up to four orders of magnitude faster than alternative implementations (even alternatives based on sparse-matrix representations), and in our experiments we demonstrate scalability to a KB with over 13 million entities and nearly 44 million facts. This new architectural component leads to radically simpler architectures for neural semantic parsing from denotations—architectures based on a single end-to-end differentiable process, rather than cascades of retrieval and neural processes.

2.2 Method

2.2.1 Background

KBs, entities, and relations. A KB consists of *entities* and *relations*. We use x to denote an entity and r to denote a relation. Each entity has an integer index between 1 and N_E, where N_E is the number of entities in the KB, and we write x_i for the entity that has index i. A relation is a set of entity pairs, and represents a relationship between entities: for instance, if x_i represents "Quentin Tarantino" and x_j represents "Pulp Fiction" then (x_i, x_j) would be an member of the relation *director_of*. A relation r can thus be represented as a subset of $\{1, \ldots, N_E\} \times \{1, \ldots, N_E\}$. Finally a KB consists a set of relations and a set of entities.

Weighted sets as "k-hot" vectors. Our differentiable operations are based on *weighted sets*, where each element x of weighted set X is associated with a non-negative real number. It is convenient to define this weight to be zero for all $x \notin X$, while for $x \in X$, a weight less than 1 is a confidence that the set contains x, and weights more than 1 make X a multiset. If all elements of X have weight 1, we say X is a *hard set*. A weighted set X can be encoded as an *entity-set vector* $\mathbf{x} \in \mathbb{R}^{N_E}$, where the i-th component of $\mathbf{x}$ is the weight of x_i in X. If X is a hard entity set, then this will be a "k-hot" vector, for $k = |X|$. The set of indices of $\mathbf{x}$ with non-zero values is called the *support of $\boldsymbol{x}$*.

Sets of relations, and relations as matrices Often we would like to reason about sets of relations[2], so we also assume every relation r in a KB is associated with an entity and hence an integer index. We write r_k for the relation with index k, and we assume that relation entities are listed first in the index of entities, so the index k for r_k is between 1 and N_R, where N_R is the number of relations in the KB. We use R for a set of relations, e.g., $R = \{$*writer_of*, *director_of*$\}$ might be

[2]This is usually called *second-order* reasoning.

such a set, and use $\mathbf{r}$ for a vector encoding of a set. A relation r can be encoded as a *relation matrix* $\mathbf{M}_r \in \mathbb{R}^{N_E \times N_E}$, where the value for $\mathbf{M}_r[i,j]$ is (in general) the weight of the assertion $r(x_i, x_j)$ in the KB. In the experiments of this section, all KB relations are hard sets, so $\mathbf{M}_r[i,j] \in \{0,1\}$.

Sparse vs. dense matrices for relations. Scalably representing a large KB requires careful consideration of the implementation. One important issue is that for all but the smallest KBs, a relation matrix must be implemented using a *sparse matrix* data structure, as explicitly storing all N_E^2 values is impractical. For instance, consider a KB containing 10,000 movie entities and 100,000 person entities. A relationship like *writer_of* would have only a few tens of thousands of facts (since most movies have only one or two writers), but a dense matrix would have 1 billion values.

We thus model relations as *sparse matrices*. Let N_r be the number of entity pairs in the relation r: common sparse matrix data structures require space $O(N_r)$. One common sparse matrix data structure is a *sparse coordinate pair (COO)* encoding. A COO encoding consists of a $N_r \times 2$ matrix $\mathbf{Ind}_r$ containing pairs of entity indices, and a parallel vector $\mathbf{w}_r \in \mathbb{R}^{N_r}$ containing the weights of the corresponding entity pairs. In this encoding, if (i,j) is row k of $\mathbf{Ind}$, then $\mathbf{M}_r[i,j] = \mathbf{w}_r[k]$, and if (i,j) does not appear in $\mathbf{Ind}_r$, then $\mathbf{M}[i,j]$ is zero. With a COO encoding, each KB fact requires storing only two integers and one float.

Our implementations are based on Tensorflow [1], which offers limited support for sparse matrices. In particular, driven by the limitations of GPU architecture, Tensorflow only supports matrix multiplication between a sparse COO matrix and a dense matrix, but not between two sparse matrices, or between sparse higher-rank tensors and dense tensors.

Entity types. It is often possible to easily group entities into disjoint sets by some notion of "type": for example, in a movie domain, all entities might be either of the type "movie", "person", or "movie studio". It is straightforward to extend the formalism above to typed sets of entities, and doing this can lead to some useful optimizations. We use these optimizations below where appropriate: in particular, relation-set vectors $\mathbf{r}$ are of dimension N_R, not N_E, in the sections below.

Extension to soft KBs. In this work, we assume the non-zero weights in a relation matrix $\mathbf{M}_r$ are all equal to 1.0. This can be relaxed: if assertions in a KB are associated with confidences, then this confidence can be stored in $\mathbf{M}_r$. In this case, the reified KB must be extended to encode the weight for a triple: we find it convenient to redefine $\mathbf{M}_{rel}$ to hold that weight. In particular if the weight for the the ℓ-th triple $r_k(x_i, x_j)$ is w_ℓ, then we let

$$\mathbf{M}_{rel}[\ell, m] \equiv \begin{cases} w_\ell & \text{if } m = k_\ell \\ 0 & \text{else} \end{cases}$$

2.2.2 Reasoning in a KB

The relation-set following operation

Note that relations can also be viewed as labeled edges in a *knowledge graph*, the vertices of which are entities. Adopting this view, we define the *r-neighbors of an entity* x_i to be the set of entities x_j that are connected to x_i by an edge labeled r, i.e., *r-neighbors(x)* $\equiv \{x_j : (x_i, x_j) \in r\}$. Extending this to relation sets, we define

$$\textit{R-neighbors(X)} \equiv \{x_j : \exists r \in R, x_i \in X \text{ so that } (x_i, x_j) \in r\}$$

Computing the R-neighbors of an entity is a single-step reasoning operation: e.g., the answer to the question q =“*what movies were produced or directed by Quentin Tarantino*” is precisely the set R-neighbors(X) for $R = \{\textit{producer\_of, writer\_of}\}$ and $X = \{\textit{Quentin\_Tarantino}\}$. “Multi-hop” reasoning operations require nested R-neighborhoods, e.g. if $R' = \{\textit{actor\_of}\}$ then R'-neighbors(R-neighbors(X)) is the set of actors in movies produced or directed by Quentin Tarantino.

We would like to approximate the R-neighbors computation with differentiable operations that can be performed on the vectors encoding the sets X and R. Let $\mathbf{x}$ encode a weighted set of entities X, and let $\mathbf{r}$ encode a weighted set of relations. We first define $\mathbf{M}_R$ to be a weighted mixture of the relation matrices for all relations in R i.e.,

$$\mathbf{M}_R \equiv (\sum_{k=1}^{N_R} \mathbf{r}[k] \cdot \mathbf{M}_{r_k}) \tag{2.1}$$

We then define the *relation-set following operation for* $\boldsymbol{x}$ *and* $\boldsymbol{r}$ as:

$$\textit{follow}(\mathbf{x}, \mathbf{r}) \equiv \mathbf{x}\mathbf{M}_R = \mathbf{x}(\sum_{k=1}^{N_R} \mathbf{r}[k] \cdot \mathbf{M}_{r_k}) \tag{2.2}$$

As we will show below, this differentiable numerical relation-set following operation can be used as a neural component to perform certain types of logical reasoning. In particular, Eq 2.2 corresponds closely to the logical R-neighborhood operation, as shown by the claim below.

Claim 1 *The support of follow($\boldsymbol{x}, \boldsymbol{r}$) is exactly the set of R-neighbors(X).*

To better understand this claim, let $\mathbf{z} = \textit{follow}(\mathbf{x}, \mathbf{r})$. The claim states $\mathbf{z}$ can approximate the R neighborhood of any hard sets R, X by setting to zero the appropriate components of $\mathbf{x}$ and $\mathbf{r}$. It is also clear that $\mathbf{z}[j]$ decreases when one decreases the weights in $\mathbf{r}$ of the relations that link x_j to entities in X, and likewise, $\mathbf{z}[j]$ decreases if one decreases the weights of the entities in X that are linked to x_j via relations in R, so there is a smooth, differentiable path to reach this approximation.

More formally, consider first a matrix $\mathbf{M}_r$ encoding a single binary relation r, and consider the

Strategy	Definition	Batch?	Space complexity	# Operations		
				sp-dense matmul	dense $+$ or $\odot$	sparse $+$
naive mixing	Eq 2.1-2.2	no	$O(N_T + N_E + N_R)$	1	0	N_R
late mixing	Eq 2.3	yes	$O(N_T + bN_E + bN_R)$	N_R	N_R	0
reified KB	Eq 2.5	yes	$O(bN_T + bN_E)$	3	1	0

Table 2.1: Complexity of implementations of relation-set following, where N_T is the number of KB triples, N_E the number of entities, N_R the number of relations, and b is batch size.

vector $\mathbf{x}' = \mathbf{x}\mathbf{M}_r$. As weighted sets, X and r have non-negative entries, so clearly for all i,

$$\mathbf{x}'[j] \neq 0 \text{ iff } \exists j : \mathbf{M}_r[i,j] \neq 0 \wedge \mathbf{x}[i] \neq 0 \text{ iff } \exists x_i \in X \text{ so that } (x_i, x_j) \in r$$

and so if $\mathbf{r}$ is a one-hot vector for the set $\{r\}$, then the support of *follow*$(\mathbf{x}, \mathbf{r})$ is exactly the set r-neighbors(X). Finally note that the mixture $\mathbf{M}_R$ has the property that $\mathbf{M}_R[i(e_1), i(e_2)] > 0$ exactly when e_1 is related to e_2 by some relation $r \in R$.

2.2.3 Scalable relation-set following with a reified KB

2.2.3.1 Baseline Implementations

Suppose the KB contains N_R relations, N_E entities, and N_T triples. Typically $N_R < N_E < N_T \ll N_E^2$. As noted above, we implement each $\mathbf{M}_r$ as a sparse COO matrix, so collectively these matrices require space $O(N_T)$. Each triple appears in only one relation, so $\mathbf{M}_R$ in Eq 2.1 is also size $O(N_T)$. Since sparse-sparse matrix multiplication is not supported in Tensorflow we implement $\mathbf{x}\mathbf{M}_R$ using dense-sparse multiplication, so $\mathbf{x}$ must be a dense vector of size $O(N_E)$, as is the output of relation-set following. Thus the space complexity of *follow*$(\mathbf{x}, \mathbf{r})$ is $O(N_T + N_E + N_R)$, if implemented as suggested by Eq 2.2. We call this the *naive mixing* implementation, and its complexity is summarized in Table 2.1.

Because Tensorflow does not support general sparse tensor contractions, it is not always possible to extend sparse-matrix computations to minibatches. Thus we also consider a variant of naive mixing called *late mixing*, which mixes the *output* of many single-relation following steps, rather than mixing the KB itself:

$$\textit{follow}(\mathbf{x}, \mathbf{r}) = \sum_{k=1}^{N_R} (\mathbf{r}[k] \cdot \mathbf{x}\mathbf{M}_{r_k}) \tag{2.3}$$

Unlike naive mixing, late mixing can be extended easily to a minibatches:

$$\textit{follow}(\mathbf{X}, \mathbf{R}) = \sum_{k=1}^{N_R} (\mathbf{R}[:,k] \cdot \mathbf{X}\mathbf{M}_k) \tag{2.4}$$

Here, $\mathbf{X}$ be a minibatch of b examples $[\mathbf{x}_1; \ldots; \mathbf{x}_b]$. The k-th column of $\mathbf{R}$, is “broadcast” to element of the matrix $\mathbf{XM}_k$. This approach leads to N_R matrices $\mathbf{XM}_k$, each of size $O(bN_E)$. However, they need not all be stored at once, so the space complexity becomes $O(bN_E + bN_R + N_T)$. An additional cost of late mixing is that we must now sum up N_R dense matrices.

2.2.3.2 A Reified Knowledge Base

While semantic parses for natural questions often use small sets of relations (often singleton ones), in learning there is substantial uncertainty about what the members of these small sets should be. Furthermore, realistic wide-coverage KBs have many relations—typically hundreds or thousands. This leads to a situation where, at least during early phases of learning, it is necessary to evaluate the result of mixing very large sets of relations. When many relations are mixed, late mixing becomes quite expensive (as experiments below show).

An alternative is to represent each KB assertion $r_k(x_i, x_j)$ as a tuple (i, j, k) where i, j, k are the indices of x_i, x_j, and r_k. There are N_T such triples, so for $\ell = 1, \ldots, N_T$, let (i_ℓ, j_ℓ, k_ℓ) denote the ℓ-th triple. We define these sparse matrices:

$$\mathbf{M}_{subj}[\ell, m] \equiv \begin{cases} 1 & \text{if } m = i_\ell \\ 0 & \text{else} \end{cases} \qquad \mathbf{M}_{obj}[\ell, m] \equiv \begin{cases} 1 & \text{if } m = j_\ell \\ 0 & \text{else} \end{cases} \qquad \mathbf{M}_{rel}[\ell, m] \equiv \begin{cases} 1 & \text{if } m = k_\ell \\ 0 & \text{else} \end{cases}$$

Conceptually, $\mathbf{M}_{subj}$ maps the index ℓ of the ℓ-th triple to its subject entity; $\mathbf{M}_{obj}$ maps ℓ to the object entity; and $\mathbf{M}_{rel}$ maps ℓ to the relation. We can now implement the relation-set following as below, where $\odot$ is Hadamard product:

$$\mathit{follow}(\mathbf{x}, \mathbf{r}) = (\mathbf{x}\mathbf{M}_{subj}^T \odot \mathbf{r}\mathbf{M}_{rel}^T)\mathbf{M}_{obj} \tag{2.5}$$

Notice that $\mathbf{x}\mathbf{M}_{subj}^T$ are the triples with an entity in $\mathbf{x}$ as their subject, $\mathbf{r}\mathbf{M}_{rel}^T$ are the triples with a relation in $\mathbf{r}$, and the Hadamard product is the intersection of these. The final multiplication by $\mathbf{M}_{obj}$ finds the object entities of the triples in the intersection. These operations naturally extend to minibatches. The reified KB has size $O(N_T)$, the sets of triples that are intersected have size $O(bN_T)$, and the final result is size $O(bN_E)$, giving a final size of $O(bN_T + bN_E)$, with no dependence on N_R.

Table 2.1 summarizes the complexity of these three mathematically equivalent but computationally different implementions. The analysis suggests that the reified KB is preferable if there are many relations, which is the case for most realistic KBs[3].

2.2.3.3 Distributing a Large Reified KB

The reified KB representation is quite compact, using only six integers and three floats for each KB triple. However, since GPU memory is often limited, it is important to be able to *distribute* a

[3]The larger benchmark datasets used in this section have 200 and 616 relations respectively.

KB across multiple GPUs. Although to our knowledge prior implementations of distributed matrix operations (e.g., [140]) do not support sparse matrices, sparse-dense matrix multiplication can be distributed across multiple machines. *We thus implemented a distributed sparse-matrix implementation of reified KBs.* We distibuted the matrices that define a reified KB “horizontally”, so that different triple ids ℓ are stored on different GPUs. Details are provided in Section 2.2.3.4.

2.2.3.4 Distributed matrix multiplication

Matrix multiplication $\mathbf{xM}$ was distributed as follows: $\mathbf{x}$ can be split into a “horizontal stacking” of m submatrices, which we write as $[\mathbf{x}_1; \ldots; \mathbf{x}_m]$, and $\mathbf{M}$ can be similarly partitioned into m^2 submatrices. We then have the result that

$$\mathbf{xM} = [\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_m] \begin{bmatrix} \mathbf{M}_{1,1} & \mathbf{M}_{1,2} & \ldots & \mathbf{M}_{1,m} \\ \vdots & \vdots & & \vdots \\ \mathbf{M}_{m,1} & \mathbf{M}_{m,2} & \ldots & \mathbf{M}_{m,m} \end{bmatrix} = \left[(\sum_{i=1}^{m} \mathbf{x}_1 \mathbf{M}_{i,1}); \ldots; (\sum_{i=1}^{m} \mathbf{x}_m \mathbf{M}_{i,m}) \right]$$

This can be computed without storing either $\mathbf{X}$ or $\mathbf{M}$ on a single machine, and mathematically applies to both dense and sparse matrices. In our experiments we distributed the matrices that define a reified KB “horizontally”, so that different triple ids ℓ are stored on different GPUs.

Specifically, we shard the “triple index” dimension N_T of matrices $\mathbf{M}_{subj}$, $\mathbf{M}_{rel}$ and $\mathbf{M}_{obj}$ in Eq. 2.5 to perform a distributed relation-set following on the reified KB. Let $\mathbf{M}_{subj,i}$ be the i'th shard of matrix $\mathbf{M}_{subj}$, and thus $\mathbf{M}_{subj} = [\mathbf{M}^T_{subj,1}; \ldots; \mathbf{M}^T_{subj,m}]^T \in \mathbb{R}^{N_T \times N_E}$. $\mathbf{M}_{obj}$ and $\mathbf{M}_{rel}$ are represented in the similar way. A distributed relation-set following is computed as a combination of relation-set following results on all shards of the KB.

$$\begin{aligned} \mathit{follow}(\mathbf{x}, \mathbf{r}) &= (\mathbf{x}\mathbf{M}^T_{subj} \odot \mathbf{r}\mathbf{M}^T_{rel})\mathbf{M}_{obj} \\ &= \Big([\mathbf{x}\mathbf{M}^T_{subj,1}; \ldots; \mathbf{x}\mathbf{M}^T_{subj,m}] \odot [\mathbf{r}\mathbf{M}^T_{rel,1}; \ldots; \mathbf{r}\mathbf{M}^T_{rel,m}] \Big) \begin{bmatrix} \mathbf{M}_{obj,1} \\ \vdots \\ \mathbf{M}_{obj,m} \end{bmatrix} && (2.6) \\ &= \sum_{i=1}^{m} (\mathbf{x}\mathbf{M}^T_{subj,i} \odot \mathbf{r}\mathbf{M}^T_{rel,i})\mathbf{M}_{obj,i} && (2.7) \end{aligned}$$

This method can be easily extended to a mini-batch of examples $\mathbf{X}$.

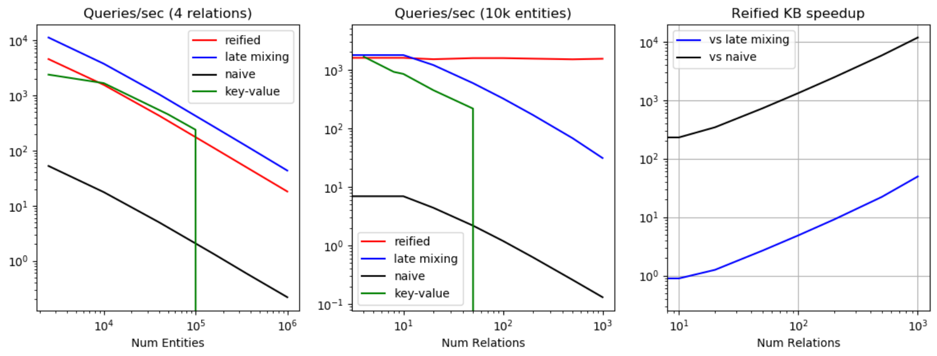


Figure 2.1: Left and middle: inference time in queries/sec on a synthetic KB as size and number of relations is varied. Queries/sec is given as zero when GPU memory of 12Gb is exceeded. Right: speedups of reified KBs over the baseline implementations.

2.3 Experiments

2.3.1 Scalability

2.3.1.1 Grid World

Like prior work [28, 36], we used a synthetic KB based on an n-by-n grid to study scalability of inference. Every grid cell is an entity, related to its immediate neighbors via relations *north*, *south*, *east*, and *west*. The entity vector $\mathbf{x}$ is a randomly-chosen singleton set, and the relation vector $\mathbf{r}$ weights relations roughly uniformly—more specifically, each relation has weight $1+\epsilon$ where ϵ is a drawn uniformly at random between 0 and 0.001.[4] We vary the number of relations by inventing m new relation names and assigning existing grid edges to each new relation. The KB for an n-by-n grid thus has $O(n^2)$ entities and $O(n^2)$ triples. We measured the time to compute the 2-hop inference $\mathit{follow}(\mathit{follow}(\mathbf{x}, \mathbf{r}), \mathbf{r})$ for minibatches of $b = 128$ one-hot vectors, and report it as queries per second (qps) on a single GPU (e.g., qps=1280 would mean a single minibatch requires 100ms).

We also compare to a key-value memory network [100]. The key is the concatenation of a relation and a subject entity, and the value is the object entity. We considered only the run-time for queries on an untrained randomly-initialized network (since run-time performance on a trained network would be the same); however, it should be noted that considerable time that might be needed to train the key-value memory to approximate the KB. (In fact, it is not obvious under what conditions a KB can be approximated well by the key-value memory.) We use an embedding size of 64 for entities and relations, where there is one memory entry for every triple in the KB.

[4]If the relation weights do not vary from trial to trial, some versions of Tensorflow will optimize computation by precomputing and caching the matrix $\mathbf{M}_R$ from Eq. 2.1, which speeds up the naive method considerably. Of course, this optimization is impossible when learning relation sets.

2.3.1.2 Experimental Results

The results are shown Figure 2.1 (left and middle), on a log-log scale because some differences are very large. With only four relations (the leftmost plot), late mixing is about 3x faster than the reified KB method, and about 250x faster than the naive approach. However, for more than around 20 relations, the reified KB is faster (middle plot). As shown in the rightmost plot, the reified KB is 50x faster than late mixing with 1000 relations, and nearly 12,000x faster than the naive approach.

With this embedding size, the speed of the key-value network is similar to the reified KB for only four relations, however it is about 7x slower for 50 relations and 10k entities. Additionally, the space needed to store a triple is much larger in a key-value network than the reified KB, so memory is exhausted when the KB exceeds 200,000 entities (with four relations), or when the KB exceeds 100 relations (with 10,000 entities.) The reified KB scales much better, and can handle 10x as many entities and 20x as many relations.

2.3.2 Models using reified KBs

The reified KB is closely related to key-value memory networks, so it can be viewed as a more efficient implementation of existing neural modules, optimized for reasoning with symbolic KBs. However, being able to include an entire KB into a model can lead to a *qualitative* difference in model complexity, since it is not necessary to build machinery to retrieve from the KB. To illustrate this, below we present simple models for several tasks, each using the reified KB in different ways, as appropriate to the task. We consider two families of tasks: learning semantic parsers from denotations over a large KB, and learning to complete a KB.

2.3.2.1 KBQA for Multi-hop Questions

MetaQA [184] consists of 1.2M questions, evenly distributed into one-hop, two-hop, and three-hop questions. (E.g, the question "*who acted in a movie directed by Quentin Tarantino?*" is a two-hop question.) The accompanying KB [100] contains 43k entities and 186k triples. Past work treated one-hop, two-hop and three-hop questions separately, and the questions are labeled with the entity ids for the "seed entities" that begin the reasoning chains (e.g., the question above would be tagged with the id of the entity for *Quentin Tarantino*).

Using a reified KB for reasoning means the neural model only needs to predict the relations used at each stage in the reasoning process. For each step of inference we thus compute relation sets $\mathbf{r}^t$ using a differentiable function of the question, and then chain them together with relation-set following steps. Letting $\mathbf{x}^0$ be the set of entities associated with q, the model we use is:

$$\text{for } t = 1, 2, 3: \quad \mathbf{r}^t = f^t(q); \quad \mathbf{x}^t = \mathit{follow}(\mathbf{x}^{t-1}, \mathbf{r}^t)$$

where $\mathit{follow}(\mathbf{x}^{t-1}, \mathbf{r}^t)$ is implemented with a reified KB as described in Eq. 2.5.

To predict an answer on a T-hop subtask, we compute the softmax of the appropriate set $\mathbf{x}^T$. We used cross entropy loss of this set against the desired answer, represented as a uniform distribution over entities in the target set. Each function $f^t(q)$ is a different linear projection of a common encoding for q, specifically a mean-pooling of the tokens in q encoded with a pre-trained 128-dimensional word2vec model [98]. The full KB was loaded into a single GPU in our experiments.

It is interesting to contrast this simple model with the one proposed by [184]. The "module for logic reasoning" they propose in previous sections is fairly complex, with a description that requires a figure, three equations, and a page of text; furthermore, training this model requires constructing an example-dependent subgraph for each training instance. In our model, the "logic reasoning" (and all interaction with the KB) has been encapsulated completely in the $\mathit{follow}(\mathbf{x}, \mathbf{r})$ operation—which, as we will demonstrate below, can be re-used for many other problems. Encapsulating all KB reasoning with a single scalable differentiable neural module greatly simplifies modeling: in particular, *the problem of learning a structured KB query has been reduced to learning a few differentiable functions of the question*, one for each reasoning "hop". The learned functions are also interpretable: they are mixtures of relation identifiers which correspond to soft weighted sets of relations, which in turn softly specify which KB relation should be used in each stage of the reasoning process. Finally, optimization is simple, as the loss on predicted denotations can be back-propagated to the relation-prediction functions.

A similar modeling strategy is used in all the other baseline models presented below.

2.3.2.2 KBQA on FreeBase

WebQuestionsSP [180] contains 4737 natural language questions, all of which are answerable using FreeBase [14], a large open-domain KB. Each question q is again labeled with the entities $\mathbf{x}$ that appear in it.

FreeBase contains two kinds of nodes: real-world *entities*, and *compound value types* (CVTs), which represent non-binary relationships or events (e.g., a movie release event, which includes a movie id, a date, and a place.) Real-world entity nodes can be related to each other or to a CVT node, but CVT nodes are never directly related to each other. In this dataset, all questions can be answered with 1- or 2-hop chains, and all 2-hop reasoning chains pass through a CVT entity; however, unlike MetaQA, the number of hops is not known. Our model thus derives from q three relation sets and then uniformly mixes both potential types of inferences:

$$\mathbf{r}_{\mathrm{E}\to\mathrm{E}} = f_{\mathrm{E}\to\mathrm{E}}(q); \quad \mathbf{r}_{\mathrm{E}\to\mathrm{CVT}} = f_{\mathrm{E}\to\mathrm{CVT}}(q); \quad \mathbf{r}_{\mathrm{CVT}\to\mathrm{E}} = f_{\mathrm{CVT}\to\mathrm{E}}(q)$$
$$\hat{\mathbf{a}} = \mathit{follow}(\mathit{follow}(\mathbf{x}, \mathbf{r}_{\mathrm{E}\to\mathrm{CVT}}), \mathbf{r}_{\mathrm{CVT}\to\mathrm{E}}) + \mathit{follow}(\mathbf{x}, \mathbf{r}_{\mathrm{E}\to\mathrm{E}})$$

We again apply a softmax to $\hat{\mathbf{a}}$ and use cross entropy loss, and $f_{\mathrm{E}\to\mathrm{E}}$, $f_{\mathrm{E}\to\mathrm{CVT}}$, and $f_{\mathrm{CVT}\to\mathrm{E}}$ are again linear projections of a word2vec encoding of q. We used a subset of Freebase with 43.7 million facts and 12.9 million entities, containing all facts in Freebase within 2-hops of entities mentioned in any question, excluding paths through some very common entities. We split the KB across three 12-Gb GPUs, and used a fourth GPU for the rest of the model.

This dataset is a good illustration of the scalability issues associated with prior approaches to including a KB in a model, such as key-value memory networks. A key-value network can be trained to implement something similar to relation-set following, if it stores all the KB triples in memory. If we assume 64-float embeddings for the 12.9M entities, *the full KB of 43.7M facts would be 67Gb in size*, which is impractical. Additionally performing a softmax over the 43.7M keys would be prohibitively expensive, as shown by the experiments of Figure 2.1. This is the reason why in standard practice with key-value memory networks for KBs, the memory is populated with a heuristically subset of the KB, rather than the full KB. We compare experimentally to this approach in Table 2.2.

2.3.2.3 Knowledge Base Completion

Following [176] we treat KB completion as an inference task, analogous to KBQA: a query q is a relation name and a head entity $\mathbf{x}$, and from this we predict a set of tail entities. We assume the answers are computed with the disjunction of multiple inference chains of varying length. Each inference chain has a maximum length of T and we build N distinct inference chains in total, using this model (where $\mathbf{x}_i^0 = \mathbf{x}$ for every chain i):

$$\text{for } i = 1, \ldots, N \text{ and } t = 1, \ldots, T\text{:} \quad \mathbf{r}_i^t = f_i^t(q); \quad \mathbf{x}_i^t = \mathit{follow}(\mathbf{x}_i^{t-1}, \mathbf{r}_i^t) + \mathbf{x}_i^{t-1}$$

The final output is a softmax of the mix of all the $\mathbf{x}_i^T$'s: i.e., we let $\hat{\mathbf{a}} = \mathit{softmax}(\sum_{i \in \{1 \ldots N\}} \mathbf{x}_i^T)$. The update $\mathbf{x}_i^{t+1} = \mathit{follow}(\mathbf{x}_i^t, \mathbf{r}_i^t) + \mathbf{x}_i^t$ gives the model access to outputs of all chains of length less than t. The encoding of q is based on a lookup table, and each f_i^t is a learned linear transformation of q's embedding.[5]

2.3.2.4 An Encoder-Decoder Architecture for Varying Inferential Structures

To explore performance on more complex reasoning tasks, we generated simple artificial natural-language sentences describing longer chains of relationships on a 10-by-10 grid. For this task we used an encoder-decoder model which emits chains of relation-set following operations. The question is encoded with the final hidden state of an LSTM, written here $\mathbf{h}^0$. We then generate a reasoning chain of length up to T using a decoder LSTM. At iteration t, the decoder emits a scalar probability of "stopping", p^t, and a distribution over relations to follow $\mathbf{r}^t$, and then, as we did for the KBQA tasks, sets $\mathbf{x}^t = \mathit{follow}(\mathbf{x}^{t-1}, \mathbf{r}^t)$. Finally the decoder updates its hidden state to $\mathbf{h}^t$ using an LSTM

[5] In the experiments we tune the hyperparameters $T \in \{1, \ldots, 6\}$ and $N \in \{1, 2, 3\}$ on a dev set.

cell that "reads" the "input" $\mathbf{r}^{t-1}$. For each step t, the model thus contains the steps

$$p^t = f_p(\mathbf{h}^{t-1}); \quad \mathbf{r}^t = f_r(\mathbf{h}^{t-1}); \quad \mathbf{x}^t = \mathit{follow}(\mathbf{x}^{t-1}, \mathbf{r}^t); \quad \mathbf{h}^t = \mathrm{LSTM}(\mathbf{h}^{t-1}, \mathbf{r}^{t-1})$$

The final predicted location is a mixture of all the $\mathbf{x}_t$'s weighted by the probability of stopping p_t at iteration t, i.e., $\hat{\mathbf{a}} = \mathit{softmax}(\sum_{t=1}^{T} \mathbf{x}^t \cdot p^t \prod_{t'<t}(1 - p^{t'}))$. The function f_r is a softmax over a linear projection, and f_p is a logistic function. In the experiments, we trained on 360,000 sentences requiring between 1 and T hops and tested on an additional 12,000 sentences.

2.3.2.5 Experimental Results

We next consider the performance of these models relative to strong baselines for each task. We emphasize our goal here is *not to challenge the current state of the art on any particular benchmark*, and clearly there are many ways the models of this work could be improved. (For instance, our question encodings are based on word2vec, rather than contextual encodings [40], and likewise relations are predicted with simple linear classifiers, rather than, say, attention queries over some semantically meaningful space, such as might be produced with language models or KB embedding approaches [15]). Rather, our contribution is to present a generally useful scheme for including symbolic KB reasoning into a model, and we have thus focused on describing simple, easily understood models that do this for several tasks. However, it is important to confirm experimentally that the reified KB models "work"—e.g., that they are amenable to use of standard optimizers, etc.

Performance (using Hits@1) of our models on the KBQA tasks is shown in Table 2.2. For the non-synthetic tasks we also compare to a Key-Value Memory Network (KV-Mem) baseline [100]. For the smaller MetaQA dataset, KV-Mem is initialized with all facts within 3 hops of the query entities, and for WebQuestionsSP it is initialized by a random-walk process seeded by the query entities (see [150, 184] for details). ReifKB consistently outperforms the baseline, dramatically so for longer reasoning chains. The synthetic grid task shows that there is very little degradation as chain length increases, with Hits@1 for 10 hops still 89.7%. It also illustrates the ability to predict entities in a KB, as well as relations.

We also compare these results to two much more complex architectures that perform end-to-end question answering in the same setting used here: VRN [184], GRAFT-Net [150], and PullNet [144]. All three systems build question-dependent subgraphs of the KB, and then use graph CNN-like methods [73] to "reason" with these graphs. Although not superior, ReifKB model is competitive with these approaches, especially on the most difficult 3-hop setting.

A small extension to this model is to mask the seed entities out of the answers. This model (given as ReifKB + mask) has better performance than GRAFT-Net on 2-hop and 3-hop questions.

For KB completion, we evaluated the model on the NELL-995 dataset [174] which is paired with a KB with 154k facts, 75k entities, and 200 relations. On the left of Table 2.3 we compare our model

	ReifKB (ours)	ReifKB + mask	KV-Mem (baseline)	VRN	GRAFT-Net	PullNet
WebQSP	52.7	—	46.7	—	**67.8**	**68.1**
MetaQA						
1-hop	96.2	—	95.8	**97.5**	97.0	97.0
2-hop	81.1	95.4	25.1	89.9	94.8	**99.9**
3-hop	72.3	79.7	10.1	62.5	77.2	**91.4**
Grid						
5-hop	98.4	—	—	—	—	–
10-hop	89.7	—	—	—	—	–

Table 2.2: Hits@1 on the KBQA datasets. Results for KV-Mem and VRN on MetaQA are from [184]; results for GRAFT-Net, PullNet and KV-Mem on WebQSP are from [150] and [144].

with three popular embedding approaches (results are from [31]). The reified KB model outperforms DistMult [175], is slightly worse than ConvE [38], and is comparable to ComplEx [156].

The competitive performance of the ReifKB model is perhaps surprising, since it has many fewer parameters than the baseline models—only one float and two integers per KB triple, plus a small number of parameters to define the f_i^t functions for each relation. The ability to use fewer parameters is directly related to the fact that our model *directly uses inference on the existing symbolic KB* in its model, rather than having to learn embeddings that approximate this inference. Or course, since the KB is incomplete, some learning is still required, but learning is quite different: the system learns logical inference chains in the incomplete KB that approximate a target relation. In this setting for KBC, the ability to perform logical inference "out of the box" appears to be very advantageous.

Another relative disadvantage of KB embedding methods is that KB embeddings are generally *transductive*—they only make predictions for entities seen in training. As a non-transductive baseline, we also compared to the MINERVA model, which uses reinforcement learning (RL) methods to learn how to traverse a KB to find a desired answer. Although RL methods are less suitable as "neural modules", MINERVA is arguably a plausible competitor to end-to-end learning with a reified KB.

MINERVA slightly outperforms our simple KB completion model on the NELL-995 task. However, unlike our model, MINERVA is trained to find a *single answer*, rather than trained to infer a

	NELL-995	
	H@1	H@10
ReifKB (Ours)	64.1	82.4
DistMult*	61.0	79.5
ComplEx*	61.2	82.7
ConvE*	**67.2**	**86.4**

	ReifKB (Ours)	MINERVA
NELL-995	64.1	**66.3**
Grid with seed entity		
10-hop NSEW	98.9	**99.3**
10-hop NSEW-VH	**73.6**	34.4
MetaQA 3-hop	**72.3**	41.7

Table 2.3: Left: Hits@1 and Hits@10 for KB completion on NELL 995. Starred KB completion methods are transductive, and do not generalize to entities not seen in training. Right: Comparison to MINERVA on several tasks for Hits@1.

	NELL-995	MetaQA-3hop	WebQuestionsSP
# Facts	154,213	196,453	43,724,175
# Entities	75,492	43,230	12,942,798
# Relations	200	9	616
Time (seconds)	44.3	72.6	1820

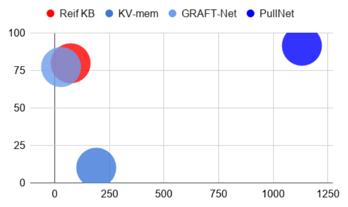


Table 2.4: Left, time to run 10K examples for KBs of different size. Right, time for 10k examples vs Hits@1 performance for ReifKB compared to three baselines on MetaQA-3hop questions.

set of answers. To explore this difference, we compared to MINERVA on the grid task under two conditions: (1) the KB relations are the grid directions north, south, east and west, so the output of the target chain is always a *single* grid location, and (2) the KB relations also include a "vertical move" (north or south) and a "horizontal move" (east or west), so the result of the target chain can be a *set* of locations. As expected MINERVA's performance drops dramatically in the second case, from 99.3% Hits@1 to 34.4 %, while our model's performance is more robust. MetaQA answers can also be sets, so we also modified MetaQA so that MINERVA could be used (by making the non-entity part of the sentence the "relation" input and the seed entity the "start node" input) and noted a similarly poor performance for MINERVA. These results are shown on the right of Table 2.3.

In Tables 2.4 we compare the training time of our model with minibatch size of 10 on NELL-995, MetaQA, and WebQuestionsSP. With over 40 million facts and nearly 13 million entities from Freebase, it takes less than 10 minutes to run one epoch over WebQuestionsSP (with 3097 training examples) on four P100 GPUs. In the accompanying plot, we also summarize the tradeoffs between accuracy and training time for our model and three baselines on the MetaQA 3-hop task. (Here ideal performance is toward the upper left of the plot). The state-of-the-art PullNet [144] system, which uses a learned method to incrementally retrieve from the KB, is about 15 times slower than the reified KB system. GRAFT-Net is only slightly less accurate, but also only slightly faster: recall that GRAFT-Net uses a heuristically selected subset (of up to 500 triples) from the KB for each query, while our system uses the full KB. Here the full KB is about 400 times as large as the question-specific subset used by GRAFT-Net. A key-value memory baseline including the full KB is nearly three times as slow as our system, while also performing quite poorly.

2.4 Related Work

TensorLog [28], a probabilistic logic which also can be compiled to Tensorflow, and hence is another differentiable approach to neuralizing a KB. TensorLog is also based on sparse matrices, but does not support relation sets, making it unnatural to express the models shown in this chapter, and does not use the more efficient reified KB representation. The differentiable theorem prover (DTP) is another differentiable logic [133], but DPT appears to be much less scalable: it has not been applied to KBs larger than a few thousand triples. The Neural ILP system [176] uses approaches related to late mixing together with an LSTM controller to perform KB completion and some simple QA tasks, but it is a monolithic architecture focused on rule-learning, while in contrast we propose a re-usable neural component, which can be used in as a component in many different architectures, and a scalable implementation of this. It has also been reported that neural ILP does not scale to the size of the NELL995 task [31].

The goals of this chapter are related to KB embedding methods, but distinct. In KB embedding, models are generally fully differentiable, but it is not considered necessary (or even desirable) to accurately match the behavior of inference in the original KB. Being able to construct a learned *approximation* of a symbolic KB is undeniably useful in some contexts, but embedded KBs also have many disadvantages. In particular, they are much larger than a reified KB, with many more learned parameters—typically a long dense vector for every KB entity. Embedded models are typically evaluated by their ability to score a single triple accurately, and many models are not capable of executing multi-step KB inferences efficiently; further, models that do allow multi-step inference are known to produce cascaded errors on long reasoning chains [60, 61]. In contrast we focus on accurate models of reasoning in a symbolic KB, which requires consideration of novel scalability issues associated with sparse matrice representations.

Mathematically, our definition of relation-set following is much like the bilinear model for path following from [60]; however, we generalize this to path queries that include weighted sets of relations, allowing the relations in paths to be learned. Similar differences apply to the work of [61], which extends the work of [60] to include intersection operations. The vector representation used here for weighted sets in a reified KB makes intersection trivial to implement, as intersection corresponds to Hadamard product. Conveniently set union also corresponds to vector sum, and the complement of X is $1 - \mathbf{x}$, which is perhaps why only a single additional neural operation is needed to support the KB reasoning tasks needed for the five benchmark tasks considered here.

Neural architectures like memory networks [171], or other architectures that use attention over some data structure approximating assertions [6, 57] can be used to build soft versions of relation-set following: however, they also do not scale well to large KBs, so they are typically used either with a non-differentiable *ad hoc* retrieval mechanism, or else in cases where a small amount of information is relevant to a question [170, 186]. Similarly graph CNNs [73] also can be used for reasoning, and often do use sparse matrix multiplication, but again existing implementations have not been scaled to tens of millions of triples/edges or millions of entities/graph nodes. Additionally, while graph CNNs have been used for reasoning tasks, the formal connection between them and logical reasoning remains unclear, whereas there is a precise connection between relation-set following and inference.

Reinforcement learning (RL) methods have been used to learn mappings from natural-language questions to non-differentiable logical representations [85, 87] and have also been applied to KB completion tasks [31, 174]. Above we compared experimentally to MINERVA, one such method;

however, the gradient-based approaches enabled by our methods are generally preferred as being easier to implement and tune on new problems, and easier to combine in a modular way with other architectural elements.

2.5 Discussion

We introduced here a novel way of representing a symbolic knowledge base (KB) called a sparse-matrix reified KB. This representation enables neural modules that are fully differentiable, faithful to the original semantics of the KB, expressive enough to model multi-hop inferences and set intersection and union, and scalable enough to use with realistically large KBs. In a reified KB, all KB relations are represented with three sparse matrices, which can be distributed across multiple GPUs, and symbolic reasoning on realistic KBs with many relations is much faster than with naive implementations—more than four orders of magnitude faster on synthetic-data experiments compared to naive sparse-matrix implementations.

This new architectural component leads to radically simpler architectures for neural semantic parsing from denotations and KB completion—in particular, they make it possible to learn neural KBQA models in a completely end-to-end way, mapping from text to KB entity sets, for KBs with tens of millions of triples and entities and hundreds of relations.

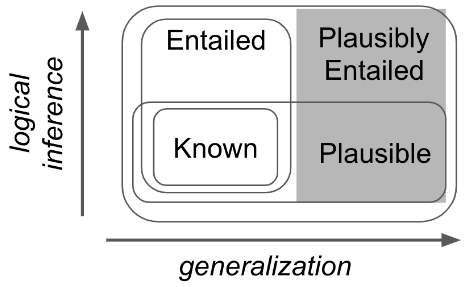
The proposed method, however, is restricted to triples provided in KBs. It is not possible to generalize to triples that are not provided but still plausible. In the next chapter, we will discuss another query language, EmQL, which performs reasoning in embedding space and shows generalizability to plausible triples beyond provided ones.

Chapter 3

Embedding Query Language (EmQL)

3.1 Overview

NQL performs logical inference in the symbolic space by compiling KBs into sparse matrices and run the inference tasks with vector and matrix operations that enables to find deductively entailed answers to queries. KBs, however, are in practice incomplete and over-specified, failing to answer queries for real world questions. *Query embedding* (QE) methods extend logical queries to incomplete KBs by representing KB entities and KB queries in a joint embedding space, supporting relaxation and generalization in KB inference [60, 61, 165, 128]. Figure 3.1 summarizes the relationship between traditional KB embedding (KBE), query embedding (QE), and logical inference. Traditional logical inference enables a system to find deductively *entailed* answers to queries; KBE approaches allow a system to *generalize* from explicitly-stored KB tuples to similar tuples; and QE methods combine both of these ideas, providing a soft form of logical entailment that generalizes.



KB embedding (KBE) methods generalize from *known* KG facts to *plausible* ones, and logical inference computes answers to compositional queries that are *entailed* by known facts. Query embedding (QE) combines both of these tools for extending a set of known facts, by finding answers to a query that are *plausibly entailed* by known facts.

Figure 3.1: Overview of differences between KBE and QE. Shaded area indicates the kinds of test cases used in prior studies of QE.

We say that a QE system is *logically faithful* if it behaves similarly to a traditional logical inference system with respect to entailed answers. However, experiments illustrating that QE systems are often *not* faithful: in particular, they performs quite poorly in finding logically-entailed answers. We conjecture this is because models that generalize well do not have the capacity to model all the information in a large KB accurately, unless embeddings are impractically large. We thus propose two novel methods for improving faithfulness while preserving the ability to generalize. First, we implement some logical operations using neural retrieval over a KB of embedded triples, rather than with geometric operations in embedding space, thus adding a non-parametric component to QE. Second, we employ a randomized data structure called a count-min sketch to propagate scores of logically-entailed answers. We show that this combination leads to a QE method, called EmQL (Embedding Query Language) which is differentiable, compact, scalable, and (with high probability) faithful. Furthermore, strategically removing the sketch in parts of the QE system allows it to generalize very effectively.

3.2 Method

The query language EmQL operates on weighted sets of entities. Let U be the set of all entities in a KB. A weighted set $X \subseteq U$ is *canonically encoded* as a k-hot vector $\mathbf{v}_X \in \mathbb{R}^N$, where $N = |U|$ and $\mathbf{v}_X[i]$ holds the non-negative real *weight* of element i in X. However the k-hot encoding is very inefficient if N is large, which we address later. EmQL relies on a learned embedding $\mathbf{e}_i \in \mathbb{R}^d$ for each entity i, which together form the matrix $\mathbf{E} \in \mathbb{R}^{d \times N}$ of entity embeddings. A weighted set X will be represented by a pair consisting of (1) a dense vector derived from its entity embeddings $\{\mathbf{e}_i\}$, $i \in X$, plus (2) an efficient sparse representation of the weights $\mathbf{v}_X[i]$.

In addition to (weighted) set intersection, union, and difference, which are common to many KBE models, EmQL implements two operators for relational reasoning: *relation following* and *relational filtering*. EmQL also supports a limited form of set difference. In this section, we will start by discussing how to encode and decode sets with EmQL representations, and then discuss the operators in EmQL for relational reasoning.

3.2.1 Background on Count-min Sketches

3.2.1.1 Definitions

Count-min sketches [29] are a widely used randomized data structure. We include this discussion for completeness, and our analysis largely follows [30].

A count-min sketch, as used here, is an approximation of a vector representation of a weighted set. Assume a universe U which is a set of integer "object ids" from $\{1, \ldots, N\}$. A set $A \subseteq U$ can be encoded as a vector $\mathbf{v}_A \in \mathbb{R}^n$ such that $\mathbf{v}_A[i] = 0$ if $i \notin S$, and otherwise $\mathbf{v}_A[i]$ is a real-numbered weight for entity i in set S. The purpose of the count-min sketch is to approximate $\mathbf{v}_A$ with limited storage.

Let h be a hash function mapping $\{1, \ldots, N\}$ to a smaller range of integers $\{1, \ldots, N_W\}$, where $N_W \ll N$. The *primitive sketch of $\boldsymbol{v}_A$ under h*, written $\mathbf{s}_h(\mathbf{v}_A)$, is a vector such that

$$\mathbf{s}_h(\mathbf{v}_A)[j] = \sum_{i:h(i)=j} \mathbf{v}_A[i]$$

Algorithmically, this vector could be formed by starting with an all-zero's vector of length N_W, then looping over every pair (i, w_i) where $w_i = \mathbf{v}_A[i]$ and incrementing each $\mathbf{s}_h[j]$ by w_i. A primitive sketch $\mathbf{s}_h$ contains some information about $\mathbf{v}_A$: to look up the value $\mathbf{v}_A[i]$, we could look up $\mathbf{s}_h[h(i)]$, and this will have the correct value if no other set element i' hashed to the same location. We can improve this by using multiple hash functions.

Specifically, let $H = \{h_1, \ldots, h_{N_D}\}$ be a list of N_D hash functions mapping $\{1, \ldots, N\}$ to the smaller range of integers $\{1, \ldots, N_W\}$. The *count-min sketch $\boldsymbol{S}_H(\boldsymbol{v}_A)$ for a $\boldsymbol{v}_A$ under H* is a matrix such that each row j is the primitive sketch of $\mathbf{v}_A$ under h_j. This sketch is an $N_W \times N_D$ matrix: N_W is called the sketch width and N_D is called the sketch depth.

Let $\mathbf{S}$ be the count-min sketch for A. To "look up" (approximately recover) the value of $\mathbf{v}_A[i]$, we compute this quantity

$$\mathit{CM}(i, \mathbf{S}) \equiv \min_{j=1}^{N_D} \mathbf{S}[j, h_j(i)]$$

In other words, we look up the hashed value associated with i in each of the N_D primitive sketches, and take the minimum value.

3.2.1.2 Linearity and implementation nodes

Count-min sketches also have a useful "linearity" property, inherited from primitive sketches. It is easy to show that for any two sets A and B represented by vectors $\mathbf{v}_A$ and $\mathbf{v}_B$

$$\begin{aligned}
\mathbf{S}_H(\mathbf{v}_A + \mathbf{v}_B) &= \mathbf{S}_H(\mathbf{v}_A) + \mathbf{S}_H(\mathbf{v}_B) \\
\mathbf{S}_H(\mathbf{v}_A \odot \mathbf{v}_B) &= \mathbf{S}_H(\mathbf{v}_A) \odot \mathbf{S}_H(\mathbf{v}_B)
\end{aligned}$$

Here, as elsewhere in this section, $\odot$ is Hadamard product.

In general, although it is mathematically convenient to define the behavior of sketches in reference to k-hot vectors, it is *not necessary to construct a vector $\boldsymbol{v}_A$ to construct a sketch*: all that is needed is the non-zero weights of the elements of A. Alternatively, if one precomputes and stores the sketch for each singleton set, it is possible to create sketches for an arbitrary set by gathering and sum-pooling the sketches for each element.

3.2.1.3 Probabilistic bounds on accuracy

We assume the hash functions are random mappings from $\{1,\ldots,N\}$ to $\{1,\ldots,N_W\}$. More precisely, we assume that for all $i \in \{1,\ldots,N\}$, and all $j \in \{1,\ldots,N_W\}$, $\Pr(h_i(x)=a) = \frac{1}{N_W}$. We will also assume that the N_D hash functions are are all drawn *independently* at random. More precisely, for all $i \neq i'$, $i,i' \in \{1,\ldots,N\}$, all $j,j' \in \{1,\ldots,N_D\}$ and all $k,k' \in \{1,\ldots,N_W\}$, $\Pr(h_j(i) = k \wedge h_{j'}(i') = k') = \frac{1}{N_W^2}$.

Under this assumption, the probability of errors can be easily bounded. Suppose the sketch width is at least twice the cardinality of A, i.e., $|A| < m$ and $N_W > 2m$. Then one can show for all primitive sketches j:

$$\Pr(\mathbf{S}[j, h_j(i)] \neq \mathbf{v}_A[i]) \leq \frac{1}{2}$$

From this one can show that the probability of any error in a count-min sketch decreases exponentially in sketch depth. (This result is a slight variant of one in [30].)

Theorem 1 *Assuming hash functions are random and independent as defined above, then if $\boldsymbol{S}$ is a count-min sketch for A of depth N_D, and $N_W > 2|A|$, then*

$$\Pr(CM(\boldsymbol{S}, i) \neq \boldsymbol{v}_A[i]) \;\leq\; \frac{1}{2^{N_D}}$$

This bound applies to a single CM operation. However, by using a union bound it is easy to assess the probability of making an error in any of a series of CM operations. In particular, we consider the case that there is some set of candidates C including all entities in A, i.e., $A \subseteq C \subseteq U$, and consider recovering the set A by performing a CM lookup for every $i' \in C$. Specifically, we say that A *can be recovered from* $\boldsymbol{S}$ *using* C if $A \subseteq C$ and

$$\forall i' \in C, CM(i', \mathbf{S}) = \mathbf{v}_A[i']$$

Note that this implies the sketch must correctly score every $i' \in C - A$ as zero. Applying the union bound to Theorem 1 leads to this result.

Theorem 2 *Let $\boldsymbol{S}$ be a count-min sketch for A of depth N_D and with $N_W > 2|A|$, and let $C \supseteq A$. If $N_D > \log_2 \frac{|C|}{\delta}$ then with probability at least 1-δ, A can be recovered from $\boldsymbol{S}$ using C.*

Many other bounds are known for count-min sketches: perhaps the best-known result is that for $N_W > \frac{2}{\epsilon}$ and $N_D > \log \frac{1}{\delta}$, the probability that $CM(i, \mathbf{S}) > \mathbf{v}_A[i] + \epsilon$ is no more than δ [29]. Because there are many reasonable formal bounds that might or might not apply in an experimental setting, typically the sketch shape is treated as a hyperparameter to be optimized in experimental settings.

3.2.2 Representing Sets

We would like to represent entity sets with a scheme that supports generalization, but also allows for precisely encoding weights of sets that are defined by compositional logic-like operations. Our representation will assume that sets are of limited cardinality, and contain “similar” entities (as defined below).

We represent a set X with the pair $(\mathbf{a}_X, \mathbf{b}_X)$, $\mathbf{a}_X = \sum_i \mathbf{v}_X[i]\ \mathbf{e}_i$, $\mathbf{b}_X = \mathbf{S}_H(\mathbf{v}_X)$ where $\mathbf{a}_X$ is the weighted centroid of elements of X that identifies the general region containing elements of X, and $\mathbf{b}_X$ is an optional count-min sketch [29], which encodes additional information on the weights of elements of X.

To reconstruct a set from this encoding, we first take the k elements with highest dot product $\mathbf{a}_X^T \mathbf{e}_i$, where k is a fixed hyperparameter. This is done efficiently with a maximum inner product search [108] (MIPS), which we write $\text{TOP}_k(\mathbf{a}_X, \mathbf{E})$.[1] These top k elements are then filtered by the count-min sketch, resulting in a sparse (no more than k non-zeros) *decoding* of the set representation

$$\hat{\mathbf{v}}_X[i] = \begin{cases} CM(i, \mathbf{b}_X) \cdot \text{softmax}(\mathbf{a}_X^T\ \mathbf{e}_i) \text{ if } i \in \text{TOP}_k(\mathbf{a}_X, \mathbf{E}) \\ 0 \text{ else} \end{cases}$$

The two pairs of the centroid-sketch representation are complementary. The region around a centroid will usually contain entities with many similar properties, for example “US mid-size cities,” or “Ph.D. students in NLP”: conceptually, it can be viewed as defining a *soft type* for the entities in X. However, simple geometric representations like centroids are not expressive enough to encode arbitrary sets X, like “Ph.D. students presenting papers in session z”. Count-min sketches do allow arbitrary weights to be stored, but may return incorrect values (with low probability) when queries. However, in this scheme the sketch is only queried for k candidates close to the centroid, so it is possible to obtain very low error probabilities with small sketches (discussed later).

The centroid-sketch representation does assume that all elements of the same set are similar in the sense that they all have the same “soft type”—i.e., are all in a sphere around a specific centroid. It also assumes that sets are of size no more than k. (Note the radius of the sphere is not learned—instead k is simply a hyperparameter.)

3.2.3 Faithfulness

Below we define compositional operations (like union, intersection, etc) on centroid-sketch set representations. A representation produced this way is associated with a particular logical definition of a set X (e.g., $X = Z_1 \cup Z_2$), and we say that the representation is *faithful* to that definition to the

[1]While $\mathbf{a}_X$ could be based on other geometric representations for sets, we use MIPS queries because obtaining candidates this way can be very efficient [108].

extent that it yields the appropriate elements when decoded (which can be measured experimentally).

Experimentally sketches improve the faithfulness of EmQL. However, *the sketch part of a set representation is optional*—specifically it can be replaced with a vacuous sketch that returns a weight of 1.0 whenever it is queried.[2] Removing sketches is useful when one is focusing on generalization.

3.2.4 Intersection and union

Set interesection and union of sets A and B will be denoted as $(\mathbf{a}_{A\cap B}, \mathbf{b}_{A\cap B})$ and $(\mathbf{a}_{A\cup B}, \mathbf{b}_{A\cup B})$, respectively. Both operations assume that the soft types of A and B are similar, so we can define the new centroids as

$$\mathbf{a}_{A\cap B} = \mathbf{a}_{A\cup B} = \frac{1}{2}(\mathbf{a}_A + \mathbf{a}_B)$$

To combine the sketches, we exploit the property that if $\mathbf{b}_A$ and $\mathbf{b}_B$ are sketches for A and B respectively, then a sketch for $A \cup B$ is $\mathbf{b}_A + \mathbf{b}_B$, and the sketch for $A \cap B$ is $\mathbf{b}_A \odot \mathbf{b}_B$ (where $\odot$ is Hadamard product). Hence

$$\mathbf{b}_{A\cap B} = \mathbf{b}_A \odot \mathbf{b}_B \qquad \mathbf{b}_{A\cup B} = \mathbf{b}_A + \mathbf{b}_B$$

3.2.5 Relational following

As noted above, relation following takes a set of entities X and a set of relations R and computes the set of entities related to something in X via some relation in R:

$$X.\mathit{follow}(R) \equiv \{y \mid \exists r \in R, x \in X : r(x,y)\}$$

where "$r(x,y)$" indicates that this triple is in the KB. For example, to look up the headquarters of the Apple company one might compute $Y = X.\mathit{follow}(R)$ where X and R are singleton sets containing "*Apple_Inc*" and "*headquarters_of*" respectively, and result set $Y = \{\mathit{Cupertino}\}$.

Relation following is implemented using an embedding matrix $\mathbf{K}$ for KB triples that parallels the element embedding matrix $\mathbf{E}$: for every triple $t = r(x,y)$ in the KB, $\mathbf{K}$ contains a row $\mathbf{r}_t = [\mathbf{e}_r; \mathbf{e}_x; \mathbf{e}_y]$ concatenating the embeddings for r, x, and y. To compute $Y = X.\mathit{follow}(R)$ first we create a query $\mathbf{q}_{R,X} = [\lambda \cdot \mathbf{a}_R; \mathbf{a}_X; \mathbf{0}]$ by concatenating the centroids for R and X and padding it to the same dimension as the triple embeddings (and λ is a hyper-parameter scaling the weight of the relation). Next using the query $\mathbf{q}_{R,X}$, we perform a MIPS search against all triples in KB $\mathbf{K}$ to get the top k triples matching this query, and these triples are re-scored with the sketches of X and R. Let $\mathbf{r}_t = \left[\mathbf{e}_{r_i}; \mathbf{e}_{x_j}; \mathbf{e}_{y_\ell}\right]$ be the representation of retrieved triple $t = r_i(x_j, y_\ell)$. Its score is

$$s(\mathbf{r}_t) = \mathrm{CM}(i, \mathbf{b}_R) \cdot \mathrm{CM}(j, \mathbf{b}_X) \cdot \mathrm{softmax}(\mathbf{q}_{R,X}^T\ \mathbf{r}_t)$$

[2]For count-min sketches, if $\mathbf{b}_{\mathbb{I}}$ is an all-ones matrix of the correct size, then $\forall i\ \mathit{CM}(i, \mathbf{b}_{\mathbb{I}}) = 1$.

We can then project out the objects from the top k triples as a sparse k-hot vector:

$$\hat{\mathbf{v}}_Y(\ell) = \sum_{\mathbf{r}_t \in \mathrm{TOP}_k(\mathbf{q}_{R,X}, \mathbf{K}), t = \_(\_, y_\ell)} s(\mathbf{r}_t)$$

Finally $\hat{\mathbf{v}}_Y$ is converted to a set representation $(\mathbf{a}_Y, \mathbf{b}_Y)$, which represents the output of the operation, $Y = X.\mathit{follow}(R)$. The triple store used for implementing *follow* is thus a kind of key-value memory network [100], augmented with a sparse-dense filter in the form of a count-min sketch.

3.2.6 Relational filtering

Relational filtering, similar to an existential restriction in description logics, removes from X those entities that are not related to something in set Y via some relation in R:

$$X.\mathit{filter}(R, Y) \equiv \{x \in X | \exists r \in R, y \in Y : r(x, y)\}$$

For example, $X.\mathit{filter}(R, Y)$ would filter out the companies in X whose headquarters are not in Cupertino, if R and Y are as in the previous example. Relational filtering is implemented similarly to *follow*. For $X.\mathit{filter}(R, Y)$, the query must also be aware of the objects of the triples, since they should be in the set Y. The query vector is thus $\mathbf{q}_{R,X,Y} = [\lambda \cdot \mathbf{a}_R; \mathbf{a}_X; \mathbf{a}_Y]$. Again, we perform a retrieval using query $\mathbf{q}_{R,X,Y}$, but we filter with subject, relation, and object sketches $\mathbf{b}_R$, $\mathbf{b}_X$, $\mathbf{b}_Y$, so the score of an encoded triple $\mathbf{r}_t$ is

$$s(\mathbf{r}_t) = \mathrm{CM}(i, \mathbf{b}_R) \cdot \mathrm{CM}(j, \mathbf{b}_X) \cdot \mathrm{CM}(\ell, \mathbf{b}_Y) \cdot \mathrm{softmax}(\mathbf{q}_{R,X,Y}^T \ \mathbf{r}_t)$$

The same aggregation strategy is used as for the *follow* operation, except that scores are aggregated over the subject entities instead of objects.

3.2.7 Set difference

Another operation we use is set difference: e.g. "movie directors but not writers" requires one to compute a set difference $A_{\text{directors}} - B_{\text{writers}}$. In computing a set difference, the soft-type of the output $A - B$ is the same as that of A, and we exclude the necessary elements from the count-min sketch to produce $(\mathbf{a}_{A-B}, \mathbf{b}_{A-B})$, where

$$\begin{aligned} \mathbf{a}_{A-B} &= \mathbf{a}_A \\ \mathbf{b}_{A-B} &= \mathbf{b}_A \odot (\mathbf{b} \neq 0) \end{aligned}$$

This is exact when B is unweighted (the case we consider here), but only approximates set difference for general weighted sets.

3.2.8 Size and density of sketches

Although the centroid-based geometric constraints are not especially expressive, we note that EmQL's sparse-dense representation can still express sets accurately, as long as the k-nearest neighbor retrieval has good recall. Concretely, consider a set A with $|A| = m$ and sparse-dense representation $(\mathbf{a}_A, \mathbf{b}_A)$. Suppose that $k = cm$ ensures that all m elements of A are retrieved as k-nearest neighbors of $\mathbf{a}_A$; in other words, retrieval precision may be as low as $1/c$. By Theorem 2 in Appendix 3.2.1, a sketch of size $2m \log_2 \frac{cm}{\delta}$ will recover *all* the weights in A with probability at least $1 - \delta$.

In our experiments we assume sets are of size $m < 100$, and that $c = 10$. Using 32 numbers per potential set member leads to $\delta \approx \frac{1}{50}$ and a sketch size of about 4k. Put another way, sets of 100 elements require about as much storage as the BERT [41] contextual encoding of 4 tokens; alternatively the sketch for 100 elements requires about 1/4 the storage of 100 embeddings with $d = 128$.[3]

It is also easy to see that for a set of size m, close to half of the numbers in the sketch will have non-zero values. Thus only a moderate savings in space is obtained by using a sparse-matrix data structure: it is quite practical to encode sketches with GPU-friendly dense-tensor data structures.

3.2.9 Loss function

This representation requires entities that will be grouped into sets to be close in embedding space, so entity embeddings must be trained to have this property—ideally, for all sets that arise in the course of evaluating queries. In the training process we use to encourage this property, an example is a query (such as "$\{$*Apple_Inc*$\}$.*follow*($\{$*headquarters_of*$\}$ $\cup$ $\{$*Sunnyvale*$\}$") and a target output set Y. Evaluation of the query produces an approximation $\hat{Y}$, encoded as $(\hat{\mathbf{a}}_Y, \hat{\mathbf{b}}_Y)$, and the goal of training is make $\hat{Y}$ approximate Y.

Let $\mathbf{v}_Y$ be the canonical k-hot encoding of Y. While the sketches prevent an element $y' \notin \hat{Y}$ from getting too high a score, the top-k operator used to retrieve candidates only has high recall if the elements in $\hat{Y}$ are close in the inner product space. We thus train embeddings to minimize

$$\text{cross\_entropy}(\text{softmax}(\hat{\mathbf{a}_Y}^T, \mathbf{E}), \mathbf{v}_Y / \|\mathbf{v}_Y\|_1)$$

Note that this objective ignores the sketch[4], so it forces the dense representation to do the best job possible on its own. In training $\hat{Y}$ can be primitive set, or the result of a computation (see § 3.3.1).

[3]Of course, directly storing 100 embeddings is less useful for modeling, since that representation does not support operations like relation following or intersection.

[4]The sketch is not used for this objective, but is used in § 3.3.1 where we train a QA system which includes EmQL as a component. Hence in general it is necessary for inference with the sketch to be differentiable.

3.3 Experiments

We evaluate EmQL first intrinsically for its ability to model set expressions [61], and then extrinsically as the reasoning component in two multi-hop KB question answering benchmarks (KBQA).

3.3.1 Learning to reason with a KB

3.3.1.1 Dataset

To evaluate performance in reasoning over KBs, we follow the procedure of Ren et al. [128] who considered nine different types of queries, as summarized in Table 3.1, and data automatically constructed from three widely used KB completion (KBC) benchmarks. Briefly, to evaluate performance for QE, Ren et al. first hold out some triples from the KB for validation and test, and take the remaining triples as the *training KB*. Queries are generated randomly using the query templates of Table 3.1. The gold answer for a query is the traditional logical evaluation on the *full KB*, but the QE system is trained to approximate the gold answer using only the smaller *training KB*. Queries used to evaluate the system are also constrained to *not* be fully answerable using only logical entailment over the training KB. We modify the original Q2B task to test the ability to infer logically entailed answers. EmQL and Q2B were trained with the full KB instead of the training KB, so only reasoning (not generalization) is required to find answers. As we argue above, it is important for a query language to be also be faithful to the KB when answering compositional logical queries.

	Query Template		Query Template
1p	$X.\mathit{follow}(R)$	ip	$(X_1.\mathit{follow}(R_1) \cap X_2.\mathit{follow}(R_2)).\mathit{follow}(R)$
2p	$X.\mathit{follow}(R_1).\mathit{follow}(R_2)$	pi	$X_1.\mathit{follow}(R_1).\mathit{follow}(R_2) \cap X_2.\mathit{follow}(R_3)$
3p	$X.\mathit{follow}(R_1).\mathit{follow}(R_2).\mathit{follow}(R_3)$	2u	$X_1.\mathit{follow}(R_1) \cup X_2.\mathit{follow}(R_2)$
2i	$X_1.\mathit{follow}(R_1) \cap X_2.\mathit{follow}(R_2)$	up	$(X_1.\mathit{follow}(R_1) \cup X_2.\mathit{follow}(R_2)).\mathit{follow}(R)$
3i	$X_1.\mathit{follow}(R_1) \cap X_2.\mathit{follow}(R_2) \cap X_3.\mathit{follow}(R_3)$		

Table 3.1: Nine query templates used. Query2Box is trained on templates 1p, 2p, 3p, 2i, and 3i. EmQL is trained on a variation of 1p and set intersection.

3.3.1.2 Baselines

We compare EmQL to two baseline models: GQE [61] and Query2Box (Q2B) [128]. GQE represents logical queries as learned geometric operations in the embedding space. Query2Box is the state-of-the-art QE model that learns to embed queries to box embeddings. Both models use the learned query embeddings to score candidate answers. The numbers are shown in Table 3.3.[5] Query2Box is trained on examples from only five reasoning tasks (1p, 2p, 3p, 2i, 3i), with the remainder held out to measure the ability to generalize to new query templates.[6] EmQL was trained on only two tasks:

[5]Some numbers in the table are different from the original publication at [142] because we fixed a bug in the evaluation metrics.

[6]Of ccourse, test queries are always distinct from training queries.

	Entities	Relations	Training Triples	Test Triples	Total Triples
FB15k	14,951	1,345	533,142	59,071	592,213
FB15k-237	14,505	237	289,641	20,438	310,079
NELL995	63,361	200	128,537	14,267	142,804

	Train			Test	
task	Basic sets	Follow (1p)	Intersection	Follow (1p)	Others
FB15k	11,611	96,750	355,966	67,016	8,000
FB15k-237	11,243	50,711	191,934	22,812	5,000
NELL995	19,112	36,469	108,958	17,034	4,000

Table 3.2: Statistics for the Query2Box datasets. **Top**: Size of splits into train and test for all the Query2Box KBs. **Bottom**: Number of training and testing examples of the Query2Box datasets. Training data for EmQL are derived from the same training KB as Query2Box. EmQL is directly evaluated on the same test data without further fine-tuning.

entailment											
FB15k	Q2B	68.0	39.4	32.7	48.5	**65.3**	16.2	32.9	**61.4**	28.9	43.7
	+d=2000	59.0	36.8	30.2	40.4	57.1	14.8	28.9	49.2	28.7	38.3
	EmQL(ours)	**91.9**	**53.1**	**39.4**	**51.2**	59.6	**29.6**	**39.5**	36.2	**43.6**	**49.3**
	- sketch	88.5	32.6	19.0	48.3	56.4	17.7	31.9	42.3	25.1	40.2
FB15k-237	Q2B	58.5	34.3	28.1	44.7	**62.1**	11.7	23.9	**40.5**	22.0	36.2
	+d=2000	50.7	30.1	26.1	34.8	55.2	11.4	20.6	32.8	21.5	31.5
	EmQL(ours)	**89.4**	**48.2**	**30.8**	**47.4**	54.9	**25.2**	**33.0**	29.8	**38.6**	**44.1**
	- sketch	87.1	29.6	16.0	45.1	51.9	14.2	26.6	32.0	23.0	36.2
NELL995	Q2B	83.9	57.7	47.8	49.9	66.3	19.9	29.6	**73.7**	31.0	51.1
	+d=2000	75.7	49.9	36.9	40.5	60.1	17.1	25.6	63.5	24.4	43.7
	EmQL(ours)	**94.5**	**68.3**	**53.5**	**70.1**	**74.4**	**33.7**	**44.8**	56.6	**65.4**	**62.4**
	- sketch	88.8	40.1	20.3	66.6	70.6	22.6	36.2	59.0	38.3	49.2

Table 3.3: Hits@3 results for all the Query2Box datasets.

relational following (a variation of 1p), and *set intersection.* Specifically we define a "basic set" X to be any set of entities that share the same property y with relation r, i.e. $X = \{x|r(x,y)\}$. In training EmQL to answer intersection queries ($X_1 \cap X_2$), we let X_1 and X_2 be non-disjoint basic sets, and for relational following (1p), X is a basic set and R a singleton relation set. Training, using the method proposed in §3.2, produces entity and relation embeddings, and queries are then executed by computing the EmQL representations for each subexpression in turn.[7] Following the Query2Box paper [128] we use $d = 400$ for their model and report Hits@3. For EmQL, we use $d = 64$, $k = 1000$, $N_W = 2000$ and $N_D = 20$ throughout.

3.3.1.3 Experimental Results

The results in Table 3.3 show EmQL dramatically outperforms Q2B on all tasks in this setting. To see if larger embeddings would improve Q2B's performance on entailment tasks, we increased

[7]In particular, intermediate EmQL representations are never "decoded" by converting them to entity lists.

the dimension size to $d = 2000$, and observed a decrease in performance, relative to the tuned value $d = 400$ [128].[8] In the ablation experiment(EmQL−sketch), we remove the sketch and only use the centroid to make predictions. The results are comparable for generalization, but worse for entailment.

3.3.2 Question answering

3.3.2.1 Datasets

The statistics of MetaQA and WebQuestionsSP datasets are listed in Table 3.4. For WebQuestionsSP, we used a subset of Freebase obtained by gathering triples that are within 2-hops of the topic entities in Freebase. We exclude a few extremely common entities and restrict our KB subset so there are at most 100 tail entities for each subject/relation pair (reflecting the limitation of our model to sets of cardinality less than 100).

	Train	Dev	Test
MetaQA 2-hop	118,980	14,872	14,872
MetaQA 3-hop	114,196	14,274	14,274
WebQuestionsSP	2,848	250	1,639

	Triples	Entities	Relations
MetaQA	392,906	43,230	18
WebQuestionsSP	1,352,735	904,938	695

Table 3.4: Statistics for the MetaQA and WebQuestionsSP datasets. **Top**: Number of train/dev/test data. **Bottom**: Size of KB.

3.3.2.2 Baselines

To evaluate QE as a neural component in a larger system, we followed ReifKB in §2 and embed EmQL as a reasoning component in a KBQA model. The reasoner of ReifKB is a sparse-matrix "reified KB" rather than a QE method, which does not generalize, but is perfectly faithful for entailment questions. In this section we evaluate replacing ReifKB with EmQL.

ReifKB was evaluated on two KBQA datasets, MetaQA [184] and WebQuestionsSP [179], which access different KBs. For both datasets, the input to the KBQA system is a question q in natural language and a set of entities X_q mentioned in the question, and the output is a set of answers Y (but no information is given about the latent logical query that produces Y.) EmQL's set operations were pre-trained for each KB as in § 3.3.1, and then the KB embeddings were fixed while training the remaining parts of the QA model.

[8]Here $d = 2000$ was the largest value of d supported by our GPUs. Note this is still much smaller than the number of entities, which would be number required to guarantee arbitrary sets could be memorized with boxes. .

3.3.2.3 MetaQA Model

We used similar models as those used for ReifKB. The model for 2-hop questions is given on the left of Table 3.5, where W_1 and W_2 are learned parameters, and $\mathbf{b}_{\mathbb{I}}$ is a vacuous sketch, and *encode*(q) is obtained by pooling the (non-contextual) embeddings of words in q. The 3-hop case is analogous.[9]

MetaQA	**WebQuestionsSP**
$\hat{Y} = X_q.follow(R_1).follow(R_2) - X_q$	$X_1 = X_q.follow(R_1^e)$
$R_1 = (\mathbf{a}_1, \mathbf{b}_{\mathbb{I}}),\ \mathbf{a}_1 = W_1^T encode(q)$	$X_2 = X_q.follow(R_1^{cvt}).follow(R_2^e)$
$R_2 = (\mathbf{a}_2, \mathbf{b}_{\mathbb{I}}),\ \mathbf{a}_2 = W_2^T encode(q)$	$\hat{Y} = X_1 \cup X_2 \cup (X_1 \cup X_2).filter(R_3, Z)$

Table 3.5: EmQL models for MetaQA and WebQuestionsSP datasets.

MetaQA makes use of the set difference operation. For example, to answer the question "What are other movies that have the same director as *Inception*?", we need to first find the director of *Inception*, *Christopher Nolan*, and all movies directed by him. Since the question above asks about *other* movies, the model should also remove the movie *Inception* from this set to obtain the final answer set Y. Thus in the first line of our model, we write

$$\hat{Y} = X_q.follow(R_1).follow(R_2) - X_q$$

For MetaQA, the entity embedding is just a learned lookup table. The question representation *encode*(q) is computed with a bag-of-word approach, i.e., an average pooling on the word embeddings of question q. The embedding size is 64, and scaling parameter for relation λ is 1.0. Our count-min sketch has depth $N_D = 20$ and width $N_W = 500$. We set $k = 100$ to be the number of entities we retrieve at each step, and we pre-train KB embeddings and fix the embeddings when training our QA model.

3.3.2.4 WebQuestionsSP Model

The model we use (see Table 3.5, right) is similar to MetaQA model, except that the final stage is a union of several submodels—namely, chains of one and two follow operations, with or without relational filtering. The submodels for the R's also similar to those for MetaQA, except that we used a BERT [41] encoding of the question, and augmented the entity embeddings with pre-trained BERT representations of their surface forms.

We use pre-trained BERT to encode our question q, i.e., *encode*(q) is the BERT embedding of the [CLS] token. The relation sets R_1, R_2, R_3 are linear projections of the question embedding *encode*(q) paired with a vacuous all-ones sketch $\mathbf{b}_{\mathbb{I}}$. Relation centroids are stacked with one extra dimension that encodes the hard-type of entities: here the hard-type is a binary value that indicates if the entity is a *cvt* node or not.

[9] An important difference is that in ReifKB R_1 and R_2 are k-hot representations of sets of relation ids, not centroids in embedding space.

For this dataset, to make the entities and relations easier to predict from language, the embedding of each entity was adapted to include a transformation of the BERT encoding of the surface form of the entity names. Let $\mathbf{e}_x^0$ be the embedding of the [CLS] token from a BERT [41] encoding of the canonical name for entity x, and let $\mathbf{e}_x^1$ be a vector unique to x. Our pre-trained embedding for x is then $\mathbf{e}_x = \left[W^T\mathbf{e}_x^0; \mathbf{e}_x^1\right] p$, where W is a learned projection matrix. The embedding of relation r is set to the BERT encoding ([CLS] token) of the canonical name of relation r. In this experiments the BERT embeddings are transformed to 128 dimensions and the entity-specific portion $\mathbf{e}_x^1$ has a dimension of 32. The scaling parameter for relation λ is 0.1.

The KB embedding is fixed after pre-training. We use a count-min sketch with depth $N_D = 20$ and width $N_W = 2000$, and we retrieve $k = 1000$ intermediate results at each step.

The model in ReifKB does not support relational filtering[10], so for it we report results with the simpler model $\hat{Y} = X_1 \cup X_2$. Of course the EmQL models also differ in being QE models, so they model relations as centroids in embedding space instead of k-hot vectors.

3.3.2.5 Experimental Results

In addition to ReifKB, we report results for GRAPH-Net [150] and PullNet [144], which are Graph-CNN based methods. EmbedKGQA [137] is the current state-of-the-art model that applies KB embeddings ComplEx [157] in KBQA. Since our models make heavy use of the *follow* operation, which is related to a key-value memory, we also compare to a key-value memory network baseline [99]. The results are shown on Table 3.6.

On MetaQA3 and WebQSP datasets we exceed the previous state-of-the-art by a large margin (7.7% and 5.8% hits@1 absolute improvement), and results on MetaQA2 are comparable to the previous state-of-the-art. We also consider two ablated versions of our model, EmQL-sketch and EmQL-filter. EmQL-filter uses the same model as used in ReifKB (§2), but still improves over ReifKB significantly, showing the value of coupling learning with a QE system rather than a localist KB. EmQL-sketch disables sketches throughout (rather than only in the submodels for the R's), and works consistently worse than the full model for all datasets. Thus it underscores the value of *faithful QE* for KBQA tasks. (Notice that errors in computing entailed answers will appear to the KBQA system as noise in the end-to-end training process.) Finally, Figure 3.2 (right) shows that performance of EmQL-sketch is improved only slightly with much larger embeddings, also underscoring the value of the sketches.

As noted above, the QE component was first pre-trained on synthetic queries (as in § 3.3.1), and then the QA models were trained with fixed entity embeddings. We also jointly trained the KB embeddings and the QA model for MetaQA3, using just the QA data. In this experiment, we also varied k, the top-k entities retrieved at each step. Figure 3.2 (left) shows pre-training the KB

[10]The relational filtering operation is not defined for ReifKB, although it could be implemented with sequences of simpler operations.

	MetaQA2	MetaQA3	WebQSP
KV-Mem	82.7	48.9	46.7
GRAFT-Net	94.8	77.7	70.3
PullNet	**99.9**	91.4	69.7
EmbedKGQA	98.8	94.8	66.6
ReifKB	81.1	72.3	52.7
EmQL (ours)	98.6	**99.1**	**75.5**
– filter	–	–	65.2
– sketch	70.3	60.9	53.2

Table 3.6: Hits@1 of WebQuestionsSP, MetaQA2, and MetaQA3. GRAFT-Net and PullNet were re-run on WebQuestionsSP with oracle sets of question entities X_q.

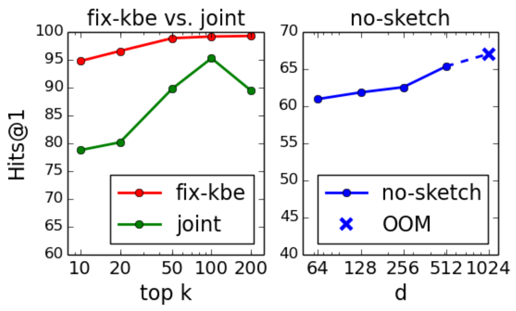


Figure 3.2: Hits@1 on MetaQA3. **Left:** Jointly training (joint) or fixed QE (fix-kbe) varying k for top k retrieval. **Right:** Varying d for EmQL-sketch.

embeddings (fix-kge) consistently outperforms jointly training KB embeddings for the QA model (joint), and demonstrates that pre-training QE on simple KB queries can be useful for downstream tasks.

3.4 Related Work

KBE and reasoning. There are many KB embedding (KBE) methods, surveyed in [166]. Typically KBE methods generalize a KB by learning a model that scores the plausibility of a potential KB triple $r(x, y)$, where r is a KB relation, x is a head (aka subject) entity, and y is a tail (aka object) entity. In nearly all KBE models, the triple-scoring model assumes that every entity x is represented by a vector $\mathbf{v}_x$.

Traditional query languages for symbolic KBs do support testing whether a triple is present in a KB, but also allow expressive compositional queries, often queries that return sets of entities. Several previous works also propose representing *sets* of entities with embeddings [164, 163, 182]; box embeddings [163], for instance, represent sets with axis-parallel hyperrectangles.

Many KBE models also support *relation projection*, sometimes also called *relation following*. *Relation following* [27] maps a set of entities X and a set of relations R to a set of entities related to something in X via some relation in R: here we use the notation $X.follow(R) \equiv \{y \mid \exists x \in X, r \in R : r(x, y)\}$. Many KBEs naturally allow computation of some soft version of relation following, perhaps restricted to singleton sets.[11] However, most KBE methods give poor results when relation following operations are composed [60], as in computing $X.follow(R_1).follow(R_2)$. To address this, some KBE systems explicitly learn to follow a path (aka chain) of relations [60, 88, 32].

Extending this idea, the graph-query embedding (GQE) method [61] defined a query language

[11]E.g., translational embedding schemes like TransE [15] would estimate the embedding for y as $\hat{\mathbf{e}}_y = \mathbf{e}_x + \mathbf{e}_r$, where $\mathbf{e}_x$, and $\mathbf{e}_r$ are vectors embedding entity x and relation r respectively. Several other methods [60, 91] estimate $\hat{\mathbf{e}}_y = \mathbf{e}_x\mathbf{M}_r$ where $\mathbf{M}_r$ is a matrix representing r.

containing both relation following and set intersection. In GQE inputs and output to the relation following operation are entity sets, defined by cosine-distance proximity to a central vector. More recently, the Query2Box [128] method varied GQE by adopting a box embedding for sets, and also extended GQE by including a set union operator. In Query2Box unions are implemented by rewriting queries into a normal form where unions are used only as the outermost operation, and then representing set unions as unions of the associated boxes.

Quantum logic [152] is another neural representation scheme, which might be considered a query language. It does not include relation following, but is closed under intersection and negation, and approximately closed under union (via computation of an upper bound on the union set.)

Other studies [37, 126] use logical constraints such as transitivity or implication to improve embeddings. Here we go in the opposite direction, from KBE to reasoning, answering compositional queries in an embedded KB that is formed in the absence of prior knowledge about relations.

Sparse-matrix neural reasoning. An alternative to representing entity sets with embeddings is to represent sets with "k-hot" vectors. Set operations are easily performed on k-hot vectors[12] and relation following can be implemented as matrix multiplication [26]. Such "localist" representations can exactly emulate logical, hence faithful, reasoning systems. However, they do not offer a direct way to generalize because entities are just (represented by) array indices.

3.5 Discussion

EmQL is a new query embedding (QE) method, which combines a novel centroid-sketch representation for entity sets with neural retrieval over embedded KB triples. In this chapter we showed that EmQL generalizes well, is differentiable, compact, scalable, and faithful with respect to deductive reasoning. However, there are areas for improvement. Compared to the reified KB method (NQL) [27] proposed in the previous chapter, EmQL learns and generalizes better, and does not rely on expensive sparse-matrix computations; however, unlike a reified KB, it requires KB-specific pretraining to find entity embeddings suitable for reasoning. Like most previous KBE or QE methods, EmQL sets must correspond to neighborhoods in embedding space[13]; EmQL's centroid-sketch representation additionally assumes that sets are of moderate cardinality. Finally, in common with prior QE methods [61, 128], EmQL does not support general logical negation, and has only very limited support for set difference.

In spite of these limitations, we showed that EmQL substantially outperforms previous QE methods in the usual experimental settings, and massively outperforms them with respect to faithfulness. In addition to improving on the best-performing prior QE method, we demonstrated that it is possible to incorporate EmQL as a neural module in a KBQA system: to our knowledge this is the

[12]If $\mathbf{v}_A, \mathbf{v}_B$ are k-hot vectors for sets A, B, then $\mathbf{v}_A + \mathbf{v}_B$ encodes $A \cup B$ and $\mathbf{v}_A \odot \mathbf{v}_B$ encodes $A \cap B$.

[13]A notable exception is Query2Box which represents sets with unions of rectangles using a non-compositional set union operator. Although non-compositional set union could be added to EmQL it is currently not implemented.

first time that a QE system has been used to solve an downstream task (i.e., a task other than KB completion). Replacing a faithful localist representation of a KB with EmQL (but leaving rest of the QA system intact) leads to double-digit improvements in Hits@1 on three benchmark tasks, and leads to a new state-of-the-art on the two more difficult tasks.

In the next chapter, we will relax the assumption of reasoning with a symbolic KB. Instead, we propose to construct a virtual KB (VKB) from a text corpus which supports similar reasoning operaions as NQL and EmQL, but has a broader range of information to answer more questions.

Chapter 4

Open Predicate Query Language (OPQL)

4.1 Overview

So far, we've discussed two query languages for logical inference tasks: NQL which compiles knowledge bases in to sparse matrices and runs logical inference in the symbolic space, and EmQL which embeds triples in KBs into vectors and performs logical operations in the embedding space. Both query languages manage to solve many KB-based question answering tasks using the provided symbolic knowledge bases. However, the application of query language methods in real question answering (QA) systems is largely restricted by the incompleteness of KBs, e.g. Freebase and Wikidata, despite efforts to automate their creation through text extraction [7, 106].

An alternative to extracting information to augment existing KBs is to directly use information provided in text corpora by constructing a virtual KB (VKB). The VKB is built from a text corpus. The key idea for a VKB is to greatly simplify the construction of a KB by building a "soft KB" that is closely related to the original text but still keeps the structure of KB triples with subjects, objects and relations so it naturally supports the embedding-based query languages, e.g. EmQL.

One of the biggest challenges in constructing the VKB is to learn to represent the relationship between a pair of entities as a vector.[1] Different from the relations in the symbolic KB that have a predefined vocabulary and whose embeddings can be looked up from a learned embedding table, relations in the VKB are described in natural language text without a fixed schema. In this work, we discuss our method, OPQL, that learns a relation encoder using a dual-encoder pre-training process. Our process is similar to that used in [9], which was shown to be useful for tasks such as relation

[1]Here, we focus on representing binary relationships between pairs of entities (e_1, e_2), and leave n-ary relations for future work.

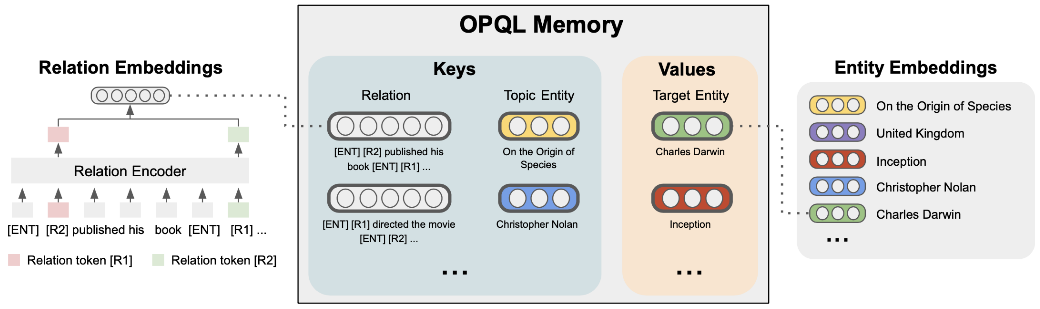


Figure 4.1: **OPQL memory structure.** The OPQL memory is a key-value memory. Keys are computed from embeddings of the topic entity, e.g. *On the Origin of Species*, looked up from the entity embedding table, and relation embeddings from the pretrained relation encoder. Values are embeddings of target entities. The memory is constructed from any entity-linked text corpus, e.g. Wikipedia.

classification. We adopt this pre-training process to learn to compute relation embeddings that are used to construct a VKB that effectively supports complex reasoning operations.

4.2 Method

Entries in the OPQL memory encode the relationship between pairs of entities, described in natural language. For example, a sentence "*Charles Darwin* published his book *On the Origin of Species* in 1859" describes the authorship of entity *On the Origin of Species*. Importantly, these unstructured relationships expressed in text can be extremely fine-grained, covering semantics that would never be included in a pre-defined KB schema. This is made possible because OPQL has the ability of learning without any structured supervision. These relationships are organized into a key-value memory index. Each key is the composition of a topic entity (e.g. *On the Origin of Species*) and an associated latent relationship expressed in text, constructed using pretrained entity embeddings and a pretrained text encoder. Its value is the corresponding target entity, in this case, *Charles Darwin*. When the memory is queried, the returned value can be used directly to answer the input query.

4.2.1 Background

Input The input to the model used for pretraining OPQL is a sequence of n tokens $C = [c_0, c_1 ... c_n]$ for an arbitrary text span, e.g. "*Charles Darwin* published his book *On the Origin of Species* in 1859", that contains a set of demarcated entity mentions M. A mention "*Charles Darwin*" $\in M$ is denoted as $m = (c_i, c_j, e_m) \in M$, where i and j denote the first and last token of the mention span and the mention is linked to entity e_m in a predefined entity vocabulary $\mathcal{E}$, e.g. Charles_Darwin (Q1035) in Wikidata.

Enity Pairs OPQL learns to encode the relationship between a pair of entities (e_1, e_2), where e_1 is referred as topic entity and e_2 as target entity. In the example above, *On the Origin of Species* is the topic entity e_1 and *Charles Darwin* is the target entity $e2$. Potentially[2] each distinct entity pair in a sentence can be encoded. The relationship between a pair of entities is directional, so the pairs (e_1, e_2) and (e_2, e_1) are encoded differently.
Entity Embeddings A global entity embedding table $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d_e}$ is pretrained using the RELIC strategy [90]; here $\mathcal{E}$ is the entity vocabulary and d_e is the embedding dimensionality. Similar to pretraining a masked language model (MLM), an entity mention is randomly masked from the context and the goal is to retrieve the masked entity from the entity vocabulary $\mathcal{E}$.
Key-Value Memory We structure OPQL as a key-value memory. A key-value memory $(\mathbf{K}, \mathbf{V})$ is a general way of storing and retrieving knowledge [99]. When queried, key embeddings $\mathbf{k}_i \in \mathbf{K}$ are matched with the query vector, and the corresponded value embedding $\mathbf{v}_i$ is returned. Each entry in OPQL encodes a pair of entities (e_1, e_2), along with a piece of text C that describes the relationship between e_1 and e_2. The key-value memory is constructed from the pretrained entity embedding table $\mathbf{E}$ and relation embeddings precomputed from a pretrained relation encoder. A key memory holds information from the topic entity and the relationship, and a value memory holds the target entity. We will discuss the pretrained entity and relation embeddings first and then discuss how to construct the key-value memory using the pretrained embeddings.

4.2.2 Relation Encoder

Preprocessing Text for the Relation Encoder OPQL encodes the relationship between a pair of entities (e_1, e_2) in a natural language sentence C where both e_1 and e_2 are mentioned. Intuitively C describes the relationship between e_1 and e_2 in the context, and we train a relation encoder to represent the relationship as a vector.

To indicate the location of the topic and target entities e_1 and e_2 we introduce two special tokens [R1] and [R2] that are inserted directly after the mentions of the topic and target entities in the sentence (e.g., C_r becomes "*Charles Darwin* [R2] published his book *On the Origin of Species* [R1] in 1859"). The contextual encodings of [R1] and [R2] will be used to compute relation embeddings. We also introduce another special token [ENT] to mask the topic and target entity mentions; thus sentence C_r finally becomes "[ENT] [R2] published his book [ENT] [R1] in 1859". Masking the mentions of entities prevents the relation encoder from memorizing the surface forms of the entities, and helps it generalize to similar relations involving other entities.
Computing Relation Embedding Given this preprocessing of an entity-mention pair (e_1, e_2), the relation embedding $\mathbf{r}_{e_1,e_2}$ is defined as follows. Let s and t be the location of tokens [R1] and [R2] in the masked sentence C_r. We construct a projection of the concatenation of the contextual

[2]Heuristics for limiting the number of entity pairs derived from text are discussed later in this chapter.

embeddings $\mathbf{h}_s$ and $\mathbf{h}_t$ at the locations s and t, as follows:

$$\mathbf{r}_{e_1,e_2} = \mathbf{W}_r^T \; [\mathbf{h}_s; \mathbf{h}_t] \tag{4.1}$$

Training the Relation Encoder We train our relation encoder following [9], who train embeddings such that relation mentions containing the same entity pairs are more similar to each other than relation mentions that contain different entity pairs.

Specifically, mini-batches are constructed which contain at least two documents that contain entity pair (e_1, e_2), as well as negative documents containing different relation pairs. Use $\mathbf{r}_{e_1,e_2}$ to denote the embedding from input C that of (e_1, e_2), use $\mathbf{r}_{e_i,e_j}$ for embeddings from documents $C'_0, \ldots, C'_n$ of other pairs (e_i, e_j)'s, and let $\mathbb{I}_{e_i=e_1,e_j=e_2}$ be an equality indicator for entity pairs in the minibatch. We maximize the inner product between relation embeddings iff the same pair (e_1, e_2) is mentioned in the candidates:

$$L_{\text{rel}} = \text{cross\_ent}(\text{softmax}({\mathbf{r}_{e_1,e_2}}^T \mathbf{r}_{e_i,e_j}), \mathbb{I}_{e_i=e_1,e_j=e_2}) \tag{4.2}$$

4.2.3 Entity Linking

Additionally, we use a multi-task training objective to learn representations of individual entities by learning to link other mentions in the context—i.e., mentions other than the topic and target entities—to the correct entity. For example, in the masked sentence C_r, “[ENT] [R2], an *English* naturalist, published his book [ENT] [R1] in 1859”, the mention *English* is not part of the pair (*On the Origin of Species*, *Charles Darwin*) and thus not masked with [ENT], but should be linked to an entity for “England”. These *context entity* mentions are represented as $m_{e_a} = (c_i, c_j, e_a)$ where c_i and c_j are the start and end positions, and e_a is the entity to which this mention should be linked. We construct an embedding of mentions m_{e_a} from the contextual embedding $\mathbf{h}_i$ at the start position c_i for the mention, i.e. $\mathbf{m}_{e_a} = \mathbf{W}_e^T \mathbf{h}_i$. This is used to retrieve the most similar entity from the embedding table $\mathbf{E}$, scored with inner product distance, using this loss:

$$L_{\text{el}} = \text{cross\_ent}(\text{softmax}({\mathbf{m}_{e_a}}^T \mathbf{e}_i), \mathbb{I}_{e_i=e_a}) \tag{4.3}$$

4.2.4 Storing OPQL's VKB as a key-value memory

OPQL stores relationships between pairs of entities in a key-value memory $(\mathbf{K}, \mathbf{V})$. Given an input C that mentions a pair of entities e_1 and e_2, the key embedding $\mathbf{k}_{e_1,e_2}$ for the pair (e_1, e_2) is constructed compositionally using the embeddings of topic entities $\mathbf{e}_1$ from the entity embedding table $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d_e}$ and the pretrained relation embedding $\mathbf{r}_{e_1,e_2}$. $\mathbf{W}_k$ is a linear projection matrix that will be learned in the finetuning tasks. The value $\mathbf{v}_{e_1,e_2}$ is the embedding of the target entity

$\mathbf{e}_2$.

$$\mathbf{k}_{e_1,e_2} = \mathbf{W}_k^T\ [\mathbf{e}_1; \mathbf{r}_{e_1,e_2}] \in \mathbf{K}, \quad \mathbf{v}_{e_1,e_2} = \mathbf{e}_2 \in \mathbf{V}$$

We iterate through all sentences in the Wikipedia corpus that contain two or more entities to construct the OPQL memory. Note that each entity pair can be mentioned one or multiple times in the corpus. The memory could keep all mentions of an entity pair (e_1, e_2), or instead reduce multiple mentions of the same entity pair to a single entry, for example averaging the key embedding of the individual mentions. This choice can be made based on application, or computational constraints. We can keep all mentions of entity pairs in the OPQL memory or randomly select a few mentions for each entity pair and average their key embedding as the final entry in the OPQL memory.

4.2.5 Pretraining Data

Entity Linking Data We use Wikipedia passages with hyperlinks as our pretraining. The hyperlinks link a span of tokens to a Wikipedia page for an entity e. The spans of tokens that are linked are considered as mentions m, and entity e is the corresponding entity. We take the top 1M entities that are most frequently mentioned in Wikipedia as our entity vocabulary $\mathcal{E}$. Passages are split into text pieces of 128 tokens. Entity linking is trained on the mentions of entities in the context, excluding the ones that are treated as topic and target entities. The pretraining corpus contains 93.5M mentions of the top 1M entities in the vocabulary.

Relation Encoder Data We first construct the vocabulary of entity pairs (e_1, e_2) from the Wikipedia corpus. We count the pairs of entities that co-occur in the same text piece, and discard the that appears less than 5 times. The pairs are sorted by their point-wise mutual information (PMI). The top 800k pairs of entities are selected as our candidates for training. During training, an input with pair (e_1, e_2) will be paired with 2 positive examples that mentions the same pair of entities, and 8 hard negative examples (e_i, e_j) that mention either the same topic or target entity, i.e. $e_i = e_1$ or $e_j = e_2$, but not both. Batch negatives are also included in pretraining. 30.6M training data on the top 800k entity pairs are constructed from the Wikipedia corpus.

Pretraining Loss The relation encoder is pretrained with entity linking loss L_{el} in Eq. 4.3 and relation encoder loss L_{rel} in Eq. 4.2, i.e. $L = L_{\text{el}} + L_{\text{rel}}$.

4.3 Experiments: Reasoning Over VKB

4.3.1 Datasets

MetaQA is a multi-hop QA dataset that we used in the previous chapters.

Multi-hop Slot Filling (MSF) [45] presents a large scale multi-hop reasoning dataset constructed from WikiData that contains 120k passages, 888 relations, and more than 200k entities. Queries are constructed from multi-hop paths in WikiData and turned into natural language questions by

	MetaQA		MSF	
Model	2Hop	3Hop	2Hop	3Hop
KVMem	7.0	19.5	3.4	2.6
DrQA	32.5	19.7	14.1	7.0
GRAFT-Net	36.2	40.2	-	-
PullNet	81.0	78.2	-	-
PIQA	-	-	36.9	18.2
DrKIT	86.0	**87.6**	46.9	24.4
OPQL-pretrained	84.7	84.3	48.5	28.1
OPQL	**88.5**	87.1	**49.2**	**29.7**

Table 4.1: Hits@1 results on multi-hop relational following task.

concatenating the head entity with a series of relations, e.g. ("Steve Jobs, founder, headquarter in, ?"). Similar to MetaQA, we constructed entity pairs by taking the Wikipedia page title as the topic entity and the entities mentioned in the passages as target entities. We extracted 1.3m and 781k entity pairs for 2-hop and 3-hop questions respectively.

4.3.2 Baselines

GRAFT-Net [149] is a GCN based model that can perform reasoning jointly over text and knowledge bases. PullNet [144] extends GRAFT-NET by introducing an iterative retrieve-and-classify mechanism to solve multi-hop questions. PIQA [138] and DrKIT [45] build mention-level indexes using a pretrained encoder. Training DrKIT requires distant supervision using artificial queries constructed from KB. KV-Mem [99] and DrQA [17] are two widely-used open-domain QA baselines.

4.3.3 Results

We experiment on the relational following task on both a pretrained memory (OPQL-pretrained) and a finetuned memory (OPQL). The Hits@1 results of the model are listed in Table 4.1. The OPQL-pretrained memory directly applies the pretrained relation encoder on entity pairs extracted from the dataset corpus, without any finetuning of the relation encoder. OPQL-pretrained achieves the state-of-the-art performance on both MSF 2-hop and 3-hop datasets, though OPQL-pretrained is slightly lower than DrKIT on the MetaQA. This is because DrKIT finetuned its mention encoder on MetaQA corpus. The MetaQA corpus only contains 14 relations (including their inverse) in the movie domain, so it's easy for the model to learn only capturing these relationship between entities. To mitigate this bias, we finetune the OPQL relation encoder on MetaQA corpus. The finetuning data is distantly constructed from 1-hop questions in the MetaQA dataset, by masking the topic entity from the question and inserting a placeholder for the target entity at the end of the question. We end up with 10K finetuning data for MetaQA and 19K for MSF. OPQL with finetuned memory outperforms DrKIT on 2-hop questions by 2.5 points and is very comparable on 3-hop questions.

4.3.4 Generalization to Novel Relations

The previous state-of-the-art model, DrKIT, uses training data in its pretraining procedure that is distantly constructed from KB. This restricts the capacity of DrKIT to only encode KB relations observed at pretraining time. However, pretraining OPQL does not require any signal from knowledge bases, so it can also encode relations that are out of the relation vocabulary in KB. To demonstrate this difference, we hold out some portions of relations in the pretraining phase of DrKIT but evaluate on queries where at least one of the held-out relations is required to answer the queries. This simulates a scenario where KBs are incomplete and limited in their relation vocabularies. We compare DrKIT to the pretrained OPQL memory. The result is shown in Figure 4.2. The performance of DrKIT drops significantly when evaluated on queries with novel relations not seen at pretraining time, while the performance of OPQL is consistent.

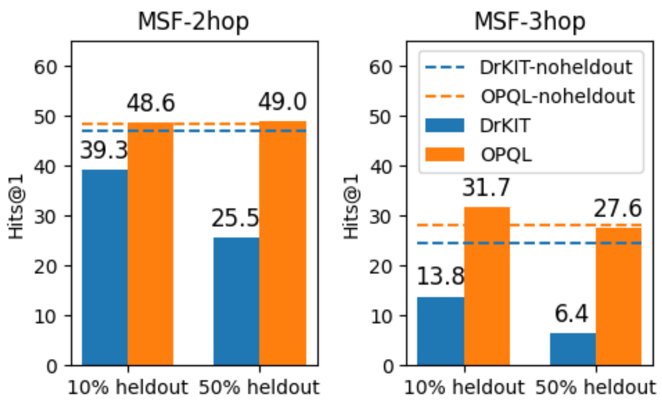


Figure 4.2: Hits@1 on multi-hop queries containing at least one novel relation. The dashed lines represent the accuracy on the full dataset from Table 4.1.

4.4 Related Work

Automatically extracting triples from a text corpus has been pursued for many years as a means of improving the coverage of KBs [106]. Rather than extracting triples into a predefined vocabulary, OpenIE (typically) uses linguistic patterns to extract open vocabulary relations from text [48, 49]. [58]) build an Open Knowledge Graph over Open IE extractions similar to a VKB.

A few methods [138, 45] have built pre-computed memories of mention embeddings which are used directly to answer questions. These methods are most similar to our work though OPQL differs by embedding mention pairs, rather than single mentions, and additionally, our method requires no structured supervision. Another related of work proposed to build a passage level index [70] to improve retrieval accuracy in the "retrieve and read" pipeline. Multi-hop retrieval models [120] are proposed for complex questions. Other approaches propose to retrieve embedded passages which are then passed to an LM to reason over [70, 81, 59, 78]. In contrast OPQL does not include a separate model for reading retrieved documents.

4.5 Discussion

We proposed OPQL that can construct a virtual knowledge base from a text corpus without any supervision from existing KB. The pretrained OPQL can effectively solve relational following task, achieving the state-of-the-art performance on two multi-hop relational following datasets. The improvement is more significant if evaluated on queries with relations not seen at training time. OPQL can be injected into a language model to answer open-domain questions. It outperforms several large pretrained language models on two benchmark open-domain QA datasets.

So far, we've discussed three query languages, NQL, EmQL, and OPQL, to solve challenging reasoning tasks in different settings – i.e. reasoning with symbolic KBs in a symbolic or embedding space, and even using virtual KBs. Queries of the reasoning tasks in the experiments are often structured or semi-structured. However, questions asked in the real world are often more complex in semantics and therefore require more efforts in question understanding. Recent research in Language Models (LMs) suggests pretrained LMs are good semantic encoders. We would like to explore the combinations of the reasoning modules and pretrained LMs for question answering. In the next part, we will discuss FILM and OPQL-LM.

Part II

Integrating Reasoning with Language Models

Chapter 5

Fact-Injected Language Model (FILM)

5.1 Overview

Language Models (LM)[39, 92, 23, 5] are pretrained on large text corpora with self-supervision tasks, such as masked language modeling, to learn semantic and syntactic information. While large LMs can be coerced to answer factual queries, it's correctness and faithfulness to factual knowledge is limited. Furthermore, they lack the reasoning abilities from complex queries that are discussed above, e.g. multi-hop reasoning. It is also difficult to distinguish answers produced by memorizing factual statements in the pre-training corpus from answers produced by linguistic generalization [119].

In this chapter, we propose to integrate the previously proposed embedding-based KB reasoning module, EmQL, to a pretrained language model, called FILM (Fact-Injected Language Model). Different from text-only LMs, e.g. BERT and RoBERTa [41], FILM additionally includes a *fact memory* that represents external knowledge from KBs. The model learns to query from the fact memory to find relevant information to perform logical reasoning, and then mix the reasoning results with the LMs to make the final predictions.

In addition to equip LMs with better reasoning ability to handle complex queries, the KB augmented models also have the flexibility of adding or removing factual information without retraining the LM, an expensive process[1]. The difficulty of updating knowledge in traditional LMs contrasts with symbolic KBs, where it is very easy to add or modify triples, and is a major disadvantage of using a LM "as a KB"—as in many domains (news, product reviews, scientific publications, etc) the

[1]Models large enough to achieve good factual coverage require extreme amounts of compute, and the largest neural LMs now cost millions of dollars to train [16].

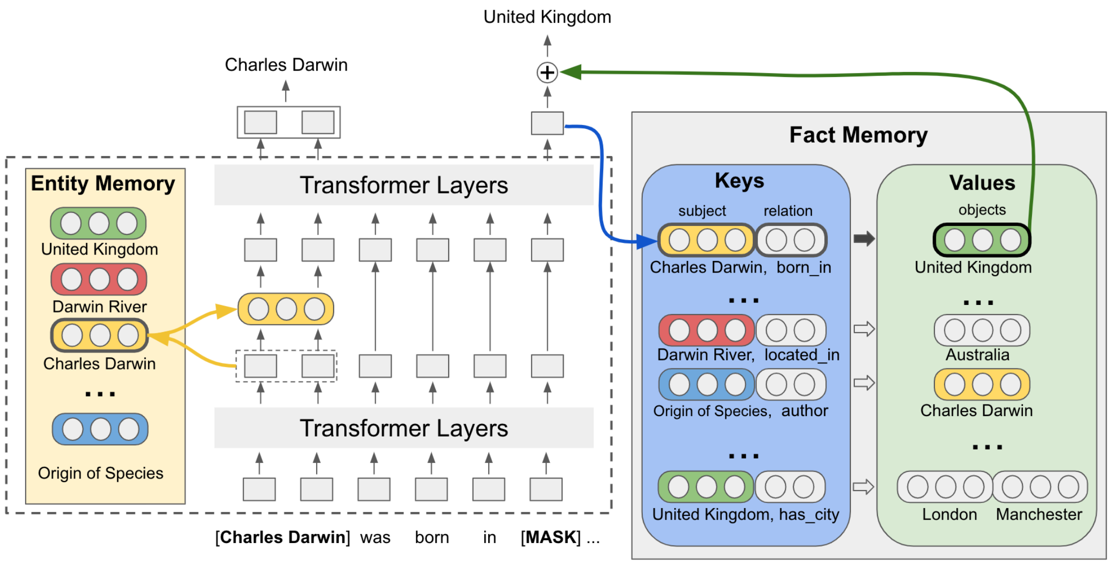


Figure 5.1: **Fact Injected Language Model architecture.** The model takes a piece of text (a question during fine-tuning or arbitrary text during pre-training) and first contextually encodes it with an entity enriched transformer. FILM uses the contextually encoded MASK token as a query to the fact memory. In this case, the contextual query chooses the fact key (*Charles Darwin, born_in*) which returns the set of values {*United Kingdom*} (The value set can be multiple entity objects such as the case from calling the key [*United Kingdom, has_city*]) . The returned object representation is incorporated back into the context in order to make the final prediction. Note that the entity representations in the facts (both in keys and values) are shared with the entity memory. The portion within the dashed line follows the procedure from [51].

set of known facts changes frequently. Since triples in the fact memory are defined compositionally from (representations of) entities and relations, the fact memory can be easily extended with new facts and the model can use them to make new predictions correspondingly.

5.2 Method

The Fact Injected Language Model (FILM) model (see Figure 5.1) extends the Transformer [158] architecture of BERT [42] with additional entity and facts memories. These memories store semantic information which can later be retrieved and incorporated into the representations of the transformer. Similar to the approach in [51], entity embeddings will (ideally) store information about the textual contexts in which that entity appears, and by inference, the entity's semantic properties. The *fact memory* encodes triples from a symbolic KB, constructed compositionally from the learned embeddings of the entities that comprise it and implemented as a key-value memory which is used to retrieve entities given their KB properties. This combination results in a neural LM which learns to access information from a symbolic KB.

5.2.1 Definitions

We represent a Knowledge Base $\mathcal{K}$ as a set of triples (s, r, o) where $s, o \in \mathcal{E}$ are the subject and object entities and $r \in \mathcal{R}$ is the relation, where $\mathcal{E}$ and $\mathcal{R}$ are pre-defined vocabularies of entities and relations. A text corpus $\mathcal{C}$ is a collection of paragraphs[2] $\{p_1, \ldots, p_{|C|}\}$. Let $\mathcal{M}$ be the set of entity mentions in the corpus $\mathcal{C}$. A mention m_i is encoded as (e_m, s^p_m, t^p_m), indicating entity e_m is mentioned in paragraph p starting at token position s^p_m and ending at t^p_m. We will usually drop the superscript p and use s_m and t_m for brevity.

5.2.2 Input

The input to our model is a piece of text; either a question during fine tuning (see §5.2.7) or a paragraph, during pre-training. Pretraining is formulated as a cloze-type Question Answering (QA) task: given a paragraph $p = \{w_1, \ldots, w_{|p|}\}$ with mentions $\{m_1, \ldots, m_n\}$, we sample a single mention m_i to act as the cloze answer and replace all tokens of m_i with [MASK] tokens. The entity in $\mathcal{E}$ named by the masked entity is the answer to the cloze question q ('United Kingdon' in the example input of Figure 5.1). Mentions in the paragraph other than m are referred to below as *context mentions*. In the following sections we describe how our model learns to jointly link context entities and predict answer entities.

5.2.3 Entity Memory

Our entity memory $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d_e}$ is a matrix containing a vector for each entity in $\mathcal{E}$ and trained as an entity-masked LM. The model input is a text span containing unlinked entity mentions with known boundaries[3]. Mentions are masked with some probability. Like Entity as Experts (EaE) [51], we interleave standard Transformer [158] layers with layers that accesss the entity memory[4].

Given a piece of text $q = \{w_1, \ldots, w_{|q|}\}$ the contextual embedding $\mathbf{h}^{(l)}_i$ is the output at the i'th token of the l'th intermediate transformer layer. These contextual embeddings are used to compute query vectors that interface with the entity memory.

For each context mention $m_i = (e_{m_i}, s_{m_i}, t_{m_i})$ in q, we form a query vector to access the Entity memory by concatenating the context embeddings for the mention m_i's start and end tokens, $\mathbf{h}^{(l)}_{s_{m_i}}$ and $\mathbf{h}^{(l)}_{t_{m_i}}$ and projecting them into the entity embedding space. We use this query to compute attention weights over the full entity vocabulary and produce an attention-weighted sum of entity

[2]Although we use the term paragraph here, in our experiments we use spans of 128 tokens, which need not follow paragraph boundaries.

[3][51] also showed the model is capable of learning to predict these boundaries. For simplicity, in this work we assume they are given.

[4]We follow the implementation of [51] and have a single entity memory access between the fourth and fifth transformer layers.

embeddings $\mathbf{u}^l_{m_i}$. The result is then projected back to the dimension of the j-indexed contextual token embeddings, and added to what would have been the input to the next layer of the Transformer:

$$\mathbf{h}^{(l)}_{m_i} = \mathbf{W}_e^T[\mathbf{h}^{(l)}_{s_{m_i}}; \mathbf{h}^{(l)}_{t_{m_i}}] \tag{5.1}$$

$$\mathbf{u}^{(l)}_{m_i} = \text{softmax}(\mathbf{h}^{(l)}_{m_i}, \mathbf{E}) \times \mathbf{E} \tag{5.2}$$

$$\tilde{\mathbf{h}}^{(l+1)}_j = \mathbf{h}^{(l)}_j + \mathbf{W}_2^T \mathbf{u}^{(l)}_{m_i}, \quad s_{m_i} < j < t_{m_i} \tag{5.3}$$

After the final transformer layer T, $\mathbf{h}^{(T)}_{m_i}$ is used to predict the context entities $\hat{e_{m_i}}$ and produce a loss with $\mathbb{I}_{e_{m_i}}$, the one-hot label of entity e_{m_i}. Following [51], we supervise the entity access for the intermediate query vector in Eq. 5.1.

$$\begin{aligned}
\hat{e}_{m_i} &= \text{argmax}_{e_i \in \mathcal{E}}(\mathbf{c}^T_{m_i}\mathbf{e}_i) \\
\text{loss}_{\text{ctx}} &= \text{cross\_entropy}(\text{softmax}(\mathbf{c}_{m_i}, \mathbf{E}), \mathbb{I}_{e_{m_i}}) \\
\text{loss}_{\text{ent}} &= \text{cross\_entropy}(\text{softmax}(\mathbf{h}^{(l)}_{m_i}, \mathbf{E}), \mathbb{I}_{e_{m_i}})
\end{aligned}$$

5.2.4 Fact Memory

FILM extends EaE with a second *fact memory*, populated by triples from the knowledge base $\mathcal{K}$, as shown on the right side of Figure 5.1[5]. The fact memory shares its on entity representations with the entity memory embeddings in $\mathbf{E}$, but each element of the fact memory corresponds to a symbolic substructure, namely a key-value pair $((s, r), \{o_1, \ldots, o_n\})$. The key (s, r) is a (subject entity, relation) pair, and the corresponding value $\{o_1, \ldots, o_n\}$ is the list of object entities associated with s and r, i.e. $(s, r, o_i) \in \mathcal{K}$ for $i = \{1, \ldots, n\}$. Conceptually, KB triples with the same subject entity and relation are grouped into a single element. We call the subject and relation pair $a_j = (s, r) \in A$ a *head pair* and the list of objects $b_j = \{o_1, \ldots, o_n\} \in B$ a *tail set*[6].

In more detail, we encode a head pair $a_j = (s, r) \in A$ by concatenating embeddings for the subject entity and relation, and then projecting them linearly to a new head-pair embedding space. More precisely, let $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d_e}$ be the entity embeddings trained in §5.2.3, and $\mathbf{R} \in \mathbb{R}^{|\mathcal{R}| \times d_r}$ be embeddings of relations $\mathcal{R}$ in the knowledge base $\mathcal{K}$. We encode a head pair a as:

$$\mathbf{a}_j = \mathbf{W}_a^T \ [\mathbf{s}; \mathbf{r}] \ \in \mathbb{R}^{d_a}$$

where $\mathbf{s} \in \mathbf{E}$ and $\mathbf{r} \in \mathbf{R}$ are the embeddings of subject s and relation r, and $\mathbf{W}_a$ is a learned linear transformation matrix. We let $\mathbf{A} \in \mathbb{R}^{|A| \times d_a}$ denote the embedding matrix of all head pairs.

Let the answer for q be denoted e_{ans}, and its masked mention $m_{\text{ans}} = (e_{\text{ans}},\ s_{\text{ans}},\ t_{\text{ans}})$. For a

[5] In our experiments we use a single fact memory access after the final (12th) transformer layer.

[6] The size of the tail set b_j can be large for a popular head pair (s, r). In such cases, we randomly select a few tails and drop the rest of them. The maximum size of the tail set is 32 in the experiments.

masked mention m_{ans}, define a query vector to access the fact memory as:

$$\mathbf{v}_{m_{\text{ans}}} = \mathbf{W}_f^T \; [\mathbf{h}^{(T)}_{s_{\text{ans}}}; \mathbf{h}^{(T)}_{t_{\text{ans}}}] \tag{5.4}$$

where $\mathbf{h}^{(T)}_{s_{\text{ans}}}$ and $\mathbf{h}^{(T)}_{t_{\text{ans}}}$ are the contextual embeddings for the start and end tokens of the mention m_{ans}, and $\mathbf{W}_f$ is the linear transformation matrix into the embedding space of head pairs $\mathbf{A}$.

Head pairs in A are scored by the query vector $\mathbf{v}_{m_{\text{ans}}}$ and the top k head pairs with the largest inner product are retrieved. This retrieval process on the fact memory is distantly supervised. We define a head pair to be a *distantly supervised positive example* $a_{\text{ds}} = (s, r)$ for a passage if its subject entity s is named by a context mention m_i and the masked entity e_{ans} is an element of the corresponding tail set, i.e. $e_{\text{ans}} \in b_{\text{ds}}$. When no distantly supervised positive example exists for a passage, it is trained to retrieve a special “null” fact comprised of the s_{null} head entity and r_{null} relation: i.e. $a_{\text{ds}} = (s_{\text{null}}, r_{\text{null}})$ and its tail set is empty. This distant supervision is encoded by a loss function:

$$\begin{gathered}\text{TOP}_k(\mathbf{v}_{m_{\text{ans}}}, \mathbf{A}) = \text{argmax}_{k, j \in \{1,\dots,|A|\}} \mathbf{a}_j^T \mathbf{v}_{m_{\text{ans}}} \\ \text{loss}_{\text{fact}} = \text{cross\_entropy}(\text{softmax}(\mathbf{v}_{m_{\text{ans}}}, \mathbf{A}), \mathbb{I}_{a_{\text{ds}}})\end{gathered}$$

The result of this query is that the tail sets associated with the top k scored head pairs, i.e. $\{b_j | j \in \text{TOP}_k(\mathbf{v}, \mathbf{A})\}$, are retrieved from the fact memory.

5.2.5 Integrating Knowledge and Context

Next, tail sets retrieved from the fact memory are aggregated. Recall that a tail set b_j returned from the fact memory is the set of entities $\{o_1, \dots, o_n\}$ s.t. $(s, r, o_i) \in \mathcal{K}$ for $i \in \{1, \dots, n\}$ with the associated $a_j = (s, r)$. Let $\mathbf{o}_i \in \mathbf{E}$ be the embedding of entity o_i. We encode the returned tail set $\mathbf{b}_j$ as a weighted centroid of the embeddings of entities in the tail set b_j.

$$\mathbf{b}_j = \sum_{o_i \in b_j} \alpha_i \mathbf{o}_i \;\; \in \mathbb{R}^{d_e}$$

where α_i is a context-dependent weight of the object entity o_i. To compute the weights α_i, we use a process similar to Eq. 5.4: we compute a second query vector $\mathbf{z}_{m_{\text{ans}}}$ to score the entities inside the tail set b_j, and the weights α_i are the softmax of the inner products between the query vector $\mathbf{z}_{m_{\text{ans}}}$ and the embeddings of entities in the tail set b_j.

$$\mathbf{z}_{m_{\text{ans}}} = \mathbf{W}_b^T [\mathbf{h}^{(T)}_{s_{\text{ans}}}; \mathbf{h}^{(T)}_{t_{\text{ans}}}] \tag{5.5}$$

$$\alpha_i = \frac{\exp\,(\mathbf{o}_i^T \; \mathbf{z}_{m_{\text{ans}}})}{\sum_{o_l \in b_j} \exp\,(\mathbf{o}_l^T \; \mathbf{z}_{m_{\text{ans}}})} \tag{5.6}$$

where $\mathbf{W}_b$ is a transformation matrix distinct from $\mathbf{W}_e$ in Eq. 5.1 and $\mathbf{W}_f$ in Eq. 5.4.

The top k tail sets $\mathbf{b}_j$ are further aggregated using weights β_j, which are the softmax of the retrieval (inner product) scores of the top k head pairs a_j. This leads to a single vector $\mathbf{f}_{m_{\text{ans}}}$ that we call the *knowledge embedding* for the masked mention m_{ans}.

$$\mathbf{f}_{m_{\text{ans}}} = \sum_{j \in \text{TOP}_k(\mathbf{v}_{m_{\text{ans}}}, \mathbf{A})} \beta_j \mathbf{b}_j \tag{5.7}$$

$$\beta_j = \frac{\exp\left(\mathbf{a}_j^T\ \mathbf{v}_{m_{\text{ans}}}\right)}{\sum_{t \in \text{TOP}_k(\mathbf{v}_{m_{\text{ans}}}, \mathbf{A})} \exp\left(\mathbf{a}_t^T\ \mathbf{v}_{m_{\text{ans}}}\right)} \tag{5.8}$$

Intuitively $\mathbf{f}_{m_{\text{ans}}}$ is the result of retrieving a set of entities from the fact memory. The last step is to integrate this retrieved set into the Transformer's contextual embeddings. Of course, KBs are often incomplete, and especially during pre-training, it might be necessary for the model to ignore the result of retrieval, if no suitable triple appears in the KB. To model this, the final step in the integration process is to construct an integrated query $\mathbf{q}_{m_{\text{ans}}}$ with a learnable mixing weight λ. Algorithmically, λ is computed as the probability of retrieving a special "null" head a_{null} from the fact memory, i.e. whether an oracle head pair exists in the knowledge base. $\mathbf{q}_{m_{\text{ans}}}$ is used to predict the masked entity.

$$\begin{aligned}
\mathbf{q}_{m_{\text{ans}}} &= \lambda \cdot \mathbf{c}_{m_{\text{ans}}} + (1-\lambda) \cdot \mathbf{f}_{m_{\text{ans}}}, \quad \lambda = P(a_{\text{null}}) \\
\hat{e}_{\text{ans}} &= \text{argmax}_{e_i \in \mathcal{E}}(\mathbf{q}_{m_{\text{ans}}}^T \mathbf{e}_i) \\
\text{loss}_{\text{ans}} &= \text{cross\_entropy}(\text{softmax}(\mathbf{q}_{m_{\text{ans}}}, \mathbf{E}), \mathbb{I}_{e_{\text{ans}}})
\end{aligned}$$

The final loss is the sum of the individual losses.

$$\text{loss}_{\text{final}} = \text{loss}_{\text{ent}} + \text{loss}_{\text{ctx}} + \text{loss}_{\text{fact}} + \text{loss}_{\text{ans}}$$

5.2.6 Pretraining Data

FILM is pretrained on Wikipedia and Wikidata using the same data from [51]. Text in Wikipedia is chunked into 128 token pieces. To compute the entity-linking loss loss_{ent}, we use as training data entities linked to the 1 million most frequently linked-to Wikidata entities. Text pieces without such entities are dropped. This results in 30.58 million text pieces from Wikipedia. As described in §5.2.1, we generate n training examples from a piece of text containing n entity mentions, where each mention serves as the masked target for its corresponding example, and other entity mentions in the example are treated as context entities[7]. This conversion results in 85.58 million pre-training examples. The knowledge base $\mathcal{K}$ is a subset of Wikidata that contains all facts with subject and

[7]We mask context entities randomly with probability .15

object entity pairs that co-occur at least 10 times on Wikipedia pages.[8] This results in a KB containing 1.54 million KB triples from Wikidata (or 3.08 million if reverse triples are included). Below, this is called the *full setting* of pretraining—we will also train on subsets of this example set, as described below. We pretrain the model for 500,000 steps with the batch size 2048, and we set $k = 1$[9] in the TOP_k operation for fact memory access.

5.2.7 Finetuning on Question Answering

In open-domain Question Answering tasks, questions are posed in natural language, e.g. "Where was Charles Darwin born?", and answered by a sequence of tokens, e.g. "United Kingdom". We focus on a subset of questions that are answerable using entities from a knowledge base. In the example above, the answer "United Kingdom" is an entity in Wikidata whose ID is Q145.

We convert an open-domain question to an input of FILM by appending the special [MASK] token to the end of the question, e.g. {'Where', 'was', 'Charles', 'Darwin', 'born', '?', [MASK]}. The task is to predict the entity named by mask. Here, "Charles Darwin" is a context entity, which is also referred to as *question entity* in the finetuning QA task.

At finetuning time, entity embeddings $\mathbf{E}$ and relation embeddings $\mathbf{R}$ are fixed, and we finetune all transformer layers and the four transformation matrices: $\mathbf{W}_a$, $\mathbf{W}_b$, $\mathbf{W}_e$, $\mathbf{W}_f$. Parameters are tuned to optimize unweighted sum of the the fact memory retrieval loss $\text{loss}_{\text{fact}}$ and the final answer prediction loss loss_{ans}. If multiple answers are available, the training label $\mathbb{I}_{e_{\text{ans}}}$ becomes a k-hot vector uniformly normalized across the answers.

$$\text{loss}_{\text{QA}} = \text{loss}_{\text{fact}} + \text{loss}_{\text{ans}}$$

5.3 Experiments

The primary focus of this work is investigating the incorporating of new symbolic knowledge by injecting new facts without retraining (§5.4.1) and updating stale facts (§5.4.2). However, we first validate the efficacy of our model on standard splits of widely used knowledge-intensive benchmarks against many state-of-the-art systems (§5.3.4), as well as two subsets of these benchmarks restricted to examples answerable with wikidata (§5.3.5) and examples filtered for train/test overlap (§5.3.6).

[8]This leads to more KB triples than entity pairs, since a pair of entities can be connected by more than one relation.

[9]We experimented with other values of k during fine tuning and evaluation but did not observe significant differences.

5.3.1 Data

We evaluate on four knowledge intensive tasks. WebQuestionsSP is an Open-domain Question Answering dataset that has been used in the previous sections. LAMA-TREx is a set of fact-related cloze questions. Since we are interested in entity prediction models, we restrict our LAMA investigations to TREx, which has answers linked to Wikidata. TriviaQA (open) contains questions scraped from quiz-league websites [68]. We use the open splits following [78]. FreebaseQA is an Open-domain QA dataset derived from TriviaQA and other trivia resources (See [66] for full details). Every answer can be resolved to at least one Freebase entity and each question contains at least one entity.

For WebQuestionsSP, we mapped question entities and answer entities to their Wikidata ids. 87.9% of the questions are answerable by at least one answer entity that is mappable to Wikidata. For all questions in FreebaseQA there exists at least one relational path in Freebase between the question entity e_i and the answer e_{ans}. The path must be either a one-hop path, or a two-hop path passing through a mediator (CVT) node, and is verified by human raters. 72% of the question entities and 80% of the answer entities are mappable to Wikidata, and 91.7% of the questions are answerable by at least one answer entity that is mappable to Wikidata.

Furthermore, WebQuestionsSP (and similarly FreebaseQA discussed in §5.4) was constructed such that all questions are answerable using the FreeBase KB, which was last updated in 2016. Because our pretraining corpus is derived from larger and more recent versions of Wikipedia, we elected to use a KB constructed from Wikidata. Many entities in Freebase are unmappable to the more recent Wikidata KB which means that some questions are no longer answerable using the KB. Because of this, we created reduced versions of these datasets which are *Wikidata answerable*—i.e., containing only questions answerable by triples from our Wikidata-based KB. The model should learn to rely on the KB to answer the questions. We do the same for TriviaQA.[10]

		Full Dataset	Wikidata Answerable
FreebaseQA	Train	20358	12535
	Dev	3994	2464
	Test	3996	2440
WebQuestionsSP	Train	2798	1388
	Dev	300	153
	Test	1639	841

Table 5.1: **Dataset stats.** Number of examples in train, dev, and test splits for our three different experimental setups. Full are the original unaltered datasets. Wikidata Answerable keeps only examples where at least one question entity and answer entity are mappable to Wikidata and there is at least one fact between them in our set of facts.

[10]TriviaQA does not have linked entities in its questions so for those results we relax this restriction to include all examples where the answer resolves to a Wikidata entity.

5.3.2 Baselines

T5 [121] and BART [79] are large text-to-text transformers. Dense Passage Retrieval (DPR) [70] is a two stage retrieve and read model. Retrieval Augmented Generation (RAG) [81] and Fusion in Decoder (FID) [63] use DPR retrieval, followed by generative decoders based on BART and T5 respectively. FID is the current state-of-the-art on the open domain setting of TriviaQA. K-Adapter [167] and Bert-KNN [72] are recent BERT extensions that perform at or near state-of-the-art on the LAMA benchmark. Entities-as-Experts (EaE) [51] is discussed in §5.2.3. Our EaE models are trained using the same hyperparameters and optimization settings as FILM.

5.3.3 Open vs Closed Book models

Generally, open book models refer to 'retrieve and read' pipelines [17] which, given a query, 1) retrieve relevant passages from a corpus, 2) separately re-encode the passages conditioned on the question and then 3) produce an answer. Conversely, closed book models answer questions directly from their parameters without additional processing of source materials. We consider FILM and EaE closed-book models as they do not retrieve and re-encode any source text, and instead attend to parameterized query-independent memories.

5.3.4 Results in Convention Settings

LAMA TREx. In Table 5.2, we can see that FILM outperforms several recently proposed models on the LAMA TREx task. FILM outperforms the next best performing model, BERT-KNN by 5.5 points.

Model	Precision@1
K-Adapter †	29.1
BERT-Large †	33.9
BERT-KNN ‡	38.7
EaE	38.6
FILM	**44.2**

Table 5.2: **LAMA TREx** Precision@1. † copied from [167], ‡ copied from [72]

Question-Answering. In Table 5.3, we compare FILM to five close-book and three open-book QA models on WebQuestionsSP and TriviaQA. The columns denoted Full Dataset-Total show results for the standard evaluation. For WebQuestionsSP, despite using far fewer parameters (see Table 5.4), FILM outperforms all other models - including the top open-book model RAG. On TriviaQA, FILM outperforms all other closed-book models—though the open-book models are substantially more accurate on this task, likely because of the enormous size of the models and their access to all of Wikipedia, which contains all (or nearly all) of the answers in TriviaQA.

	Model	WebQuestionsSP				TriviaQA			
		Full Dataset		Wikidata Answer		Full Dataset		Wikidata Answer	
		Total	No Overlap	Total	No Overlap	Total	No Overlap	Total	No Overlap
Closed -book	**FILM**	**54.7**	**36.4**	**78.1**	**72.2**	**29.1**	**15.6**	**37.3**	**28.4**
	EaE	47.4	25.1	62.4	42.9	19.0	9.1	24.4	17.1
	T5-11B	49.7	31.8	61.0	48.5	–	–	–	–
	BART-Lg	30.4	5.6	36.7	8.3	26.7	0.8	30.6	1.0
	Dense NN	27.8	2.9	34.0	3.8	28.9	0.0	33.2	0.0
	tfidf NN	21.4	2.6	25.4	3.0	23.5	0.0	26.9	0.0
Open -book	RAG	**50.1**	**30.7**	**62.5**	**45.1**	56.8	29.2	64.9	45.2
	DPR	48.6	34.1	56.9	45.1	57.9	31.6	66.3	48.8
	FID	–	–	–	–	**67.6**	**42.8**	**76.5**	**64.5**

Table 5.3: **Open Domain QA Results**. Columns denoted *Full Dataset-Total* are conventional splits discussed in §5.3.4, *Wikidata Answer* are answerable using Wikidata (§5.3.5), and *No Overlap* removes train-test overlap (§5.3.6). Highest closed-book and open-book numbers are bolded. Other than FILM and EaE, all results are derived from the prediction files used in [82] including the nearest neighbor (NN) baselines.

Model	Base	Memory	Total
FILM	0.11	0.72	0.83
EaE	0.11	0.26	0.37
BERT-L	0.35	0	0.35
BART-L	0.39	0	0.39
T5-11B	11	0	11
DPR	0.11	16	16.11
RAG	0.39	16	16.39
FID	0.77	16	16.77

Table 5.4: **Model Parameters** Approximate billions of parameters for each model's base, memories and total. Excludes token embeddings.

5.3.5 Results on KB-Answerable Questions

As seen in Table 5.3 in the column Wikidata answer-Total, FILM does much better on Wikidata answerable questions on WebQuestionsSP. EmQL [143], the state-of-the-art dataset specific model, gets 75.5% accuracy on the full dataset. Not surprisingly, this is because EmQL operates over the Freebase knowledge base, giving it full upperbound recall. However, when we restrict to Wikidata answerable questions, thus giving both EmQL and FILM potential for full recall, FILM outperforms EmQL by 3.5 points and the next best model (RAG) by over 15 points.

5.3.6 Train-Test Overlap

We are interested in the ability of models to use external knowledge to answer questions, rather than learning to recognize paraphrases of semantically identical questions. Unfortunately, analysis showed that many of the test answers also appear as answers to some training-set question: this is the case for 57.5% of the answers in WebQuestionsSP and 75.0% for FreebaseQA. This raises the possibility that some of the performance can be attributed to simply memorizing specific question/answer pairs, perhaps in addition to recognizing paraphrases of the question from its pretraining data.

Overlap in fine-tuning train/test splits was concurrently observed by [82], who created human verified filtered splits for TriviaQA and WebQuestions. We evaluate our models on those splits and report results in Table 5.3 in the "No Overlap" columns. We see that the gap between FILMand the next best performing model RAG increasses from 4.6 to 5.7 points on WebQuestionSP. On TriviaQA, FILMis still able to answer many questions correctly after overlap is removed. In contrast, the majority of closed book models such as BART get less than 1% of answers correct.

5.4 Modifying the Knowledge Base

Because our model defines facts symbolically, it can in principle reason over new facts injected into its memory, *without retraining any parameters of the model*. Since existing datasets do not directly test this capability, we elected to construct variants of FreebaseQA and WebQuestionsSP where we could simulate asking questions that are answerable only from newly injected KB facts.

The approach we used was to (1) identify pairs of entities that occur in both a question and answer of some test example; (2) filter out such pairs from the KB as well as all pre-training and fine-tuning data; and (3) test the system trained on this filtered data, and then manually updated by injecting facts about those entity pairs. This filtering procedure is reminiscent of that used by [82], but also addresses pretraining / test-set overlap.

5.4.1 Injecting New Facts to Update Memory

We evaluate EaE and FILM given full knowledge (the original setting); given filtered knowledge; and given filtered knowledge followed by injecting test-question-related facts into the KB. The gap between the filtered knowledge setting and injected knowledge setting will indicate how well the model incorporates newly introduced facts.

In more detail, we first perform a similar procedure to [82] and discard questions in the fine-tuning training data that contain answers which overlap with answers to questions in the dev and test data. We end up with 9144/2308/3996 data (train/dev/test) in FreebaseQA and 1348/151/1639 data in WebQuestionsSP. Next, to ensure that the model will be unable to memorize paraphrases of question-answer pairs that it observed in the pre-training text, we remove all overlap between the pretraining data and fine-tuning test data: specifically, for every question-answer entity pair in our fine-tuning dataset (from any split), we filter every example from our Wikipedia pretraining corpus in which that pair of entities co-occur. Additionally, we filter every fact from our fact memory containing any of these entity pairs.

	FreebaseQA			WebQuestionsSP		
	Full	Filter	Inject	Full	Filter	Inject
EaE	45.8	28.6	-	30.9	29.4	-
FILM	56.5	38.7	48.0	40.7	32.3	39.2

Table 5.5: **Injecting New Facts**. In the Filter setting, the models have access to no direct knowledge about question answer entity pairs from either the pretraining corpus or KB. In the Inject setting, the pretraining corpus and training KB are still Filtered, but at inference time, new facts are injected into the models memory allowing it to recover most of the drop from the Full setting. In the Full setting the model is exposed to full knowledge. In all cases, we remove the overlap between the finetune train and eval sets.

In these sections we compare against EaE for two reasons: 1) we are specifically looking at closed-book open domain entity based QA and EaE is shown to be at or near state-of-the-art for that task [51], 2) most importantly, we want to be able to precisely control for memorization in the training corpus and therefore did not consider existing unconstrained pre-trained models like T5 [121]. For reference, the previous state-of-the-art FOFE [66] on FreebaseQA had a score of 37.0% using the original train-test split, while FILM is at 63.3%.

The results are shown in Table 5.5. In the "Full" column, we pretrain and finetune the FILM model with the full knowledge base and corpus. In the "Filter" setting, facts about the finetuning data are hidden from the model at both pretraining and finetuning time. In this case, the model must fall back to the language model to predict the answer, and as shown in Table 5.5, the accuracies of FILM and EaE are similar. In the "Inject Facts" setting, Facts are hidden at pretraining time, but are injected at test time. The results show that FILM can effectively use the newly injected facts to make prediction, obtaining an absolute improvement of 9.3% compared to the "Filter" setting. EaE does not have a natural mechanism for integrating this new information[11].

5.4.2 Updating Stale Memories

One of the main motivations for our model is to provide knowledge representations that can be incrementally updated as the world changes, avoiding stale data. To probe this ability, we simulate an extreme version of this scenario where all answers to QA pairs in the FreebaseQA test set are replaced with plausible alternatives. For each QA pair, we replace the original answer entity e_{original} with another entity, e_{new}, from our vocabulary that has: 1) been used as an object in at least one of the same relation types in which e_{original} was used as an object, and 2) shares at least three Wikipedia categories with e_{original}. We use the same pretrained models from our earlier experiments and fine-tune on the filtered FreebaseQA train set for 10,000 steps.

We look at two methods for modifying the fact memory. *Basic Filter* deletes every modified

[11]There are various heuristics one could apply for finetuning a standard language model on this type of data by applying one or a small number of gradient steps on textualized facts. We leave this exploration for future research.

fact e_{question}, r, e_{original} and replaces it with a new fact e_{question}, r, e_{new}. The *Strict Filter* setting more aggressively removes information that may conflict with the newly added fact by additionally removing all facts that contain e_{question} or e_{original}. This is relevant when a question contains multiple entities, or the linking relation is one-to-many, leading to multitple plausible answers. Together these two settings define rough bounds on the model's ability to perform this task. Overall, FILM is able to utilize the modified KB to make the correct prediction for **54.5%** of questions in the *Basic Filter* setting and **70.3%** in the *Strict Filter* setting.

5.5 Related Work

Knowledge bases (KBs) have been a core component of AI since the beginning of the field [111, 110]. Widely available public KBs have been invaluable in research and industry [14, 8] and many companies have created massive KBs as the backbones of their most important products [55, 47].

While traditional KBs were purely symbolic, recent advances in large language models trained through self supervision [115, 42, 121, 16] have been shown to encode an impressive amount of factual information. This has led to research on the extent to which a neural language model can serve as a KB [132, 117], and how to best evaluate this factual knowledge [119].

While large LMs appear to absorb KB-like information as a preproduct of pretraining, there has also been many prior approaches proposed that explicitly embed symbolic knowledge representations into neural embedding space. Various neural-symbolic methods have attempted to unify these two extremes [118, 35, 13] including many cognitive architectures which used hybrid symbolic and subsymbolic systems [76], and more recently, compositional query languages for embedding KBs that are similar to symbolic KB query languages [28, 128, 27]. One system especially related to our proposal is EmQL [143], which includes a construct quite similar to the "fact memory" used in our Facts-As-Experts model. Unlike this work, however, EmQL did not embed its fact memory into a language model, which can be finetuned for many NLP tasks: instead EmQL must be used with task-specific query templates and integrated into some task-specific architecture.

More recently, the past decade has seen huge amount of work on knowledge base embeddings [15, 89, 156, 38] which enable generalization through similarities between learned embeddings. This idea has also been extended with works looking at ways of incorporating raw text and symbolic KGs into a shared embedding space [131, 160, 161], to be jointly reasoned over [149, 144], or to treat text as a replacement for a knowledge base [45].

Large external memories have been incorporated into different types of networks operating over latent parameters [171, 99], entities [62, 51], facts [3, 94], commonsense relations [168, 18, 97], and text passages [59, 81]. Our work directly extends one of these models, the Entities-as-Experts (EaE) model [51], one of several models that inject knowledge of entities by constructing a memory containing embedded entity representations. Unlike prior models, EaE learns entity representations end-to-end, rather than using representations from a separately-trained KB embedding system [94]. Our work extends EaE by introducing a symbolic memory of triples which is constructed from these learned entity representations, and as in EaE, the entity representations are learned end-to-end.

5.6 Discussion

In this chapter, we presented a method for interfacing a neural language model with an interpretable symbolically bound memory. We used that interface to change the output of the language model by modifying only the non-parametric memories and without any additional training. We demonstrated the effectiveness of this method by performing comparably or better than a high performing language model on factoid question answering while integrating new facts unseen in pretraining data. We even showed that we can modify facts, such that they contradict the initial pre training text, and our model is still largely able to answer these questions correctly.

While we believe this approach makes progress towards interpretable and modifiable neural language models, there is still more work to be done. First, the coverage of knowledge in symbolic KBs is limited by their incompleteness, a common problem faced by most factual KBs. Second, the model still sometimes makes mistakes when the facts contradict the latent knowledge stored in parameters of the model. The ‘mixing’ mechanism which combines these two signals is an important component and still requires further developments. Additionally, while we can add new facts or replace/overwrite existing facts, there is still not a satisfying mechanism to simply delete without replacement. Future research could explore more interesting ways of representing and mixing signals from the context, positive facts, and deleted facts. It would also be important and interesting to see how these methods can be adapted to fully generative language model tasks, and how these same fact mixing mechanisms can be incorporated into guided language generation. Lastly, our methodology currently performs a single ’hop’ of reasoning on the knowledge graph. In future work we intend to integrate multiple fact layers within the transformer to allow the model to perform multi-hop reasoning.

In the next chapter, we will resolve two of the limitations mentioned above: (1) replacing symbolic KBs with virtual KBs (constructed from OPQL) to improve the coverage of knowledge, and (2) improving models on multi-hop questions.

Chapter 6

Language Models with OPQL (OPQL-LM)

6.1 Overview

In the previous chapter, we proposed FILM which injected an external fact memory into an LM, increasing its performance on open-domain QA. However, this approach relies on a symbolic KB, suffering from many of the shortcomings discussed in previous sections such as limited coverage and applicability. In this section, we address these issues by replacing the KB-based fact memory of FILM with the virtual KB constructed by OPQL. Here, we re-used the OPQL method to construct virtual KB. For simplicity, we will skip the discussion of OPQL in this chapter and focus on its integration with LMs. Please refer to Chapter 4 for more information on OPQL.

6.2 Method

6.2.1 Background

A memory injected language model, e.g. FILM (§5), learns to retrieve relevant entries from an external memory and mix the retrieved information into the language model to make its final predictions (see Figure 6.1). OPQL-LM follows the same implementation as FILM but utilizes the pre-computed OPQL memory as the external memory. Since the relation embedding in the OPQL memory is fixed when training OPQL-LM, retrieving from the OPQL memory is harder than the fact-based memory in FILM. To address this, we make several key changes to the original FILM architecture. First, the query vector is constructed as a function of the topic mention embedding and relation embedding. Second, we modify the learned a mixing weight between the memory retrieval results and language

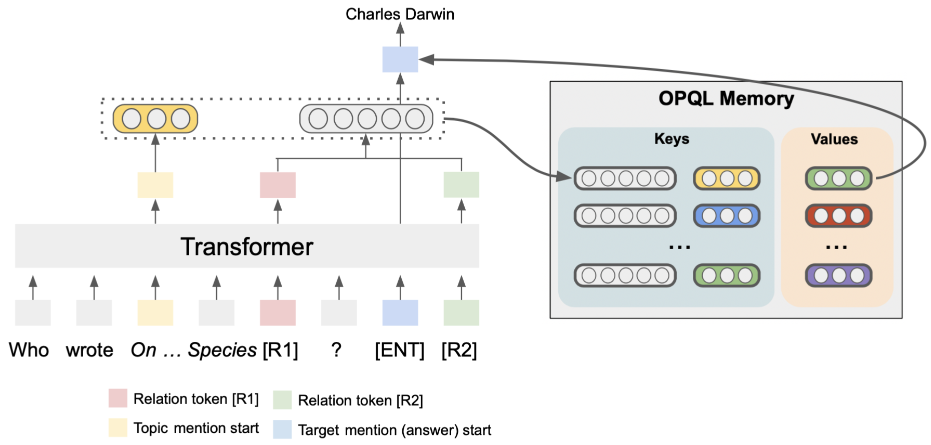


Figure 6.1: **OPQL-LM model architecture**. A query vector $\mathbf{q}_{X,Y}$ is constructed from the contextual embedding of the topic mention $\mathbf{m}_{e_1}$ and the relation embedding $\mathbf{r}_{X,Y}$, and retrieves the top few entries from the OPQL memory. The retrieval results are aggregated into a single vector $\mathbf{e}_Y$, and mixed with the contextual embedding of the masked mention (answer) $\mathbf{m}_{e_2}$ to make the final prediction.

model predictions. Third, we extend OPQL-LM to answer multi-hop questions. We will elaborate these changes in the rest of this section.

6.2.2 Input

Similar to the relational following task with OPQL discussed in Chapter 4, the special tokens [R1] and [R2] are inserted directly after the mentions of topic entity e_1 and target entity (answer) e_2. But the mention of topic entity e_1 will not be masked with [ENT], e.g. "Who published the book *On the Origin of Species* [R1] in 1859? [ENT] [R2]". We denote the mention of topic entity as m_{e_1}. The mention m_{e_1} is used to extract contextual mention embedding $\mathbf{m}_{e_1}$ for the topic entity e_1, which will be used for entity linking, as well as constructing the query to retrieve from the OPQL memory. The training objective of OPQL-LM is to predict the masked target entity (answer) e_2.

6.2.3 Query Embedding

The query $\mathbf{q}_{X,Y}$ is computed compositinoally by concatenating the contextual mention embedding $\mathbf{m}_{e_1}$ of the topic mention m_{e_1} and the relation embedding $\mathbf{r}_{X,Y}$. Ideally, $\mathbf{m}_{e_1}$ should be close to the embedding of the oracle topic entity $\mathbf{e}_1$, which is supervised with entity linking loss (Eq. 4.3). The relation embedding $\mathbf{r}_{X,Y}$ comes from the relation encoder (Eq. 4.1) that operates on special tokens

[R1] and [R2].[1]

$$\mathbf{q}_{X,Y} = \mathbf{W}_q^T[\mathbf{m}_{e_1}; \mathbf{W}_t^T \mathbf{r}_{X,Y}]$$

6.2.4 Mixing with LM

Let $\mathbf{e}_Y$ be the retrieved embedding returned from the OPQL memory and $\mathbf{m}_{e_2}$ be contextual embedding of the masked mention [ENT] predicted from the language model. $\mathbf{e}_Y$ and $\mathbf{m}_{e_2}$ are mixed with a mixing factor λ. λ decides whether retrieved embedding should be included to make final predictions. λ should be large if there is a relevant entry with pair (e_1, e_2) in the memory, so retrieving this pair can help predict the masked entity e_2. Different from FILM, which introduced a `null` fact whose embedding $\mathbf{k}_{\texttt{null}}$ is a learned variable, OPQL-LM introduces a learnable `null` relation in the OPQL memory and constructed a `null` entry with key embedding $\mathbf{k}_{\texttt{null}} = \mathbf{W}_k^T[\mathbf{m}_{e_1}; \mathbf{r}_{\texttt{null}}]$ and value embedding $\mathbf{v}_{\texttt{null}} = \overrightarrow{\mathbf{0}}$. The query is encouraged to retrieve the `null` entry if no relevant pair of (e_1, e_2) exists in the OPQL memory. We re-use the memory key projection matrix $\mathbf{W}_k$ to compute the key embedding $\mathbf{k}_{\texttt{null}}$.

$$\mathbf{m}'_{e_2} = \mathbf{m}_{e_2} + (1-\lambda)\cdot \mathbf{e}_Y, \quad \lambda = \text{softmax}(\mathbf{q}_{X,Y}^T\ \mathbf{k}_{\texttt{null}}) \tag{6.1}$$

6.2.5 Multi-hop OPQL-LM

Some open-domain questions require multi-hop reasoning to find the answers, e.g. "Where is the author of *On the Origin of Species* educated?". To answer such questions, the model should first find the answer of the first-hop of the question, and use the it as topic entity to answer the second-hop of the questions. OPQL-LM can naturally solve this problem by repeating the retrieval and mixing steps.

Let ${\mathbf{m}'_{e_2}}^{(t)}$ from Eq. 6.1 be the memory-injected contextual embedding that was used to predict the answer of the t'th hop. In the second hop, ${\mathbf{m}'_{e_2}}^{(t)}$ becomes the embedding of the topic entities and used to compute query $\mathbf{q}_{X,Y}^{(t+1)}$ for the $(t+1)$'th hop. Again, we keep the relation embedding $\mathbf{r}_{X,Y}$ unchanged, but learn another projection matrix $\mathbf{W}_t^{(t+1)}$. ${\mathbf{m}'_{e_2}}^{(t+1)}$ will be computed accordingly with updated $\mathbf{m}_{e_2}^{(t+1)}$, $\lambda^{(t+1)}$ and $\mathbf{e}_Y^{(t+1)}$ at the $(t+1)$'th hop.

$$\mathbf{q}_{X,Y}^{(t+1)} = \mathbf{W}_q^T[{\mathbf{m}'_{e_2}}^{(t)}; \mathbf{W}_t^{(t+1)T}\mathbf{r}_{X,Y}]$$

6.2.6 OPQL-LM for Conjunction Questions

A conjunction questions, e.g. "Which *English* author publish his book *On the Origin of Species*?", contains more than one topic entity *English* and *On the Origin of Species*. Both topic entities can

[1]Since the mention of entity e_1 is not masked from the input, the relation embedding $\mathbf{r}_{X,Y}$ may contain some leaked information from the topic entity e_1. One could encode the relation embedding $\mathbf{r}_{X,Y}$ separately from an input where both e_1 and e_2 are masked. We do not take this solution as it doubles the computation cost of the expensive Transformer layers.

potentially help to predict the answer, e.g. with pairs (*English*, *Charles Darwin*) that describes his nationality and (*On the Origin of Species*, *Charles Darwin*) that describes his publications. We convert the input by appending the masked answer to the end of the question and inserting the special tokens [R1] and [R2] accordingly, e.g. "Which *English* [R1] author publish his book *On the Origin of Species* [R1]? [ENT] [R2]".

A query vector $\mathbf{q}_{X,Y,e_i}$ is constructed for each topic entity e_i, with the retrieval results $\mathbf{e}_{Y,e_i}$ returned from the memory. All retrieved embeddings are mixed with the contextual embedding $\mathbf{m}_{e_2}$ to predict the final answer

$$\mathbf{m}'_{e_2} = \mathbf{m}_{e_2} + \sum_{e_i} \lambda_i \cdot \mathbf{e}_{Y,e_i}$$

where λ_i is the mixing weight that is determined by each query independently.

6.2.7 Pretraining OPQL-LM

We propose a second stage pretraining for OPQL-LM that learns to retrieve from the OPQL memory and compute the mixing weight λ. Relation embeddings $\mathbf{r}_{e_1,e_2}$ are precomputed and fixed at the second stage pretraining. We continue training Transformer layers and the entity embedding table $\mathbf{E}$ for entity linking.

We use Wikipedia passages as our pretraining data. Given an input C that contains M mentions, we randomly select one mention as our target entity e_2 and mask it with [ENT]. All other mentions are considered topic entities $\{e_1\}$. Special tokens [R1] and [R2] are inserted after mentions of topic and target entities. Mentions of the topic entities will not be masked. Each entity pair (e_1, e_2) will be treated independently. In an example with three mentions, "[ENT] [R2], an *English* [R1] naturalist, published his book *On the Origin of Species* [R1] in 1859.", the mention *Charles Darwin* is selected as the target entity e_2. Retrieval and mixing steps are performed on both topic entities *English* and *On the Origin of Species* independently to predict the masked entity *Charles Darwin*. OPQL-LM is trained on 85.6M text pieces with a length of 128 tokens.

6.3 Experiments

Next, we experiment with OPQL-LM in another realistic scenario – open-domain QA. In open-domain QA, questions are more diverse natural language and do not contain oracle-linked entities. In our experiments, we make a weaker assumption that mention detection has been run on the input providing boundaries of entity mentions to the model. OPQL-LM can effectively solve open-domain QA by retrieving relevant entries from the memory and return the corresponding values as answers and the retrieval results from the memory can be mixed with language model predictions to further improve the performance.

6.3.1 Dataset

WebQuestionSP (WebQSP) has been used in previous sections.
ComplexWebQuestions (ComplexWebQ) [153] extends WebQuestionsSP to multi-hop questions. The ComplexWebQuestions dataset contains 34,689 complex questions, including 45% composition questions, 45% conjunction questions, and 10% others. Similar to the WebQuestionsSP dataset, we convert Freebase MIDs to Wikidata QIDs. 94.2% of the questions are answerable.

6.3.2 Baselines

We compare OPQL with several strong open-domain QA baselines. GRAFT-Net and PullNet [149, 144] are Graph-CNN based models that are introduced in the previous experiments. BART [79] and T5 [121] are large pretrained text-to-text Transformer models[2]. EaE [51] is trained to predict masked entities, but without an external (virtual) KB memory. EaE does not have an external reasoning module, so it requires the reasoning process performed all from the Language Model. DPR [70] is a retrieve-and-read approach that pairs a dense retriever in the embedding space with a BERT based reader. To extend DPR to handle multi-hop questions, we concatenate answers from the previous hops to the question as the query for the next hop. This is referred as DPR-cascade.[3] We also include results of FILM though its external memory is built from KB.

6.3.3 Results

OPQL-LM outperforms the baseline models on both datasets (Table 6.1). We also experimented with an ablated model OPQL-follow that only performs multi-hop relational following on questions,[4] i.e. retrieved embeddings $\mathbf{e}_Y$ is directly used to make predictions. In WebQuestionsSP, the accuracy of retrieving the relevant pair is 85.4% and that leads to 46.6% accuracy of the WebQuestionsSP dataset. However, in ComplexWebQuestions, the coverage of OPQL memory to answer both hops of the questions is only 22.1% for compositional questions.[5] So the accuracy of OPQL-follow is bounded. Mixing OPQL with LM can further improve the performance of OPQL-follow by 5.3% on WebQuestionsSP and 22.4% on ComplexWebQuestions.

The entity-dependent null embedding $\mathbf{k}_{\text{null}}$ is crucial for OPQL-LM. In an ablation study that the null embedding does not depend on the topic entity e_1, i.e. $\mathbf{k}_{\text{null}}$ is a learned variable shared by all entities, the performance on WebQuestions drops from 51.9 to 46.3 with the retrieval accuracy dropping from 85.4 to 69.5. The accuracy on Complex WebQuestions drops from 40.7 to 19.3.

[2]WebQuestionsSP is finetuned on T5-11B. Due to hardware constraint, we finetune T5-3B for ComplexWebQuestions.

[3]We run DPR and DPR-cascade with the pretrained checkpoint on Natural Questions [75]. Finetuned DPR reader on ComplexWebQuestions only gets 18.2% Hits@1.

[4]We do not assume oracle entity linking here. The model use the contextual embedding of the topic entity to construct the query.

[5]This number is computed from the intermediate answers provided in the dataset. We do not use the intermediate answers in training.

Model	WebQSP	ComplexWebQ (dev)
GRAFT-NET	25.3	10.6
PullNet	24.8	13.1
BART-Large	30.4	-
EaE	47.4	31.3
DPR	48.6	24.6
DPR-cascade	-	25.1
T5	49.7	38.7
OPQL-follow	46.6	18.5
OPQL-LM	**51.9**	**40.7**
+ webqsp pairs	*53.7*	*41.1*
FILM	54.7	-
SoA (KB)	69.0 (F1)	47.2

Table 6.1: Hits@1 performance on open-domain QA datasets. OPQL+webqsp pairs injects additional data specific memories. The state-of-the-art model on WebQSP (NSM [86]) and ComplexWebQ (PullNet-KB [144]) both use Freebase to answer the questions. FILM has an external memory with KB triples.

6.3.4 Injecting Entries into the OPQL Memory

We show that OPQL can efficiently injecting new pairs to the memory to improve the coverage of knowledge at finetuning time, without having to retrain the relation encoder or LM. In the WebQuestionsSP experiment above, the pretrained OPQL memory that contains 1.6M popular entity pairs (800k pairs plus their inverse) only covers 54.6% of the questions, and each question entity was only included in 8.4 pairs in the memory of pre-selected entity pairs. To improve the coverage, we added 100 pairs per question entity with the highest PMI to the memory. The updated memory contains 1.8M pairs and the coverage increases to 82.9%.

The result with the updated memory is presented in Table 6.1. The retrieval accuracy on questions that has relevant pairs in the memory drops from 85.4% to 62.2% as a result of adding 10 times more pairs for each question entities, but the overall Hits@1 accuracy of the model improves from 51.9% to 53.7%. We also finetune OPQL-LM on ComplexWebQuestions dataset. The improvement is less significant since retrieval is harder on complex questions.

6.4 Related Work

Previous work has shown that injecting an external memory constructed from KB into a LM can help training LMs [116], improve generation tasks, [94], and enable reasoning over an updated memory [159]. Additionally, our model's memory scales to millions of entries, whereas most prior systems that use KB triples have been with only a few hundreds of triples in the model at any point, necessitating a separate heuristic process to retrieve candidate KB triples [3, 62, 168, 18, 97, 95].

6.5 Discussion

In this chapter, we introduced methods to inject the previously discussed query languages, OPQL, to pretrained LMs to improve their capability in answering complex questions. Experiments showed that virtual KB-augmented language models, OPQL-LM, perform well on question answering tasks in the closed-book manner.

Most of tasks studied in this chapter, however, assumes questions are complete, i.e. information needed to find answers is present in questions. Therefore, questions can be mostly modeled as relational following, i.e. $X.\text{follow}(R)$, even though entities X and relations R in FILM and OPQL-LM do not need to be explicitly extracted, in contrast to the plain query languages.

In the next chapter, we will study another group of question answering tasks, named conditional question answering, where questions have one or more answers and the answers are correct under certain conditions. This is usually because questions do not provide the *complete* information to make a definite answer. Answering questions with incomplete information is a challenging topic but has very limited research. Different from questions that have unique and definite answers, conditional QA is more challenging because it requires models to: (1) consider all possibility of the answers, (2) check whether the answers satisfy all required constraints given the information provided in the question, and (3) identify the unsatisfied constraints if answers need more clarification. We will discuss these challenges in detail in the rest of this thesis.

Part III

Answering Questions with Conditions

Chapter 7

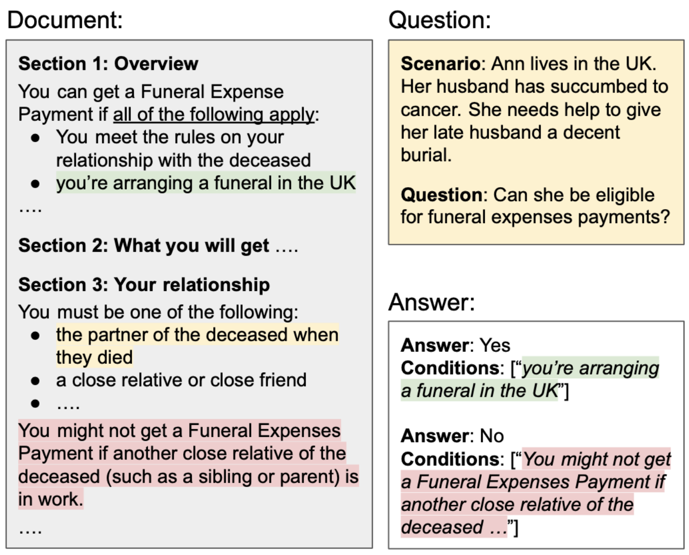
ConditionalQA Dataset

7.1 Overview

In previous chapters, we discussed several methods to solve reasoning tasks for Question Answering. For example, we studied neural-symbolic methods, NQL, EmQL, and OPQL, for reasoning tasks, e.g. single-hop and multi-hop relational following, intersection and union, and constraints, over symbolic and virtual knowledge bases. We further discussed incorporating these reasoning methods, in particular for single-hop and multi-hop relational following tasks, into language models (LMs) to improve LMs' ability in answering complex questions.

We will discuss QA that requires a different type of reasoning in this chapter – abductive reasoning. In abductive reasoning, questions have answers that are only correct under certain conditions. For example, in Figure 7.1, the document discusses "Funeral Expense Payment" and a question asks an applicant's eligibility. This question cannot be deterministically answered: the answer is "yes" only if "you're arranging a funeral in the UK", while the answer is "no", if "... another close relative of the deceased is in work" is true. We call answers that are different under different conditions *conditional answers*.

Conditional answers are commonly seen when questions do not contain all information needed to get a deterministic answer, for example, when the context is so complex (in Figure 7.1) that asking a *complete* question with a deterministic answer is impractical. Conditional answers are also seen when questions are asked with implicit assumptions that are not clearly specified in the question. For example, the question "which is the first approved Covid vaccine" has different answers in different geographical locations. A practical way to answer incomplete questions is to find probable answers to the question given the provided information – and if some answers are only true under certain conditions, the conditions should be output as well. Answering such questions requires QA systems to understand all information in the context and perform extensive reasoning to identify the answers and conditions.



Document:

Section 1: Overview
You can get a Funeral Expense Payment if all of the following apply:
- You meet the rules on your relationship with the deceased
- you're arranging a funeral in the UK

....

Section 2: What you will get ....

Section 3: Your relationship
You must be one of the following:
- the partner of the deceased when they died
- a close relative or close friend
- ....

You might not get a Funeral Expenses Payment if another close relative of the deceased (such as a sibling or parent) is in work.
....

Question:

Scenario: Ann lives in the UK. Her husband has succumbed to cancer. She needs help to give her late husband a decent burial.

Question: Can she be eligible for funeral expenses payments?

Answer:

Answer: Yes
Conditions: ["*you're arranging a funeral in the UK*"]

Answer: No
Conditions: ["*You might not get a Funeral Expenses Payment if another close relative of the deceased ...*"]

Figure 7.1: An example of question and document in ConditionalQA dataset. The left side is a snapshot of the document discussing the eligibility of the benefit "Funeral Expense Payment". The text span "Her husband" satisfies the requirement on the "relationship with the decease" (in yellow). Text pieces in green and red are requirements that must be satisfied and thus selected as conditions for the "Yes" and "No" answers.

Very limited work has been previously done in this challenging reasoning task, i.e. abductive reasoning, for answering questions with conditions. To benchmark this challenging task, we proposed a novel dataset, ConditionalQA. In particular, we test models' ability in identifying missing information from questions as conditions. We discuss the collection process of the dataset and formally define the ConditionalQA task. Experiment results with the previous state-of-the-art QA models are reported to reflect the challenges in answering questions with conditional answers.

7.2 The Task

In our task, the model is provided with a long document that describes a public policy, a question about this document, and a user scenario. The model is asked to read the document and find all answers and their conditions if any.

7.2.1 Corpus

Documents in ConditionalQA describe public policies in the UK, e.g. "Apply for Visitor Visa" or "Punishment of Driving Violations". Each document covers a unique topic and the contents are grouped into sections and subsections. Contents in the same section are closely related but may also be referred in other sections. We create ConditionalQA in this domain because these documents are rather complex with internal logic, yet annotators are familiar with the content so they can ask natural yet challenging questions, compared to formal legal or financial documents with more sophisticated terms and language.

7.2.2 Input

The input to a reading comprehension model consists of a document, a question, and a user scenario:

- A *document* describes a public policy in the UK. Content of a document is coherent and hierarchical, structured into sections and subsections. Documents are crawled from the website and processed by serializing the DOM trees of the web pages into lists of HTML elements with tags, such as <h1>, <p>, <li>, and <tr>. Please see more information in §7.3.1.
- A *question* asks about a specific aspect of the document, such as eligibility or other aspects with "how", "when", "what", "who", "where", etc. Questions are relevant to the content in the document, even though they may be "not answerable".
- A *user scenario* provides background information for the question. Some information will be used to restrict the answers that can be possibly correct. Not all information in the user scenario is relevant because they are written by crowd source workers without seeing the full document or knowing the answers. Information in the scenario is also likely to be incomplete. This setup simulates the real information seeking process of having both irrelevant and incomplete information.

7.2.3 Output

A reading comprehension model is asked to predict the answers and the list of conditions if there is any.

- An *answer* to the question has three different types: (1) "yes" or "no" for questions such as "Can I get this benefit?"; (2) an extracted text span for questions asking "how", "when", "what", etc.; (3) "not answerable" if an answer does not exist in the document. Since the information to get a definite answer is sometimes incomplete, besides predicting the answers, the model is asked to identify their conditions.

- A *condition* contains information that must be satisfied in order to make the answer correct but is not mentioned in the user scenario. In ConditionalQA, we restrict a condition to be one of the HTML elements in the document instead of the exact extracted text span.[1] Selected conditions are then evaluated as a retrieval task with F1 at the element level, i.e. the model should retrieve all HTML elements with unsatisfied information to get a perfect F1 score. If no condition is required, the model must return an empty list. Please see §7.2.4 for more details on evaluation.

[1]We argue that selecting HTML elements as conditions is already very challenging (see experimental results in §5.3) and leave extracting the exact text spans as future work.

7.2.4 Evaluation

We evaluate performance of models on the ConditionalQA dataset as a reading comprehension (RC) task. Answers are measured with exact match (EM) and F1. Some questions have multiple answers. The model should correctly predict all possible answers to get the full score. Since the order of answers does not matter, to compute the metrics, we compare all possible permutations of the predicted answers to the list of correct answers. We take the best result among all permutations as the result for this example. Let $\{\hat{a}_1, \ldots, \hat{a}_m\}$ be the list of predicted answer and $\{a_1, \ldots, a_n\}$ the reference answers. The EM of the predicted answers is

$$\text{EM} = \max_{\{\tilde{a}_1,\ldots,\tilde{a}_m\}} \frac{1}{n} \sum_{i=1}^{\min(m,n)} s_{em}(\tilde{a}_i, a_i) \cdot \gamma_{m,n} \tag{7.1}$$

$$\gamma_{m,n} = \begin{cases} e^{1-m/n} & \text{if } m > n \\ 1 & \text{if } m \leq n \end{cases}$$

where $\{\tilde{a}_1, \ldots, \tilde{a}_m\}$ is a permutation of the predicted answers $\{\hat{a}_1, \ldots, \hat{a}_m\}$ and $s_{em}(\cdot, \cdot)$ is the scoring function that measures EM between two text spans. $\gamma_{m,n}$ is a penalty term that is smaller than 1 if more answers than the reference answers are predicted, i.e. $m > n$. We compute token-level F1 in the similar way using the scoring function $s_{f1}(\cdot, \cdot)$ on the extracted answer spans. For not answerable questions, EM and F1 are 1.0 if and only if no answer is predicted.

We additionally measure the performance of answers with conditions. We adopt the same permutation strategy as above, except that the scoring function will also take into account the accuracy of predicted conditions. Let $\hat{C}_i$ be the set of predicted conditions for the predicted answer $\hat{a}_i$ and C_i be the oracle conditions for the answer a_i. The new scoring function for the predicted answer with conditions is

$$s_{em+c}(\tilde{a}_i, \tilde{C}_i, a_i, C_i) = s_{em}(\tilde{a}_i, a_i) \cdot \text{F1}(\hat{C}_i, C_i)$$

where $\text{F1}(\cdot, \cdot)$ measures the accuracy of the set of predicted conditions at HTML element level. Recall that conditions are restricted to select from HTML elements in the document. $\text{F1}(\hat{C}_i, C_i)$ equals to 1 if and only if all required conditions are selected. This is different from $s_{f1}(\cdot, \cdot)$ that measures token level F1 of the extracted answers. If the answer does not require any conditions, the model should predict an empty set. We simply replace the scoring function $s_{em}(\cdot, \cdot)$ in Eq. 7.1 with $s_{em+c}(\cdot, \cdot)$ to compute EM with conditions.

7.3 Data Collection

The ConditionalQA dataset is built on documents from the UK government website[2]. Documents in this corpus discuss public policies in the UK and were first used in the ShARC dataset [135]. It is particularly interesting for constructing the ConditionalQA dataset because it contains complex contents with complex internal logic such as conjunction, disjunction, and exception (see the example in Figure 7.1). Questions in ConditionalQA are asked by human annotators. Each example contains a question, a scenario when the question is asked, and a document that discusses the policy that the question asks about.

7.3.1 Documents

Documents are originally presented on the UK government website in HTML format. We crawled the pages from the website and processed it to only keep crucial tags, that include:

- Headings <h1, h2, h3, h4>: We keep headings at different levels. This can be used to identify the hierarchical structure in the documents.
- Text <p>: This tag is used for paragraphs for general contents. We replace descriptive tags, e.g. <strong>, with the plain tag <p> for simplicity.
- List <li>: We keep the tags for list items, but drop their parent tags <ul> or <ol> . We observe that very few ordered lists (<ol>) have been used in the dataset, so we will not distinguish them.
- Table <tr>: Again, we drop their parent tags <table> to simplify the document format. We further remove the <td> and <th> tags from cells and concatenate cells in the same row with the separation of " — ".

A processed document contains a list of strings that starts with a tag, follows with its content, and ends with the tag, e.g. ["<h1> Overview </h1>", "<p> You can apply for ... </p>", . . .].

We drop some common sections that do not contain any crucial information, e.g. "How to Apply", to make sure that questions are specific to the topic of the documents. We further require that a document should contain at least 3 sections. We end up with 652 documents as our corpus. The max length of the documents is 9230 words (16154 sub-words in T5 [123]).

7.3.2 Questions

We collect questions from crowd source workers on Amazon Mechanical Turk. To encourage workers asking questions to not be restricted to a specific piece of text, we hide the full document but instead provide a snippet of the document to the workers. A snippet includes a table of contents that contains section and subsection titles (from <h1> and <h2> tags), and the very first subsection

[2] `https://www.gov.uk/parental-leave`

in the document, which usually provides a high level overview of the topic. The snippet lets workers get familiar with the topic of this document so they can ask relevant questions. We observe that restricting the geographic location of workers to the UK can significantly improve the quality of questions because local residents are more familiar with their policies.

We ask the workers to perform three sub-tasks when coming up with the questions. First, we ask the workers to provide three attributes that can identify the group of people who may benefit from or be regulated by the policy discussed in the document. Second, they are asked to come up with a *scenario* when the group will want to read this document and a *question* about what they would like to know. Third, workers are asked to mark which attributes have been mentioned in their question and scenario. When assessing the annotation quality, we find that asking workers to provide attributes makes the questions and scenarios much more specific, significantly improving the quality of the dataset.

We assign 3 workers to documents with four or more sections and 2 workers to documents with three sections. Each worker is asked to give two questions and the two questions have to be diverse. We collect 3617 questions in this stage.

7.3.3 Find Answers

We hire another group of workers to work on the answer portion of this task. Finding answers is very challenging to crowd source workers because it requires the workers to read the full document carefully to understand every piece of information in the document. We provide one-on-one training for the workers to teach them how to select supporting evidences, answers, and conditions.

Workers are asked to perform three sub-tasks. The first step is to select supporting *evidences* from the document. Supporting evidences are HTML elements that are closely related to the questions, including elements that have content that directly justify the answers and the ones that will be selected as conditions in the next step. In the second step, workers are asked to type answers and select associated conditions. Workers can input as many answers as possible or mark the question as "not answerable". For each answer, they can select one or more supporting evidences as the answer's conditions if needed. Workers are asked not to select conditions if there is sufficient information in the scenario to answer the question. We give workers permission to slightly modify the questions or scenarios if the questions are not clearly stated, or they can mark it as a bad question (different from not answerable) so we will drop it from the dataset.

We additionally perform a revision step to improve the annotation quality. We provide the union of selected evidences and answers from multiple annotations of a question to an additional group of annotators and let them deselect unrelated evidences and merge answers. As the amount of information provided to workers at this step is significantly less than in the previous answer selection stage, the annotation quality improves significantly. We end up with 3102 questions with annotated answers.

7.3.4 Move Conditions to Scenario

To encourage the model of learning subtle difference in user scenarios that affects the answers and conditions, we create new questions by modifying existing questions with conditional answers by moving one of the conditions to their scenarios.

Specifically, we show the workers the original questions, scenarios, and the annotated answers and conditions. Evidences are also provided for workers to get them familiar with the background of the questions and reasoning performed to get the original answers. Workers are asked to pick one of the conditions and modify the original scenario to reflect this condition. The modified questions and scenarios are sent back to the answering stage to get their annotations. We randomly select a small portion of the questions that have conditional answers as inputs to this stage so as to not affect the original distribution of the dataset. We collected 325 additional examples from this stage.

7.3.5 Train / Dev / Test Splits

We partition the dataset by documents to prevent leaking information between questions from the same document. The dataset contains 436 documents and 2338 questions in the training set, 59 documents and 285 questions in the development set, and 136 documents and 804 questions in the test set. Please see §7.4 for more statistics on ConditionalQA.

7.4 Dataset Analysis

The dataset consists of yes/no questions and extractive questions. Questions may contain one or more answers, with or without conditions. The statistics of the questions are shown in Table 7.2.

Answer type Among all the answerable questions, 1751 questions have yes/no answers while the other 1527 questions have extractive answers. 1161 of the yes/no questions have the answer "yes", 712 questions have answer "no", and 122 questions have both answers "yes" and "no" under different conditions. Please see the example in Table 7.1. The average length of the extract answers is 6.36 tokens.

Condition type 803 questions have conditional answers. 390 out of the 803 questions have one answer, but this answer is only correct if the conditions are satisfied. 173 questions have multiple answers, each have their own conditions, i.e. the answers are different if different conditions apply. The rest 240 questions also have multiple answers, but some of the answers require conditions while other don't. See examples in Table 7.1. A total of 1090 answers from 803 questions have conditions, among which 672 answers have only one condition and 418 answers have multiple conditions.

Number of answers Besides questions that have different answers under different conditions, 339 questions have multiple deterministic answers.

Type	**Scenario & Question**	**Answer w/ [*Conditions*]**
Single answer	**Scenario**: "My father has recently appealed for a traffic ticket." **Question**: "How long will it take to get a decision?"	• "4 weeks"
Single answer w/ conditions	**Scenario**: "I applied to cut down a tree on my land but it was rejected 20 days ago" **Question**: "Am I still able to appeal against it?"	• "yes" [*"<p> You can appeal before the date the tree replacement notice comes into effect.</p>"*]
Multiple answers	**Scenario**: "I will get my first paycheck tomorrow." **Question**: "What information should be on my pay split?"	• "earnings before and after any deductions" • "the amount of any deductions" • "the number of hours you worked"
Multiple yes/no w/ conditions	**Scenario**: "I am looking at buying a new build home. I am 26 and a first-time buyer." **Question**: "Am I eligible to get an Equity Loan?"	• "yes" [*"<li>able to afford fees and interest<li>", "<li>sold by an eligible homebuilder</li>"*] • "no" [*"<p> You can not apply if you had any form of sharia mortgage finance</p>"*]
Multiple extractive w/ conditions	**Scenario**: "I always walk my dog in open spaces. I forgot to clean up his mess yesterday." **Question**: "How much can I be fined for this?"	• "$100" [*"<li>$100 on the spot</li>"*] • "up to $1000" [*"<li>up to $1,000 if it goes to court</li>"*]
Multiple extractive one w/ condition	**Scenario**: "I am about to apply for a Parent of a Child Student Visa to stay with my child for a year in the UK" **Question**: "What documents are needed to apply for this visa?"	• "a current passport or other travel document" • "proof that you have enough fund" • "your tuberculosis (tb) test results" [*"<li>your tuberculosis (TB) test results if you are from a country where you have to take the TB test</li>"*]

Table 7.1: Example of questions in ConditionalQA. Text pieces that follows the answers in the brackets are [*conditions*]. Some answers are deterministically correct so they are not followed by conditions.

	Type	**#**
Answer type	yes / no	1751
	extractive	1527
Condition type	deterministic	2475
	conditional	803
Number of answers	single	2526
	multiple	752
–	not answerable	149

Table 7.2: Statistics on different types of questions.

7.5 Challenges

7.5.1 Baselines

Evaluating existing models on ConditionalQA is challenging. In addition to predicting answers to questions, the ConditionalQA task also asks the model to find the answers' conditions if any of them applies. Traditional reading comprehension models also face the challenge that the context of questions in ConditionalQA is too long to fit into the memory of many Transformer-based models like BERT [42] and even ETC [4]. To the best of our knowledge, no existing model fits the purpose of this task. We modified three competitive QA models to serve as baselines to the ConditionalQA dataset. The baseline models we implemented are described below.

ETC: ETC [4] is a pretrained Transformer-based language model that is designed for longer inputs (up to 4096 tokens). ETC achieved the state-of-the-art on several challenging tasks, e.g. HotpotQA and WikiHop [177, 169]. Since ETC cannot fit the entire document (with up to 16154 tokens) into its memory, we cannot let ETC jointly predict answers and conditions, so we designed a two stage pipeline to run ETC on ConditionalQA. In the first stage, ETC is trained as a normal reading comprehension model to predict answers from the document by jointly encoding the questions and documents. We adopt a sequential reading approach that reads one section at a time. The answer with the highest probability among all sections will be considered as the final answer. We append three special tokens "`yes`", "`no`", and "`not answerable`" for the yes/no and not answerable questions. Since it is not clear how to extract multiple answers with the Transformer-based extractive QA model, we restrict to the number of predicted answers to one. The second stage in the pipeline is to select conditions. Questions, answers, and documents are concatenated together into a single input for ETC. We then use the embeddings of global tokens for sentences in ETC to predict conditions. Since the number of conditions for the answer is unknown, we train the condition selection process with a binary classification target, by labeling each global token as positive or negative. The threshold of selecting conditions is a hyper-parameter.

DocHopper: DocHopper [147] is an iterative attention method that extends ETC for reading long documents to answer multi-hop questions. It reads the full documents once and jointly predicts answers and conditions. The model iteratively attends to information at different levels in the document to gather evidences to predict the final answers. We modify the iterative process in DocHopper for the purpose of this task: specifically, DocHopper is trained to run three iterative attention steps: (1) attend to the supporting evidences; (2) attend to the sentence that contains the answer; and (3) attend to the conditions. Since the query vector in each attention step is updated with information from the previous steps, conditions attended to at the third step are aware of the previously predicted answers. Unfortunately, DocHopper is still restricted to predicting one answer for each question. The condition selection step in DocHopper is also trained with binary classification loss. Different from the ETC pipeline, the three attention steps are jointly optimized.

	Yes / No		Extractive	
	answer	w/ conds	answer	w/ conds
majority	62.2 / 62.2	42.8 / 42.8	– / –	– / –
ETC	63.1 / 63.1	47.5 / 47.5	8.9 / 17.3	6.9 / 14.6
DocHopper	**64.9 / 64.9**	**49.1 / 49.1**	17.8 / 26.7	15.5 / 23.6
FiD	64.2 / 64.2	48.0 / 48.0	**25.2 / 37.8**	**22.5 / 33.4**
human	91.4 / 91.4	82.3 / 82.3	72.6 / 84.9	62.8 / 69.1

	Conditional		Overall	
	answer	w/ conds*	answer	w/ conds
majority	– / –	– / –	– / –	– / –
ETC	39.4 / 41.8	2.5 / 3.4	35.6 / 39.8	26.9 / 30.8
DocHopper	42.0 / 46.4	3.1 / 3.8	40.6 / 45.2	31.9 / 36.0
FiD	**45.2 / 49.7**	**4.7 / 5.8**	**44.4 / 50.8**	**35.0 / 40.6**
human	74.7 / 86.9	48.3 / 56.6	82.6 / 88.4	73.3 / 76.2

Table 7.3: Experiment results on ConditionalQA (EM / F1). Numbers are obtained by re-running the open-sourced codes of the baselines. "majority" reflects the accuracy of always predicting "yes" without conditions. *See discussion in text.

FiD: FiD [65] is a generative model with an encoder-decoder architecture. The encoder reads multiple contexts independently and generates their embeddings. The decoder attends to all embeddings of the context to generate the final answers. In this task, we train FiD to sequentially generate the answers with conditions, i.e. $[a_1, c_{11}, c_{12}, \ldots, a_2, c_{21}, c_{22}, \ldots]$ where $\{a_1, \ldots, a_n\}$ are the correct answers and $\{C_1, \ldots, C_n\}$ are their conditions, i.e., $c_{ij} \in C_i$ is the j'th condition for the answer a_i. If C_i is empty, the model is trained to predict "NA" as the only condition for the i'th answer. FiD can predict multiple answers as opposed to ETC and DocHopper.

Human We randomly sample 80 questions and ask human annotators to answer them. Annotators are provided with the full instructions and 10 additional annotated examples to clarify the task. We do not provide additional training to the annotators.

7.5.2 Results

Experiment results are shown in Table 7.3. We report the numbers on yes/no questions and extractive questions separately. The numbers in Table 7.3 show that the ConditionalQA task is very challenging—the performance of the best model on yes/no questions is 64.9% (marginally higher than always predicting the majority answer "yes"), and the performance on extractive questions is 25.2% EM. FiD has the best performance on extractive questions because FiD can predict multiple answers while ETC-pipeline and DocHopper only predict one.

The performance drops significantly if answers and conditions are jointly evaluated. The best performance on jointly evaluating answers and conditions ("w/ conditions") in Table 7.3 is only 49.1% for yes/no questions and 22.5% EM for extractive questions. Even worse, this best result is

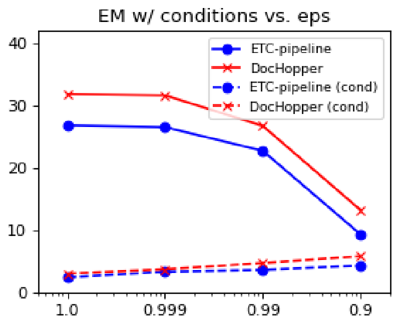


Figure 7.2: EM of answers with conditions with different thresholds of confidence (eps) on conditions. Dotted lines represent experiment results on the subset of questions that have conditional answers.

obtained when *no* condition is selected, i.e. the threshold of selecting conditions is 1.0. The difficulty of selecting conditions is more obvious if we focus on the subset of questions that have at least one conditional answer. The accuracy drops by 90% if answers and conditions are jointly evaluated.[3]

We also study how the threshold on the confidence scores of selecting conditions affects the evaluation results. Results are shown in Figure 7.2. As we decrease the threshold for selecting conditions, the EM with conditions on the subset of questions that have conditional answers slightly improves, but the overall EM with conditions drops dramatically due to the false positive conditions. FiD is a generative model so we can not evaluate it in the same way. In our evaluation, predictions from the best performing FiD checkpoint also do not select any conditions.

Table 7.5 shows the best results on the subset of questions that have conditional answers. Hyper-parameters are tuned on the subset of questions. We could possibly get better results on questions with conditional answers with threshold $\epsilon < 1.0$, but the improvement is still marginal.

7.5.3 Error Analysis

We manually check 200 examples in the prediction of the best performed model FiD and label the type of errors made. The numbers are shown in Table 7.4. The most errors are made when only a subset of correct answers is predicted. This is due to the fact that the model (FiD) has a tendency to predict one answer for each question. The second most common errors are made by predicting answers with the correct type but wrong value. Such errors are commonly made by reading comprehension models in many tasks. The model made a lot of errors in yes/no questions because they consist of around 50% of the questions. The model is good at distinguishing yes/no questions and extractive question as producing the wrong kind of answer only makes up of 4.2% of the errors.

[3]The EM/F1 w/ conditions* is non-zero on this subset of questions even if no condition is ever selected, because some questions have both conditional and deterministic answers. Models get partial credit if they predict the deterministic answers correctly.

Error types	**%**	**Examples**	**Correct answers**	**Predictions**
Not answerable	7.6	”Am I eligible for a tax reduction?”	not_answerable	”yes”
Wrong answer type (yes/no vs. extractive)	4.2	”How can I check if this design has been registered?”	”ask the intellectual property office to search for you”	”no”
Wrong answer (yes/no)	19.5	”Will it be classed as a small vessel?”	”yes”	”no”
Wrong answer (extractive, right type)	20.3	”How many points will I receive on my license?”	”6”	”3”
Wrong answer (extractive, wrong type)	9.3	”What is the account number should I send the money to?”	”12001020”	”hmrc”
Correct answer w/ wrong conditions	14.4	”Can I still send simpler annual accounts as a micro-entity?”	”yes”, [”*$316,000 or less on its balance sheet*”]	”yes”, []
Partial answer	24.5	”What will not need to be repeated for each trip?”	”a microchip”, ”rabies vaccination”	”a microchip”

Table 7.4: Error analysis on the predictions of the best performed model (FiD). The percentage is the fraction of errors made in that category over all errors.

	Best Overall	Best Conditional
ETC	2.5 / 3.4	4.4 / 4.6
DocHopper	3.1 / 3.8	**5.9 / 7.1**
FiD	**4.7 / 5.8**	4.7 / 5.8

Table 7.5: EM/F1 w/ conditions on the subset of questions with *conditional answers*. “Best Overall” uses the best checkpoints/hyper-parameters on the full dataset, while “Best Conditional” uses the best ones on the subset of questions.

7.6 Related Works

Many question answering datasets have been proposed in the past few years [125, 124, 177, 34, 50, 75] and research on these has significantly boosted the performance of QA models. As large pretrained language models [42, 93, 4, 11, 59, 159] achieved better performance on traditional reading comprehension and question answering tasks, efforts have been made to make the questions more complex. Several multi-hop QA datasets were released [177, 50, 154, 169] to test models’ ability to solve complex questions. However, most questions in these datasets are answerable by focusing on a small piece of evidence at a time, e.g. a sentence or a short passage, leaving reasoning through long and complex contents a challenging but unsolved problem.

Some datasets have been recently proposed for question answering over long documents. QASPER [34] contains questions asked from academic papers, e.g. “What are the datasets experimented in this paper?". To answer those questions, the model should read several sections and collect relevant information. NarrativeQA [107] requires reading entire books or movie scripts to answer questions about their characters or plots. Other datasets, e.g. HybridQA [22], can also be viewed as question answering over long documents if tables with hyper-linked text from the cells are flattened into a hierarchical document. ShARC [135] is a conversational QA dataset that also use UK government websites as its corpus. However, the ShARC dataset only contains yes/no questions and the conversation history is generated by annotators with the original rule text in hand, making the conversation artificial. The length of context in ShARC is usually short, such as a few sentences or a short paragraph. While using the same corpus, ConditionalQA contains completely different questions and new types of answers. It focuses on a new problem that has not been previously studied.

Most of the existing datasets, including the ones discussed above, contain questions with unique answers. Answers are unique because questions are well specified, e.g. "Who is the president of the US in 2010?". However, questions can be ambiguous if not all information is provided in the question, e.g. "When was the Harry Potter movie released?" does not specify which Harry Potter movie. AmbigQA [104] contains questions that are ambiguous, and requires the model to find all possible answers of an ambiguous question and rewrite the question to make it well specified. Similar datasets Temp-LAMA [44], TimeQA [21] and SituatedQA [183] have been proposed that include questions that require resolving temporal or geographic ambiguity in the context to find the answers. They are similar to ConditionalQA in that questions are incomplete, but ConditionalQA focuses on understanding documents with complex logic and answering questions with conditions. It's usually not possible to disambiguate questions in ConditionalQA as rewriting the questions (or scenarios) to reflect all conditions of answers to make the questions deterministic is impractical.

We create ConditionalQA in the public policy domain. There are some existing domain specific datasets, including PubMedQA and BioAsq [109, 67] in medical domain, UDC [96] in computer software domain, QASPER [34] in academic paper domain, PrivacyQA and PolicyQA [2, 127] in legal domain, etc. PrivacyQA and PolicyQA have similar context as ConditionalQA, but the questions do not require compositional reasoning and the answers are short text spans. We use a corpus in the public policy domain because it is easy to understand by non-experts while being complex enough to support challenging questions. ConditionalQA is not designed to be a domain specific dataset.

7.7 Discussion

With preliminary experiments, we showed baseline QA models did not perform well on this challenging conditional reasoning task. The performance was especially low in predicting missing conditions. In the next chapter, we will propose a model, T-Reasoner, which significantly improves the performance of ConditionalQA.

Chapter 8

Reasoning over Logically Interacting Constraints

8.1 Overview

Many real world questions have context-dependent answers. For example, “Which was the first approved Covid vaccine” has different answers in different countries, i.e. geographical constraint. Recent datasets have been proposed [44, 183, 21] to answer natural language questions under specific constraints, e.g. “Which was the first approved Covid vaccine [in the U.S.]”. Such questions require QA models to find candidate answers and additionally verify whether the constraints have been properly satisfied.

In this work, we follow the direction of Question Answering with constraints but focus on a more challenging task where the constraints are a set of conditions that interact under some complex logical operations. An example is shown in Figure 8.1. The candidate answer “\$60 a week” is restricted by two conditions “physically or mentally disabled” and “age 59 1/2 or older”, under the logical operator “if you are both” indicating that both conditions have to be satisfied in order to make the answer “\$60 a week” correct. In addition, answering such questions requires performing logical reasoning over the conditions to check whether a candidate answer is eligible. The context could be more complex as the number of conditions increases or the logical operations become more complex. Such questions are commonly seen in many professional domains, e.g. legal and financial domains.

Different from previous work [24] that assumes all information needed to make a definite prediction is provided (which is called *deductive reasoning*), we do not make such assumption in the Conditional QA task but instead ask the model to predict the most probable answer and identify

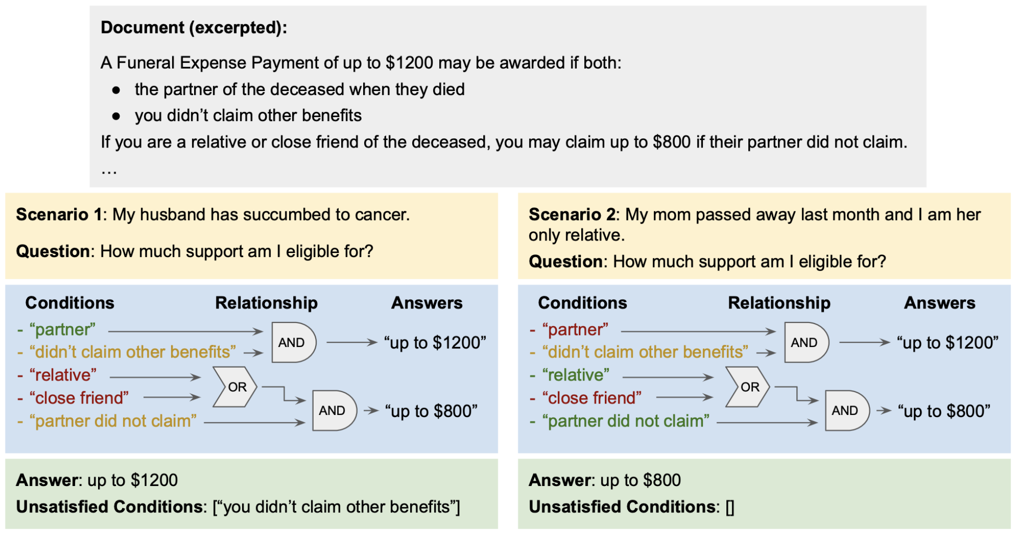

Document (excerpted):
A Funeral Expense Payment of up to $1200 may be awarded if both:
● the partner of the deceased when they died
● you didn't claim other benefits
If you are a relative or close friend of the deceased, you may claim up to $800 if their partner did not claim.
...
Scenario 1: My husband has succumbed to cancer.
Question: How much support am I eligible for?
Scenario 2: My mom passed away last month and I am her only relative.
Question: How much support am I eligible for?
Conditions
Relationship
Answers
- "partner"
- "didn't claim other benefits"
- "relative"
- "close friend"
- "partner did not claim"
AND
OR
AND
"up to $1200"
"up to $800"
Answer: up to $1200
Unsatisfied Conditions: ["you didn't claim other benefits"]
Answer: up to $800
Unsatisfied Conditions: []

Figure 8.1: Examples for the scenario-based QA task. The two examples have the same question but different scenarios which lead to different answers. Diagrams in the blue blocks show the desired reasoning process. The conditions in green, red, and yellow are entailed, contradicted, and not mentioned in the scenario. In Scenario 1, the answer "up to $1200" is logically consistent with the scenario. The condition "you didn't claim other benefits" is an unsatisfied condition because it is not mentioned but required to be satisfied for the answer. The other condition "partner did not claim" is also not mentioned but is not an unsatisfied condition because whether it is satisfied will not affect the answer. In Scenario 2, there's not an unsatisfied condition because existing information provided in the scenario is sufficient to predict the answer "up to $800".

the conditions that needs to be checked if their information is not provided in the question.[1] For example (Figure 8.1), we say "age 59 1/2 or older" is a missing condition because it is required by the candidate answer "$60 per week" but is not mentioned in the user's scenario. Predicting missing conditions tests a model's ability in performing logical reasoning tasks that includes understanding logical operations, propagating the entailment status of conditions in the logical groups, and finally determining whether a candidate answer is eligible. In this section, we propose our method to solve this complex logical reasoning task using neural network models.

8.2 T-Reasoner

8.2.1 Task: QA with Conditions

The scenario-based QA task requires models to find answers that are logically consistent with the provided user scenarios which are potentially incomplete. In this work, we consider this task in the reading comprehension (RC) setting in which a passage that contains relevant information about the

[1] This task is called *abductive reasoning* in logical inference, but we will use the term conditional QA in this section.

question is provided. We leave the open-domain setting of this problem for future work. Specifically, a model takes a question, a scenario, and a passage that contains answers and conditions as input and predicts logically consistent answers and their unsatisfied conditions. Let's consider a passage that contains a set of conditions $C = \{c_1, \ldots, c_n\}$ and the set of eligible answers for a question under all possible combinations of conditions $A = \{a_1, \ldots, a_m\}$. Each answer $a_i \in A$ is restricted by a subset of conditions $C_i \subseteq C$. Conditions in C_i interact with each other under relationship R_i (R_i is an abstract set which will not be explicitly expressed). A condition group, $G_i = (C_i, R_i)$ is a pair of C_i and R_i, which describes in what scenario the answer a_i is correct. Note that the list of answers A, sets of conditions C_i's and their relationship R_i's are not explicitly provided in training and testing examples – models have to generate them from the passage.

We say that a condition group G_i is *satisfied* if its underlying logical statement that consists of C_i and R_i has been satisfied by the scenario, for example, in Scenario 2 in Figure 7.1 where the condition group for "up to \$800" has been satisfied. Besides being satisfied, a condition group G_i has two more possible outcomes: (1) G is *partially satisfied* if some of the conditions have been satisfied but there is still some information missing so the answer is not fully supported, e.g. the condition group of the answer "up to \$1200" in Scenario 1 (Figure 7.1), and (2) G_i is *contradicted* if one or more conditions in the group are contradicted which causes the answer ineligible, e.g. the condition group of the answer "up to \$1200" in Scenario 2 (Figure 7.1).

An answer a_i is logically consistent with the scenario if the underlying condition group G_i is satisfied or partially satisfied. We denote the set of logically consistent answers $\tilde{A} \subseteq A$. The set $\tilde{A}$ contains zero or more answers – the set $\tilde{A}$ is empty if none of the answers in A is logically consistent with the user scenario. A model should predict an answer from $\tilde{A}$ if $\tilde{A}$ is not empty, and mark the question as not answerable, otherwise.[2] In addition to predicting logically consistent answers, we also perform the task of finding unsatisfied conditions $\tilde{C}_i$. The set $\tilde{C}_i$ should be concise, i.e. it should only include the conditions that are necessary. For example, the condition "have worked for more than 4 years" is not an unsatisfied condition because whether it has been satisfied or not won't affect the output of the condition group.

In summary, we evaluate a model's prediction of a logically consistent answer $a_i \in \tilde{A}$ and the set of unsatisfied conditions $\tilde{C}_i$ for answer a_i, i.e. $(a_i, \tilde{C}_i)$. Answers and unsatisfied conditions in the output are jointly evaluated.[3] This task specifically challenges models' ability in understanding the relationship between conditions and performing logical reasoning process accordingly. We will introduce a simple and effective model, TReasoner, to tackle this challenging reasoning task.

[2]An oracle model should be able to predict all answers from $\tilde{A}$. We consider a slightly simplified setting in this work in which a model is only required to predict one of the answers. In our experiments, the ShARC [136] dataset only contains questions that have a single answer, i.e. $|\tilde{A}| = 1$. The ConditionalQA [146] dataset contains questions that have multiple answers, $|\tilde{A}| > 1$, so the performance will be sacrificed. We leave the task of predicting all logically consistent answers as future work.

[3]Evaluation metrics are different in different datasets [146, 136]. Please refer to §8.3.3 and 8.3.2 for more details.

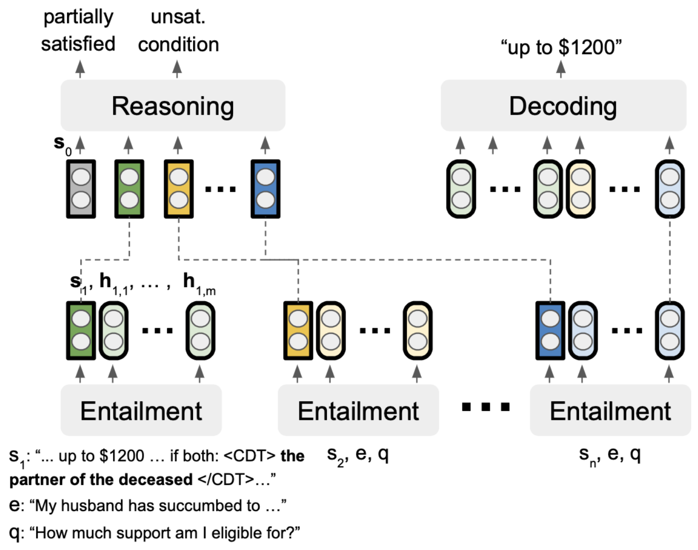

partially satisfied
unsat. condition
"up to $1200"
Reasoning
Decoding
s_0
s_1, $h_{1,1}$, … , $h_{1,m}$
Entailment
Entailment
Entailment
s_1: "... up to $1200 … if both: <CDT> **the partner of the deceased** </CDT>…"
e: "My husband has succumbed to …"
q: "How much support am I eligible for?"
s_2, e, q
s_n, e, q

Figure 8.2: **TReasoner Overview**. The entailment module independently encodes each condition c_i (along with its contextual information s_i). The entailment module outputs a condition embedding $\mathbf{s}_i$ that will be input into the reasoning module to decide if conditions have been satisfied and identify unsatisfied conditions. Other computed embeddings $\mathbf{h}_{i,j}$ will be used by the decoding module to generate answer spans (if the question has a free-form answer).

8.2.2 Model

In this section, we will discuss TReasoner which consists of an entailment module, a reasoning module, and optionally a decoding module, to perform this challenging QA task in embedding space.

Input The model, TReasoner, takes a question q with scenario e and a passage p as inputs and predicts an answer a_i that is logically consistent with the user scenario and a list of unsatisfied conditions $\tilde{C}_i$. Since the list of all conditions C for the question are not provided in the example, we chunk the passage p into pieces and consider each piece of text as a condition c_i. Conditions obtained this way may be irrelevant to questions. We rely on the entailment module (see next) to decide whether a condition c_i is relevant and what is its relationship with others. The chunking strategy may be different for different datasets. Please see §8.3.2 and §8.3.3 for more information. Briefly, passages are usually chunked into sentences, short passages with 2-3 sentences, or sub-sentences (text phrases).

Entailment Module We apply an entailment module to check whether each condition $c_i \in C$ have been entailed by the user scenario. Each condition c_i is checked independently, as opposed to concatenating all conditions into a long input and checking them all at once. This strategy significantly reduce the computation cost compared to checking all conditions at once, especially if context is long, e.g. legal documents which are tens or hundreds of pages long (see examples in

8.3.3). Specifically, the computation complexity of our approach is $O(|C|)$ where $|C|$ is the number of total conditions, compared to a complexity of $O(|C|^2)$ otherwise.

This independent checking strategy, however, separates each condition from its context and thus causes a lost of contextual information for conditions c_i and eventually negatively impacts the model's performance. Thus, we extend a condition c_i by adding tokens from its surroundings. For example, the condition "the partner of the deceased when they died" is expanded to "... up to $1200 may be awarded if both: `<CDT>` the partner of the deceased when they died `<\CDT>` you didn't claim ...", where `<CDT>` and `<\CDT>` are two special tokens that mark the beginning and end of the condition c_i. Apart from making a condition c_i more fluent and coherent, the added contextual tokens also make it easier to understand the relationship between the current condition c_i and other conditions in its neighbours. We may additionally add page titles, section titles, prompts of list items, or table headings, etc., if applicable to the expanded conditions. Please see §8.3.3 and 8.3.2 for more details. We denote conditions with extended contextual information as s_i for condition c_i.

We learn a Transformer model for the entailment module which takes an expanded condition s_i and the question q and scenario e as input, and returns a list of vectors $\mathbf{s}_i, \mathbf{h}_{i,1}, \ldots, \mathbf{h}_{i,m}$. The first vector $\mathbf{s}_i$ is a summarization vector which includes several aspects of information: (1) whether the underlying condition c_i has been satisfied, contradicted, or not mentioned by the user scenario, (2) whether the condition c_i is relevant to the question, and (3) if relevant, what is its relationship with other conditions in its neighbours. These information will be used for reasoning in the future layers. Embeddings $\mathbf{h}_{i,1}, \ldots, \mathbf{h}_{i,m}$ are token embeddings that will used for decoding if needed. Please see the description of the reasoning module for more information.

$$\mathbf{s}_i, \mathbf{h}_{i,1}, \ldots, \mathbf{h}_{i,m} = \text{Entail}(s_i, e, q) \tag{8.1}$$

One may consider supervising this entailment module by adding classification layers on $\mathbf{s}_i$ to explicitly predict the entailment status of condition c_i and its relationship with other conditions. However, obtaining supervision labels for these auxiliary tasks can be challenging as they are often not provided in the example. Fortunately, we show that our proposed model, TReasoner, can be trained end-to-end, without such intermediate supervision.

Decoding Module The decoding module generates an eligible answer $\hat{a}_i$ which is potentially logically consistent to the question. The generated answer $\hat{a}_i$ will not be returned until the status of its condition group $\hat{G}_i$ is verified by the reasoning module (discussed below).

The decoding module is analogous to FiD [64], i.e. token embeddings from different conditions (which are encoded separately) are concatenated for decoding. Different from [64] which was applied to independently retrieved passages for open-domain QA, the decoding module in TReasoner is used on coherent content, i.e. conditions from the same passage. The contextual information in the expanded condition s_i helps connect conditions that are separately encoded. The decoding module takes token embeddings for all conditions $\mathbf{h}_{1,1}, \ldots, \mathbf{h}_{n,m}$ computed from Eq. 8.1 to generate answer spans. The generation task is trained with teacher forcing. We do not write out the teacher forcing decoding loss l_{decode} here. Please refer to the T5 paper [122] for more information. If questions have multiple logically consistent answers, i.e. $\tilde{A} > 1$, we randomly select an answer $a_i \in \tilde{A}$ as the label to train the decoding module.

$$\hat{a}_i = \text{Decode}(\mathbf{h}^{(1)}_{1,1}, \ldots, \mathbf{h}^{(n)}_{k_n,m}) \tag{8.2}$$

We consider two different types of answers: "Yes"/"No" or free-form answers. In the first case, we simply let the model generate a special a special token `[YESNO]` and consider the reasoning result from the reasoning module (see next) as the answer, i.e. the answer is "Yes" if the condition group is satisfied (or partially satisfied) or "No" if contradicted. Since some datasets only contain "Yes"/"No" questions, we can then safely discard the decoding module for these datasets. In the second case, i.e. answers are free-form text spans, we will return generated spans as answers only if their condition groups have been verified as satisfied or partially satisfied by the reasoning module. If the condition group is contradicted, we will mark the question as not answerable.

Reasoning Module The reasoning module combines the local relationship between conditions from their embeddings $\mathbf{s}_1, \ldots, \mathbf{s}_n$ and performs a logical reasoning process to decide the reasoning result for a condition group G_i for the generated answer $\hat{a}_i$ and to identify unsatisfied conditions $\tilde{C}_i$.

The input to the reasoning module is a list of vectors, $\mathbf{s}_1, \ldots, \mathbf{s}_n$ for conditions $c_1, \ldots, c_n$, that are output by the entailment module (Eq. 8.1). We use another Transformer model as our reasoner, because Transformers have the self attention mechanism which allows conditions $\{s_1, \ldots, s_n\}$ to attend to each other, so the reasoning result of a condition group can be summarized. This is crucial because, for example, if one of the conditions in a disjunction group is satisfied, the condition group will be automatically satisfied regardless the status of other conditions in the same group. We prepend a trainable vector $\mathbf{s}_0$ to the list of condition embeddings to summarize the reasoning result.

The output vectors $\hat{\mathbf{s}}_0, \hat{\mathbf{s}}_1, \ldots, \hat{\mathbf{s}}_n$ will be used to predict the status of the condition group and the unsatisfied conditions for the generated answer. The first vector $\hat{\mathbf{s}}_0$ will be used to predict the reasoning result of the condition group. If the condition group is partially satisfied, we use the rest of vectors, $\hat{\mathbf{s}}_1, \ldots, \hat{\mathbf{s}}_n$, to identify unsatisfied conditions. We compute loss on both reasoning and unsatisfied condition predictions. Let $\mathbb{I}_r$ and $\mathbb{I}_c$ be the one-hot labels for the two tasks.

$$\hat{\mathbf{s}}_0, \hat{\mathbf{s}}_1, \ldots, \hat{\mathbf{s}}_n = \text{Reason}(\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_n)$$
$$l_{\text{reason}} = \text{softmax\_cross\_entropy}(\mathbf{W}_l^T \hat{\mathbf{s}}_0, \mathbb{I}_r)$$
$$l_{\text{cond}} = \text{softmax\_cross\_entropy}(\mathbf{W}_c^T \hat{\mathbf{s}}_i, \mathbb{I}_c)$$

As discussed above (§8.2.1), the reasoning results of condition groups have three possible outcomes: "satisfied", "partially satisfied", and "contradicted". We merge the first two into one label "satisfied", and differentiate them by whether unsatisfied conditions exist, i.e. $r \in \{\text{satisfied}, \text{contradicted}\}$

and its one-hot label $\mathbb{I}_r \in \{0,1\}^2$.[4]

Labels for conditions are "entailed", "contradicted", "not mentioned", "implied", and "unsatisfied", i.e. $\mathbb{I}_c \in \{0,1\}^5$. The first three labels are as they are named. The label "implied" means a condition is implied by other conditions in the condition group. For example, if one of the conditions in a disjunction group has been satisfied, the rest of conditions are "implied". The class "unsatisfied" means it is an unsatisfied condition which must be returned together with the predicted answer. The labels may not apply to all datasets, e.g. ConditionalQA [146] only annotates two labels "unsatisfied" vs. others, we will make changes to the loss function accordingly.

Loss Function We jointly train the entailment module and reasoning module. The final loss function is the sum of the answer loss l_{reason} and the condition entailment loss l_{cond}. If the answers contain text spans, we jointly train the decoding module l_{decode} as well.

$$l = l_{\text{reason}} + l_{\text{cond}}$$
$$l = l_{\text{reason}} + l_{\text{cond}} + l_{\text{decode}}$$

8.2.3 Finetune Pretrained Checkpoints

The entailment module and decoding module (if adopted) load pretrained LM checkpoints, e.g. T5 [122] and BART [80]. The pretrained parameters are loaded for the entailment module and then finetuned for downstream tasks. The reasoning module is randomly initialized and jointly trained with other modules. The number of Transformer layers in the reasoning module is a hyper-parameter. We choose the number of layers $l = 3$ or $l = 4$. Please see §8.3.1.2 for ablation study on the number of Transformer layers for the reasoning task. The decoding module is also finetuned. If a decoding module is needed, we will initialize the entailment and decoding module from the same pretrained checkpoint.

8.3 Experiments

We experiment with TReasoner on a synthetic dataset, CondNLI, and two benchmark QA datasets, ConditionalQA [146] and ShARC [136], for scenario-based QA task.

8.3.1 CondNLI

8.3.1.1 Dataset Overview

The synthetic CondNLI dataset is derived from an existing Natural Language Inference (NLI) dataset, MultiNLI [172]. An original NLI example contains a premise and a hypothesis, and a label indicating whether the premise is entailed or contradicted by the hypothesis. We treat premises

[4]Some tasks have an additional class "irrelevant" because some questions in the dataset are not relevant to the provided passages, i.e. $\mathbb{I}_r \in \{0,1\}^3$.

(Template)	
Context: If all [A, B], then U.	
If any [not C, D], then V.	
Facts: a, c, not d.	
Question: Is u correct?	
Label: Yes, if B	
(Variables)	
A: Aged 59 1/2 or older.	a: Tom is 65 years old.
B: Employed for two years.	b: NOT_USED
C: Has two children	c: He has two sons.
D: Has not applied before.	not d: Rejected last year.
U: Get at least $60 a week	u: Eligible for $60 a week.
V: Waive the application fees	v: NOT_USED

Table 8.1: An example of CondNLI. Variables A, B, ... and U, V, ... represent the conditions and premises. Variables a, b, ... represent the known facts. u is the question. Each pair of variables, e.g. (A, a), is instantiated with an NLI example.

in NLI examples as conditions and hypotheses as facts provided in user scenarios. An example is shown in Table 8.3. The example contains four conditions, among which "Aged 59 1/2 or older" and "Employed for two years" belong to a condition group under a logical reasoning type "all", indicating that both conditions have to be satisfied in order to "Get at least $60 a week". The answer statement, e.g. "Get at least $60 a week", also comes from NLI examples. We treat the premise of an NLI example as an answer statement and the corresponding hypothesis as the question, e.g. is "Eligible for $60 a week" correct? In addition to the condition group and the answer statement that is relevant to the question, we add a few more condition groups as distractors to make the constructed dataset more challenging.

8.3.1.2 Data Construction

We first construct templates for the CondNLI examples and then instantiate the variables in the template with real NLI examples.

Construct Templates We use capital letter A, B, ... to represent conditions and lower-cased letters a, b, ... to represent the corresponding facts. We use another few letters X, Y, ... to represent the statements of conclusion, and lower-cased letters x, y, ... to represent questions. Conditions are grouped together under a logical operator that specifies the relationship between conditions. For example, a logical operator "all" specifies that all conditions in the group must be satisfied in order to make the condition group satisfied. Here, we consider four types of logical operations to construct this synthetic dataset:

- "all": all conditions under this logical type should be satisfied in order to make the answer true.

- "any": only requires one of the conditions under the logical type "any" to be satisfied. It doesn't matter whether other conditions have been satisfied, contradicted, or not mentioned in the question.
- "required": This is a special case of "all" / "any" when there is only one condition. Conditions with the logical type "required" must be satisfied.
- "optional": Conditions have the type "optional" if they are not relevant to the question.

We pair a condition group with a conclusion statement and get a logical statement "If all (A, B), then X". To challenge models' ability in identifying relevant conditions from context, we add a few distracting statement that leads to different conclusions, e.g. "If all (not C, D), then Y". An example of a context template is shown in Table 8.1.

Facts are constructed by randomly sampling a subset from all possible facts $\{a,\ b,\ \dots\}$. A question is sampled from possible questions $\{x, y, \dots\}$. We then compute the answer (and unsatisfied conditions if any) from the context, facts, and the question.

Generate Examples For a templates with variables A, B, X, Y, $\dots$, a, b, x, y, $\dots$, we instantiate the variables with NLI examples to get the real data. We use the premises of original NLI examples for conditions and conclusions, i.e. capital letter variables, and the hypothesis for question and facts, i.e. lower-case variables. Note that sampling requires matching the entailment state of conditions, e.g. "not d" requires sampling from NLI examples that are labeled as "contradict".

We restrict the number of conditions in the context to 6 and randomly generate 65 distinct templates.[5] During training, we randomly pick a template and instantiate it with NLI examples to generate real training examples. This random generation process enables creating (almost) unlimited amount of training data. We randomly generate another 5000 examples for development and testing.

Quality Assessment Training and validation data in CondNLI are generated from NLI examples in the training and validation split of the MNLI dataset, respectively. This ensures that NLI examples used in validation are not exposed at training time. We control the generation process to ensure that the automatically generated data are balanced in terms of answer labels, logical types of interacting conditions, and number of conditions included in scenarios. Results are shown in Table 8.2. We additionally require scenarios must have at least 4 conditions to avoid overly simple examples.

We additionally measure the Jaccard distance between premises and hypotheses of the NLI examples used in constructing the CondNLI dataset. The token-level Jaccard distance is 27.2. Even though token-level overlap exists, a model still needs to understand the semantic relationship between premises and hypotheses to predict their entailment status.

Baselines Previous work [24] showed that pretrained Transformer-based Language Models, e.g. RoBERTa [92], have the ability to reason over multiple conditions to answer a reasoning question

[5]Restricting the number of conditions is only for the purpose of reducing training complexity. The experiment in Figure 8.3 (left) shows the model's capability of generalizing to more conditions.

Answer labels (yes / no / negated yes / negated no / partially satisfied / irrelevant)	100 / 100 / 100 / 100 / 100 / 50
Logical types (any / all / required / optional)	378 / 380 / 422 / 390
# conditions in scenario (4 / 5 / 6)	333 / 340 / 471

Table 8.2: CondNLI statistics. The table shows the statistics of 550 randomly generated examples. The answer labels "negated yes" and "negated no" mean negation exist in rule templates. We set the minimum number of conditions in scenarios to 4 to avoid overly simple example. Each example may contain multiple logical types.

Context:	If all: [*"Aged 59 1/2 or older"*, *"Employed for two years"*] then *"Get at least \$60 a week"*. If any: [not *"Has two children"*, *"Has not applied before."*] then *"Waive the application fees"*.
Question:	Is *"Eligible for \$60 a week"* correct?
Scenario:	[*"65 years old"*, *"Rejected last year"*]
Answer:	Yes, [*"Employed for two years"*]

Table 8.3: An example in CondNLI. The answer is "Yes" with unsatisfied conditions [*"Employed for two years"*].

	Ans (acc)	Conds (F1)
T5 (w/ FiD)	85.6	80.4
ETC	91.4	82.7
TReasoner	95.0	91.3

Table 8.4: Experiment results on the synthetic CondNLI dataset in answer accuracy (Ans) and conditions F1 (Conds).

	6	8	10	15	20
T5 (w/ FiD)	85.6	85.2	85.3	82.2	74.1
TReasoner	95.0	94.3	94.5	92.8	91.7

Table 8.5: Experiment results in generalizing to more conditions at inference time.

in the deductive reasoning setting, e.g. "if A and B then C" with facts on both conditions A and B provided. However, examples in CondNLI are usually longer and won't fit into RoBERTa's memory. Equivalently, we experiment with two other language models, T5 [122] (with the FiD strategy [64] to adapt to longer input) and ETC [5], on the CondNLI dataset.[6] In ETC, we use the global tokens to predict unsatisfied conditions. In T5, To simplify the generation task, we assign an id to each condition and let FiD generate unsatisfied condition ids. We also compare TReasoner with T5 on inputs that contains more conditions to test their generalization ability.

8.3.1.3 Experimental Results

The experiment results are shown in Table 8.4. We measure both the accuracy of label prediction and the F1 of unsatisfied conditions. The results show that TReasoner performs significantly better than pretrained LMs, T5 and ETC, in both predicting correct answers (Ans) and unsatisfied conditions (Conds) on CondNLI. We additionally test TReasoner's ability in generalizing to more conditions. We train TReasoner on templates with 6 conditions or fewer and test it on the examples with more

[6]Examples in CondNLI exceeds the limit of 512 tokens in RoBERTa.

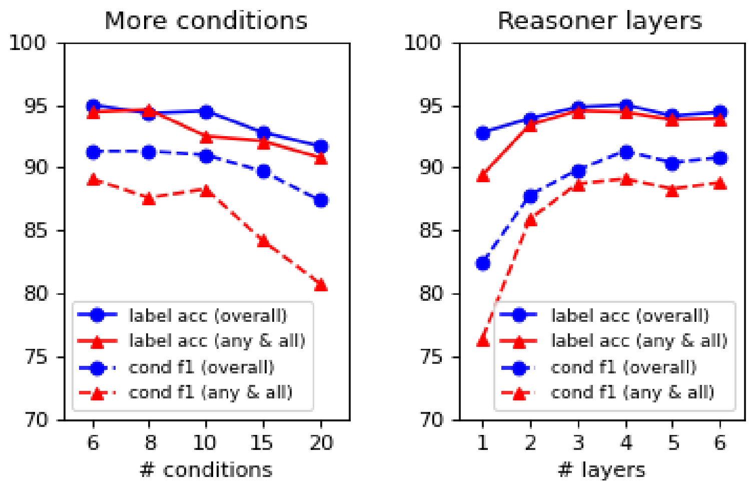


Figure 8.3: Left: Generalization results of reasoning over more conditions. Right: Results on the ablated model with different numbers of Transformer layers in the reasoning module. We report both label accuracy and F1 of unsatisfied conditions. "any & all" indicates the subset of validation data in which conditions interact each other under the relationship of "any" and "all".

than 6 conditions. Figure 8.3 (Left) shows the change of performance in both label classification and unsatisfied condition prediction tasks as the number of conditions increase. We observe some decrease in performance in both tasks, but it is still reasonable with 20 conditions. Furthermore, we experiment with different numbers of layers in the reasoning module (Right). The Transformer-based reasoning module needs at least 3 layers for the reasoning task, especially for predicting unsatisfied conditions.

8.3.2 ShARC

8.3.2.1 Dataset

In the second experiment, we run TReasoner on a real scenario-based QA dataset, ShARC [136], that has complex passages and many conditions. An example in ShARC contains a passage, a user question, and a user scenario which is expressed in a conversation history between a user and a machine. A model is expected to find an answer to the user's question, or raise a clarification question for the unsatisfied conditions. Answers in this dataset are restricted to one of the following labels: "yes", "no", "inquire", and "irrelevant". The first three labels are equivalent to "satisfied", "contradicted", and "partially satisfied". "irrelvant" is a new label that should be predicted if the conversation history and the question are irrelevant to the provided passage. This task of predicting answers is called "Decision Making" in their original ShARC paper [136] and evaluated as micro and macro accuracy. In addition to the "Decision Making" task, they consider another task "Question

Generation" which is equivalent to predicting unsatisfied condition in TReasoner,[7] evaluated with BLEU 1 and BLEU 4 scores. Compared to CondNLI, where conditions and their relationship are clearly mentioned in the context, conditions are embedded in the context in ShARC examples, e.g. Figure 7.1.

8.3.2.2 Implementation Details

Different from ConditionalQA, where each sentence in the context is treated as a condition, conditions in the ShARC dataset are shorter and are sometimes short phrases (sub-sentence). For example, the context "If you are a female Vietnam Veteran with a child who has a birth defect, you are eligible for ..." contains two conditions, "If you are a female Vietnam Veteran" and "with a child who has a birth defect".[8] In order to handle sub-sentence conditions, we follow the strategy proposed in two of the baseline models, DISCERN [54] and DGM [113], that split a sentence into EDUs (Elementary Discourse Units) using a pretrained discourse segmentation model [84]. The discourse segmentation model returns a list of sub-sentences, each considered as a condition.

While we could treat each condition independently as we did previously for other datasets, the segmented EDUs are different in that they are not full sentences and may not retain their semantic meaning. Thus, we consider using the full context (usually less than 512 tokens) as the contextual information for condition c_i, i.e. the expanded condition s_i includes the full context, but the condition c_i is highlighted using the special tokens `<CDT>` and `<\CDT>`.

We do not need the decoding module for the ShARC dataset, so we can safely discard it. We initialize the entailment module with ELECTRA [23]. The previous state-of-the-art baselines [113, 54] use ELECTRA to initialize their model. We use the same pretrained checkpoint to make a fair comparison.

For the question generation task, we use the same input s as in decision making, except that we replace the prefix "condition:" with "unsatisfied condition:" for "unsatisfied" conditions. We fine-tune a T5 model for question generation.

8.3.2.3 Baselines and Experimental Results

We compare TReasoner to several strong baseline models, including the previous state-of-the-art models, DISCERN [54] and DGM [113]. Different from the baseline models, which use pipeline systems to separately predict answer labels and unsatisfied conditions, TReasoner performs the two tasks jointly. The results are shown in Table 8.6. TReasoner outperforms the previous baselines by 3 points on the "Decision Making" task and more than 8 points on the "Question Generation" task. TReasoner also significantly outperforms other baseline models [136, 187, 162, 77, 53, 54, 113].

[7]Unsatisfied conditions are then paraphrased into questions, e.g. "Aged 59 1/2 or older" is paraphrased to "Are you aged 59 1/2 or older?"

[8]It is arguable that this could be generally treated as one condition, but it is treated as two conditions with the logical operator "all" in the ShARC dataset.

	Decision (micro / macro)	Question (BLEU1 / 4)
CM	61.9 / 68.9	54.4 / 34.4
BERTQA	63.6 / 70.8	46.2 / 36.3
UcraNet	65.1 / 71.2	60.5 / 46.1
Bison	66.9 / 71.6	58.8 / 44.3
E3	67.7 / 73.3	54.1 / 38.7
EMT	69.1 / 74.6	63.9 / 49.5
DISCERN	73.2 / 78.3	64.0 / 49.1
DGM	77.4 / 81.2	63.3 / 48.4
TReasoner	**80.4** / **83.9**	**71.5** / **58.0**

Table 8.6: Experimental results on the ShARC dataset. Numbers for the baseline models [136, 187, 162, 77, 53, 54, 113] are borrowed from [113].

	Decision (micro / macro)	Question (BLEU1 / 4)	Condition (F1)
T5	63.7 / 68.2	57.3 / 48.2	44.0
DISCERN	74.9 / 79.8	65.7 / 52.4	55.3
DGM	78.6 / 82.2	**71.8** / **60.2**	57.8
TReasoner	**79.8** / **83.5**	**71.7** / **60.4**	**69.2**

Table 8.7: Results on the ShARC dataset (dev) compared to the baselines. The Condition (F1) numbers are from open-sourced codes [54, 113].

Ablation: Condition Accuracy. One problem with the ShARC Question Generation task is that only one of the unsatisfied conditions is annotated, even though multiple unsatisfied conditions exist. To further evaluate TReasoner's performance in predicting all unsatisfied conditions, we manually annotate the logical operations in 20 contexts that have more than one condition (857 data total),[9] and use the annotated logical operations to find all unsatisfied conditions. We report the F1 of the predicted unsatisfied conditions (see Table 8.7). Compared to the baselines [54, 113], TReasoner improves the F1 by 11.4 points.

Ablation: Label Accuracy v.s. Conditions. We additionally measure the accuracy versus the number of conditions in the context. Results in Table 8.8 show that the improvement in TReasoner's performance over the previous state-of-the-art model (DGM) mostly comes from questions that have

[9]Each context in ShARC has 32.9 data on average.

# conditions	1	2	3	4
DGM	90.4	70.3	80.0	73.4
TReasoner	90.3	72.7	80.6	75.2
diff	-0.1	2.4	0.6	1.8

Table 8.8: Ablated results on ShARC on answer accuracy vs. number of conditions. Numbers of DGM [113] are obtained from open-sourced codes.

	Yes / No		Extractive	
	EM / F1	w/ conds	EM / F1	w/ conds
majority	62.2 / 62.2	42.8 / 42.8	– / –	– / –
ETC	63.1 / 63.1	47.5 / 47.5	8.9 / 17.3	6.9 / 14.6
DocHopper	64.9 / 64.9	49.1 / 49.1	17.8 / 26.7	15.5 / 23.6
FiD	64.2 / 64.2	48.0 / 48.0	25.2 / 37.8	22.5 / 33.4
TReasoner	**73.2 / 73.2**	**54.7 / 54.7**	**34.4 / 48.6**	**30.3 / 43.1**

	Conditional		Overall	
	EM / F1	w/ conds	EM / F1	w/ conds
majority	– / –	– / –	– / –	– / –
ETC	39.4 / 41.8	2.5 / 3.4	35.6 / 39.8	26.9 / 30.8
DocHopper	42.0 / 46.4	3.1 / 3.8	40.6 / 45.2	31.9 / 36.0
FiD	45.2 / 49.7	4.7 / 5.8	44.4 / 50.8	35.0 / 40.6
TReasoner	**51.6 / 56.0**	**12.5 / 14.4**	**57.2 / 63.5**	**46.1 / 51.9**

Table 8.9: Experimental results on ConditionalQA (EM / F1). The "EM/F1" columns reports the original EM/F1 metrics that are only evaluated on the answer span. The "w/ conds" is the conditional EM/F1 metric discussed in §7. Numbers of the baseline models are obtained from §7.

more than one condition.

8.3.3 ConditionalQA

8.3.3.1 Dataset

In the third experiment, we run TReasoner on ConditionalQA proposed in Chapter 7, which contains longer context (documents), more conditions and more complex relationship between conditions. Furthermore, the ConditionalQA dataset contains a mixture of "Yes"/"No" questions and questions with free-form answers.

The context in ConditionalQA is provided as a list of HTML elements. We treat each element at the leaf of the DOM tree as a condition c_i, and prepend all its parents (from its direct parent to the root) to get an expanded condition s_i.

Since we need the decoding module to generate answer spans, we initialize the model with T5, i.e. we use parameters from the encoder to initialize the entailment module, and use decoder to initialize the decoding module. The reasoning module is randomly initialized.

Results are evaluated with both EM/F1 on predicted answer spans, and Conditional EM/F1 on answer spans and unsatisfied conditions combined.

8.3.3.2 Baselines and Experimental Results

We compare TReasoner with several strong baselines, including ETC (in a pipeline) [5], DocHopper [148], and T5 (with FiD) [64]. The ETC pipeline first extracts possible answers from the context and then predict unsatisfied conditions independently. DocHopper is a multi-hop retrieval system

	Answer (w/ conds)	Conditions (P / R / F1)
T5 w/ FiD	3.2 / 4.6	98.3 / 2.6 / 2.7
TReasoner	**10.6 / 12.2**	**34.4 / 40.4 / 37.8**
T5 w/ FiD (conditional only)	6.8 / 7.4	12.8 / 63.0 / 21.3

Table 8.10: Ablated results on ConditionalQA in predicting unsatisfied conditions.

that iteratively retrieves evidence which contains answers and unsatisfied conditions. T5 (w/ FiD) is a encoder-decoder model. We train T5 (w/ FiD) to generate answers followed by a list of unsatisfied conditions ids. The experimental results are presented in Table 8.9. TReasoner significantly outperforms the baselines in predicting answers and jointly predicting answers and unsatisfied conditions – a relative improvement of 148% (Conditional) and 27.8% (Overall) in conditional F1 (F1 w/ conds).
Ablation: Condition Accuracy Since there's not a metric that only measures the quality of predicted conditions, we additionally report the F1 of the predicted unsatisfied conditions (Table 8.4). The best baseline models, T5 (w/ FiD), rarely predicts any conditions. Even though we train T5 (w/ FiD) only on the subset of questions that have conditional answers to force it predict unsatisfied conditions, its performance slightly improves but is still much lower than TReasoner by 16.5 points in condition F1.

8.4 Related Work

The task proposed by [24] is commonly referred to as *deductive reasoning* where all information required to find a definite answer is provided. Other models have been developed for deductive reasoning with symbolic rules [25, 27, 142, 129, 130]. Embedding-based methods [142, 129, 130] first convert symbolic facts and rules to embeddings and then apply neural network layers on top to softly predict answers. These models differ from our work in that the symbolic structure of the rules is typically known, whereas in our model it is implicit in a document.

Other recent work in deductive reasoning focused on tasks where rules and facts are expressed in natural language [155, 134, 24, 71]. Such tasks are more challenging because the model has to first understand the logic described in the natural language sentences before performing logical reasoning. Many of these models rely on rules that are produced by templates, or templated rules that have been paraphrased by crowd workers. In our work, the logical interactions analogous to these rules are implicit in real-world documents.

Different from most reasoning tasks, the task considered in this paper provides a list of conditions that, if true, would support an answer. Identifying such conditions is usually called *abductive* reasoning, as opposed to deductive reasoning. Very limited work has explored abductive reasoning for QA. Previous work [53, 54, 113] on the ShARC [136] dataset propose to solve this problem by predicting a special label "inquire" if there was not enough information to make a definite prediction. Specifically, EMT and DISCERN [53, 54] computed an entailment vector for each condition and performed a weighted sum of those vectors to predict the final answer. DGM [113] additionally introduced a GCN-based model to better represent the entailment vectors. Even though these models were able to predict the answer labels as "inquire" when there were unsatisfied conditions, none of them predict *which* conditions needed to be further satisfied, unlike our model. Our model is also more scalable than these, as it does not require concatenating a full context and a question.

8.5 Discussion

In this chapter, we proposed methods for the ConditionalQA task which focuses on new challenges in developing intelligent QA systems beyond relational following $X.\text{follow}(R)$. Specifically, we looked at questions that are under-specified, so answers are only correct under different conditions. The ConditionalQA dataset assumes that answers are constrained by a provided list of conditions, and that the conditions interact under complex logics, called condition groups. A challenging reasoning task is to check if any conditions in the logical groups have not yet been satisfied, given their interaction with others.

We tackled this challenge by proposing a novel model, T-Reasoner in this chapter. T-Reasoner modeled the condition verification process within logical groups with a Transformer model. The proposed method worked well in the reasoning task in both predicting the correct answers and accurately identifying conditions that are not yet satisfied.

The ConditionalQA task studied in this chapter assumes a list of candidate conditions are provided in the context. However, for questions in the open domain, candidate conditions are not known until retrieved from a open source of knowledge. In the next chapter, we study the problem of retrieving conditions and then identifying unsatisfied conditions for questions in open domain. We refer this task as Question Disambiguation.

Chapter 9

Disambiguating Open Domain Questions

9.1 Overview

Most previous research in open-domain QA focuses on finding the most probable answer to a question [69, 74]. However, many questions are ambiguous, and hence have multiple possible answers, each of which is correct under some interpretation of the question (i.e. conditions). For example, "Where is the home stadium of the Michigan Wolverines?" has different answers depending on whether one interprets the question as being about "football" or "basketball". A practical way to answer such ambiguous questions, for example, is to say the answer is "the Michigan Stadium" under the condition "football" or "Crisler Center" under the condition "basketball". Several datasets have been proposed recently that test models' ability to predict all possible answers [183, 43, 145] and find unique conditions for those answers [104, 141].

The tasks of finding all possible answers, and finding conditions for the answers, have been shown to be very challenging for existing models. Different from the ConditionalQA task proposed in the previous chapter, questions in open-domain do not have a list of candidate conditions to select from. Instead, answers and conditions should be retrieved from text corpus. This requires retrieving and aggregating diverse information [104, 141]. However, passages found by retrieval models such as BM25 and DPR [70] often lack diversity, i.e. the top-ranked passages commonly mention the same answers. While increasing the number of retrieved passages can increase recall of answers, it makes the answer prediction process more expensive—often quadratically more expensive.[1]

[1]The complexity of a Transformer-based reading comprehension model has the complexity of $O(N^2)$ where N is the number of tokens.

Rather than retrieving passages, an alternative way to implement a retrieve-and-read question-answering system is to extract information from a corpus, and retrieve the extracted information. Several recent systems have proposed extracting information as question and answer (QA) pairs [83, 19, 20, 173]. To answer open-domain questions, models retrieve generated QA pairs as evidence, rather than retrieving passages. Compared to passages, questions easier to retrieve, and are much shorter (usually 10 to 20 tokens, while passages commonly contain hundreds of tokens). Therefore, more questions can be retrieved, to increase the recall of answers.

In this chapter, we study the use of generated QA pairs in answering ambiguous questions. First, we construct a new database of questions from Wikipedia, named SIXPAQ (**S**ynthesized question **I**nterpretations to e**X**tend **P**robably **A**sked **Q**uestions), which contains 127 million questions and 137 million answers. Among the generated question, 5.8 million questions have more than one answer.[2] SIXPAQ doubled the size of the previous database of question (PAQ), and more importantly, improves the recall of answers by 30% on two open-domain QA datasets with ambiguous questions.[3] Second, we show that retrieving from SIXPAQ leads to more diverse answers and therefore higher answer recall on two benchmark QA datasets with ambiguous questions, AmbigQA [104] and WebQuestionsSP [181], with an improvement of up-to 6.7 point in recall@10. Third, we show that retrieving questions from SIXPAQ improves the performance of the ASQA task [141], a very challenging task of long-form question answering in summarizing and comparing answers of ambiguous questions. It improves the baselines by 1.9 points in DR and achieves a state-of-the-art on ASQA.

9.2 SIXPAQ

We construct SIXPAQ from Wikipedia. SIXPAQ contains 127 million questions and 137 million answers. The construction process involves three stages: answer detection, question generation, and answer verification. We discuss each stage in detail in this section and compare each stage to another popular database of questions, PAQ [83].

9.2.1 Source

We use the dump of Wikipedia preprocessed by DPR [70] as inputs to generate questions. In DPR's dump, Wikipedia articles are chunked into passages which contain 100 tokens. The preprocessed dump of Wikipedia contains 21 million passages. Since many of the question interpretations in ASQA involve less popular entities, we generate questions from all 21 million passages. In contrast PAQ generates from only 9.1 million passages (filtered by a learned question selection model).

[2]Some questions in SIXPAQ are semantically the same. Those questions are not counted in the 5.8 million questions.

[3]Please see §9.2.6 for more information on the evaluation of answer coverage.

9.2.2 Stage 1: Answer Detection

A Wikipedia passage usually contains multiple pieces of information and thus questions can be asked from different perspectives. We let the question generation process to be answer-conditioned to reflect this observation. During generation, possible answers are first detected and then questions are generated for every detected answer [83, 20].

We model the answer detection step as a sequence-to-sequence (seq2seq) generation problem with a T5 model [123].[4] We do not use a Named Entity Recognition (NER) model for answer prediction, as used in PAQ [83]. The input of the generation task is a Wikipedia passage and the output is a text span that is likely to be the answer to some questions. The answer detection model (with a pretrained T5) is finetuned on NQ [74]. We use beam search with beam size of 32 to generate multiple outputs, but filter these outputs with two heuristics. First, we require the generated spans to be sub-strings of the Wikipedia passage. Second, we merge spans which are identical after removing articles and punctuation. We end up with 283 million answers detected from 21 million Wikipedia passages.

9.2.3 Stage 2: Question Generation

Given answers detected from a Wikipedia passage, we then train a model to generate questions for the specific answers. Again, we finetune a T5 model for the question generation task. An input for question generation contains a passage and a target answer, e.g. "`answer`: Michigan Stadium `context`: The Michigan Stadium is the home stadium ...". An expected output should first repeat the target answer and then generate a question, e.g. "`answer`: Michigan Stadium `question:` Where is the home stadium ...". In preliminary experiments, encouraging the model to first repeat the answers generally improves the quality of questions by making generated questions more specific to target answers.

We use question and answer (QA) pairs from AmbigQA [104] to train the question generation task. Questions in AmbigQA originate from NQ but are revised by adding additional answer-specific information from passages to questions to remove ambiguity. This departs from most prior QG work, which generally trains models on SQuAD [125] or NQ [74]. While AmbigQA is smaller than either of these datasets, the questions are more natural than SQuAD (where questions were *formulated* by crowdworkers looking at the passage) and better-grounded than NQ (since questions are *revised* by crowdworkers looking at the passaege), which seems to be a happy medium in producing natural questions with minimal hallucination[10].

We use greedy search at inference time to generate one question per answer. While PAQ used beam search (with a beam size of 4) to increase the number of generated questions [83], we find that questions generated from beam search are often very similar to each other. Having near-duplicate

[4]We use the pretrained T5-11B model for all subtasks in constructing SIXPAQ to ensure the high quality of data.

questions makes the database larger but does not increase the utility of the database for most downstream tasks.

9.2.4 Stage 3: Answer Verification

Questions generated from the previous step are sometimes invalid – i.e. some questions may not be answerable from the provided passages, or the correct answers to the generated questions are different from the answers from which the questions are generated. Therefore, an additional answer verification step is needed.

We train a question answering (QA) model in the reading comprehension setting to perform the answer verification task. In particular, the model takes a passage and a generated question to predict an answer. If an answer does not exist in the passage, the model should predict "not answerable". We finetune a T5 model on SQuAD v2 [124], a reading comprehension dataset which contains unanswerable questions. During verification, we drop questions if their predicted answers are "not answerable" or different from their original answers.[5] After the verification step, 156 million questions are left.

The question generation process often produces questions that are ambiguous in an open-book setting, i.e. they have multiple answers. This is expected since many NQ questions are themselves ambiguous [104] when considered carefully. In PAQ, questions that have multiple open-book answers are filtered by running an open-book QA system and discarding questions with an open-book answer different from the one used for generation. This has several disadvantages: it is expensive, since open-book QA is expensive to run; it is relatively noisier than our proposed QA-based filter, since open-book QA is less accurate than machine-reading style QA; and it filters out more than 76% of the generated questions, and it is not actually appropriate for some downstream applications (such as the ones considered in §9.3.2), where questions are used in conjunction with the passages from which they were generated.

9.2.5 Statistics

We merge the question and answer pairs by merging pars with identical questions and end up with 127 million unique questions, among which 14.3 million questions have more than one answer mention, and 5.8 million questions have more than one unique answer.[6]

9.2.6 Coverage of Answers

We measure the coverage of answers of SIXPAQ on two benchmark datasets of ambiguous questions with multiple answers, AmbigQA [104] and WebQuestionsSP (WebQSP) [181]. We evaluate the

[5]We normalize the original and predicted answers before comparison using scripts provided by [125].

[6]Merging is performed by word matching, even though many questions are semantically same.

	AmbigQA		WebQSP		NQ	
	Ans	QA	Ans	QA	Ans	QA
PAQ	83.0	45.1	81.9	48.5	89.9	70.3
SIXPAQ	89.2	79.3	89.9	82.5	92.0	85.6

Table 9.1: Recall of answers of SIXPAQ on AmbigQA (dev), WebQuestionsSP (test) and NQ (dev). "Ans" represents the recall of answers by matching answer strings (after normalization steps). "QA" is evaluated on 100 randomly selected question by human annotators by checking whether questions of the matched answers in the database are actually relevant.

exact match of answer spans after normalizing the answers. The recall of answers is shown in Table 9.1. The recall of SIXPAQ is 6.2% higher than PAQ on AmbigQA and 8% higher on WebQuestionsSP (WebQSP). In addition, we measure the coverage of answers from NQ as a reference for non-ambiguous questions. SIXPAQ improves the coverage of answers by 2.1%.

However, QA pairs in the databases that contain the answers are sometimes irrelevant to the questions asked. For example, ("What is the largest stadium in the US?", "the Michigan Stadium") may not used to answer the question "Where is the home stadium of the Michigan Wolverines?" Therefore, the recall of answers is insufficient. To measure the true recall, we manually examine 100 questions to check whether the questions are relevant. We denote this metric as "QA" in Table 9.1. The true recall of QA pairs in SIXPAQ is 30% higher than PAQ on both AmbigQA and WebQuestionsSP (WebQSP) datasets. For non-ambiguous questions (NQ), the recall improves by 15.3%.

9.3 SIXPAQ for Answering Ambiguous Questions

In this section, we propose methods which consume SIXPAQ for two important tasks in answering ambiguous questions, including retrieving passages with diverse answers, and generating long outputs to discuss the difference between multiple answers for question disambiguation.

9.3.1 Retrieval of Diverse Passages

One common problem with existing retrieval models [70, 112] for open-domain QA is the lack of diversity of the retrieved results [103], i.e. only a subset of correct answers are obtained from the top-retrieved passages. This restricts models' performance in predicting multiple answers and in comparing different answers. We show that we can get more diverse passages *indirectly*, by first retrieving similar generated questions q' given a input question x, and then using as the final retrievals the passages from which the q''s here generated.

Retrieving questions q' given a question x is analogous to retrieving passages from text corpora, so any existing retrieval method can be applied. In this work, we use a sparse retrieval model, BM25,

Answers	Context	Revision 1	Revision2
Michigan Stadium	Michigan Stadium, nicknamed "The Big House", is the home stadium for the University of Michigan ***men's football team*** (Michigan Wolverines) in Ann Arbor, Michigan. Michigan Stadium was ***built in 1927***, and it is the largest stadium in the US...	Where is the home stadium of Michigan Wolverines ***men's football team***?	Where is the home stadium of Michigan Wolverines men's football team ***built in 1927***?
Crisler Center	Crisler Center (formerly known as the University Events Building and Crisler Arena) is an ***indoor*** arena located in Ann Arbor, Michigan. It is the home arena for the Michigan Wolverines ***men's and women's basketball***...	Where is the ***indoor*** home stadium of Michigan Wolverines?	Where is the indoor home stadium of Michigan Wolverines ***men's and women's basketball?***

Table 9.2: Examples of revisions of a question "Where is the home stadium of Michigan Wolverines?" (not an exclusive list). Depending on different answers, questions are revised at each revision step to add additional answer-specific information. We consider question revision as a question expansion step which moves information from passages to questions.

and a state-of-the-art dense retrieval model, GTR [112]. GTR was originally designed for passage retrieval but the query encoder and passage encoder in GTR share parameters, so, we can directly use it to encode and retrieve questions as well. We use GTR-large and finetune the checkpoint of GTR on NQ-open [74] in our experiments.

Questions retrieved from SIXPAQ are then mapped to passages where those questions were generated. With an input question x, the score for a passage p_i is

$$s(x, p_i) = \max\nolimits_{q' \in \text{GEN}(p_i)} f(x, q') \tag{9.1}$$

where $\text{GEN}(p_i)$ is the set of questions generated from the passage p_i and $f(x, q')$ is the retrieval score of the question $q' \in \text{GEN}(p)$ from BM25 or GTR. We denote this method as "max" in the our experiments (§9.4.1).

In addition, we propose another simple heuristic to map questions to passages. It returns passages from which the most top-k retrieved questions are generated. We use $k = 50$ in our experiments. This method is denoted as "count" in our experiments (§9.4.1).

$$s_c(x, p_i) = |\{\text{GEN}(p_i) \cap \text{argmax}_{k,q'} f(x, q')\}| \tag{9.2}$$

9.3.2 Ambiguous QA with Long Outputs

In the second task, we investigate the challenging task of answering ambiguous questions with long outputs summarizing multiple answers to the questions. For ambiguous questions that have different answers, one practical way to answer such questions is to specify under what conditions answers are correct. For example, for the question “Where is the home stadium of Michigan Wolverines?”, in addition to predicting a list of answers, {“Crisler Center”, “Michigan Stadium”, ...}, a QA system should clarify that “Crisler Center” is the home stadium of the Michigan basketball team while the “Michigan Stadium” is the home of the football team. The ASQA task proposed to answer ambiguous questions by summarizing the multiple answers into short paragraphs, e.g. “The home stadium of Michigan Wolverines men’s football is the Michigan Stadium, while the stadium of its men’s basketball team is the Crisler Center. Crisler Center is also the home stadium for Michigan Wolverines women’s basketball”.

Previous models simply retrieve passages from a text corpus and generate answers from the retrieved results. However, the retrieved passages are usually long and contain information irrelevant to the answers. We propose to retrieve questions from SIXPAQ as a concise representation of question-specific information from passages. We additionally propose a question revision step which operates on the retrieved questions to include more detailed information for the disambiguation task.

Question Revision While the questions in SIXPAQ are fairly unambiguous, we also explored approaches to make the questions include more information from the passages from which they were generated. We trained a sequence-to-sequence model to extract answer-specific information from passages where SIXPAQ questions are generated and rewrite the questions to include such information. Examples of questions before and after revision are shown in Table 9.2: e.g., the model locates the information “men’s football” from the context “... is the home stadium for the University of Michigan men’s football team (Michigan Wolverines) in Ann Arbor ...” and adds it to the initial question. The revised question, “Where is the home stadium of the Michigan Wolverines men’s football team built in 1927?”, contains information {“men’s football”, “built in 1927”} that is specific to the answer “Michigan Stadium”. Compared to passages with hundreds of tokens, the revised questions are more concise in capturing information that is specific to answers.

We finetune a T5 model to perform the question revision task. The T5 model is trained with data provided as auxiliary information in the ASQA dataset [141], which contains revised questions for different answers a_i and passages p_i provided to human annotators to write the revised questions q'_i.[7] The question revision model takes an ambiguous question q, an answer a_i, and a passage p_i to generate a revised question q'_i. The input and output of the model are shown below.

$$\begin{aligned} \text{input} =&\ \text{question: } q + \text{answer: } a_i + \text{passage: } p_i \\ \text{output} =&\ \text{answer: } a_i + \text{revised: } q'_i \end{aligned}$$

At inference time, we repeat the revision process k times to increase the amount of information added to original questions. In the experiments, we use $k = 2$ because we observe the model tends to generate identical questions if $k > 2$. The revised questions have an average length of 14.5 compared

[7]The revised questions originate from the AmbigQA [104] dataset. ASQA conducted additional annotation and included more auxiliary information, such as passages for question revision.

to original questions, which average 9.0 words long.

Long-form Answer Generation After revision of the top-retrieved SIXPAQ questions, we perform a generation task, to summarize the differences between multiple answers of the ambiguous questions. In addition to the revised questions from SIXPAQ, we also retrieve a few passages from Wikipedia for generating long-form answers. We find retrieving passages is necessary for ASQA—perhaps because during annotation annotators were encouraged to include background information in the long-form answers. Such information is not specific to any answer, so merely retrieving from SIXPAQ does not provide the necessary information. To mitigate this problem, we follow the baseline to also include top n passages retrieved by JPR [103] from Wikipedia [141].[8] The inputs to the generation model are thus a concatenation of the original question q, answers and retrieved questions $\{(a_i, q_i')\}$, and retrieved passages $\{p_j\}$. The target outputs are the long answers provided in ASQA. We finetune a T5-large model [123] for this generation task.

$$\text{input} = \text{ question: } q + \text{conditions: } a_1,\, q_1',\, ... + \text{passages: } p_1,\, ...$$

9.4 Experiments

In this section, we discuss the experimental results for retrieving diverse passages and generating long-form answers for ambiguous questions.

9.4.1 Retrieval of Diverse Passages

Dataset We use AmbigQA [104] and WebQuestionsSP [181] in our experiments. AmbigQA is an open-domain QA dataset derived from NQ [74] which contains questions that are ambiguous and thus have multiple possible answers. WebQuestionsSP (WebQSP) [181] is another dataset which contains open-domain questions asked by web users, and a subset of the questions have multiple answers. We only evaluate on multi-answer questions (in both datasets) in this experiment: 1172 questions in the AmbigQA dev set and 809 questions in the WebQuestionsSP test set have multiple answers.[9]

Evaluation To measure the diversity of retrieval, we evaluate models' performance as the recall of answers. Similar to traditional passage-level retrieval models [70], the recall is measured as the percentage of correct answers that are mentioned in the retrieved passages.

Results Experimental results are presented in Table 9.3. Numbers in the first block (passage-based retrieval) show the performance of baseline models, BM25, DPR [70] and GTR [112], in directly retrieving passages from Wikipedia. DPR is another popular dense retrieval method with separate query and candidate encoders, and trained with hard negatives. We re-run the DPR open-sourced

[8]JPR is an auto-regressive reranking model aiming for increasing the diversity of retrieved passages.

[9]We consider questions have multiple answers if at least one of the annotators find multiple answers.

	AmbigQA		WebQSP	
k	5	10	5	10
passage - based retrieval				
Wikipedia + BM25	35.5	44.4	23.4	32.0
Wikipedia + DPR	50.7	57.7	36.2	43.0
Wikipedia + GTR	55.0	61.9	38.8	46.3
question - based retrieval				
PAQ + BM25 (max)	36.9	43.6	26.7	32.7
PAQ + GTR (max)	43.7	51.3	33.2	39.6
SIXPAQ + BM25 (max)	35.5	45.8	24.8	34.4
SIXPAQ + GTR (max)	53.6	60.4	45.3	51.8
SIXPAQ + BM25 (count)	36.4	47.0	25.7	35.8
SIXPAQ + GTR (count)	**55.9**	**63.4**	**46.7**	**53.0**

Table 9.3: Recall@k of retrieving diverse answers for multi-answer questions in AmbigQA and WebQuestionsSP. Experiments with "max" (Eq. 9.1) and "count" (Eq. 9.2) use different methods to map top-retrieved questions to passages. We use GTR-large in the baselines and our method.

code and evaluate the retrieved results. We also run GTR-large for passage retrieval on both datasets. For question-based retrieval, we apply the proposed method on both PAQ [83] and our SIXPAQ dataset. Again, we use GTR-large in our method. Experiments with (max) refer to the method of directly mapping top-retrieved questions to passages where they are generated (Eq. 9.1), while ones with (count) refer to returning passages where most top-retrieved questions are generated (Eq. 9.2). Compared to passage-based retrieval methods, indirect retrieval with SIXPAQ yields better performance than using BM25 or GTR. In particular, the recall@10 with BM25 improves from 44.4 to 45.8 on AmbigQA and from 32.0 to 34.4 on WebQuestionsSP. The performance with GTR is also better with SIXPAQ. On AmbigQA, the recall@10 improves 61.9 to 63.4. More improvement comes on WebQuestionsSP with an increase from 46.3 to 53.0. We conjecture that the improvement with GTR is less significant on AmbigQA because GTR is pretrained on NQ, which is a superset of AmbigQA.

9.4.2 Ambiguous QA with Long Outputs

Dataset The ASQA dataset contains 4353 train and 948 dev examples. Each example contains an ambiguous question, a list of disambiguated questions and answers (short text spans), and a long-form answer which discusses the difference between short answers. Due to the high variance of long-form answers, each example in ASQA was annotated by two human annotators and the better score among the two annotations is recorded. The average length of answers is 65.0 white-space split tokens. Each question has an average of 3.4 different short answers.

Evaluation In ASQA, predicted outputs are evaluated from a few different perspectives. First, as a long output prediction task, it evaluates the similarity of predicted outputs with reference outputs with ROUGE-L scores. Second, it measures the recall of answers in the predicted outputs (named

	LEN (words)	ROUGE-L	STR-EM	DISAMBIG-F1	DR
DPR @ 1†	99.9	31.1	30.1	16.7	22.8
JPR @ 1†	196.8	27.9	45.0	25.8	26.9
T5 (1 passage)†	**63.0**	40.3	33.6	21.2	29.2
T5 (3 passages)†	71.1	42.7	39.9	25.1	32.7
T5 (5 passages)†	71.6	43.0	41.0	26.4	33.7
T5 (5 passages) *	68.1	43.0	40.1	26.4	33.7
T5 (7 passages) *	69.3	43.0	39.5	25.5	33.1
T5 (10 passages) *	68.9	43.0	39.2	25.9	33.2
ours					
T5 (1 passage + 10 questions)	58.3	41.6	39.4	26.5	33.2
T5 (2 passages + 10 questions)	62.0	42.9	41.8	28.0	34.6
T5 (3 passages + 10 questions)	63.3	42.9	41.5	28.2	34.8
T5 (5 passages + 10 questions)	63.5	**43.8**	**42.4**	**28.9**	**35.6**
T5 (oracle)	82.6	46.6	88.7	59.2	52.5
Human	64.8	49.4	98.4	77.4	61.8

Table 9.4: Performance of long-form answer generation with retrieved answers and passages. All models are finetuned on T5-large. † Numbers copied from the baseline [141]. * Numbers obtained by re-implementing the baseline models.

STR-EM)–all possible answers must be mentioned in the predicted output in order to receive full STR-EM scores. Third, it introduces a learned metric DISAMBIG-F1, with the goal of measuring whether the disambiguating information about answers in the outputs is accurate. To compute DISAMBIG-F1, the ASQA dataset uses a learned a QA model to find the answers of a sequence of disambiguated questions (provided by annotators) from the generate output. The output will receive a full DISAMBIG-F1 score if all predicted answers from the QA model match the oracle answers of the disambiguated questions. Finally, they compute an overall score, DR, as the geometric mean of ROUGE-L and DISAMBIG-F1. In addition, LEN (words) measures the average length of outputs in terms of words. Shorter outputs with higher DR scores are preferred.

Results We evaluate the finetuned T5-large model on the quality of predicted long-form answers. To show the effectiveness of the retrieved questions and answers from SIXPAQ, we compare to the outputs generated from retrieved passages only.

Results are presented in Table 9.4. The first group of results (DPR@1 and JPR@1) means we directly return the top 1 passage retrieved by DPR and JPR. The second group of results, e.g. T5 (5 passages), shows the performance of directly generating outputs with the top 5 retrieved passages with T5-large. Both groups of numbers are copied from the original paper by [141].

To check whether higher recall from more retrieved passages leads to better outputs, we re-implement the baseline model to run it on more retrieved passages. The results with 5 passages from our implementation matches the numbers reported in the original paper [141]. However, as shown in Table 9.4, as the number of passages increases, both STR-EM and DISAMBIG-F1 drops

(40.1 to 39.2, 26.4 to 25.9).

In the third group of experiments, we retrieve questions from SIXPAQ and add top 10 answers with their conditions to the input. Without changing the model, the performance of long-answer generation with information retrieved from SIXPAQ increases by 0.8 in ROUGE-L and 2.5 in DISAMBIG-F1. In addition, the output length of the model with information from SIXPAQ is also $\sim$10% shorter than the baseline but covers more answers in its outputs.

9.4.3 Ablations

Recall of Answers In the first ablation experiment, we justify the claim that retrieving questions from SIXPAQ can improve the diversity of answers. We report the recall of answers with and without questions retrieved from SIXPAQ in terms of numbers of tokens (see Figure 9.1). With 10 questions from SIXPAQ added to 5 passages, the recall of answers improve from 65.5 to 71.1, which leads to around 2 additional points in the final DR metric. From another perspective, with as few as 10 questions and 2 passages, the recall becomes comparable to 5 passages (66.1 vs. 66.5). Furthermore, the total length of 10 answers plus 2 passages is 43% less than 5 passages (574.5 vs. 1008.8), since information from the revised questions are more concise. This eventually leads to 1 point of increase in the final DR metric (34.6 vs. 33.7) as shown in Table 9.4.

Accuracy vs. Input Length. We additionally compare the performance of models in DISAMBIG-F1 under different numbers of tokens. Results are shown in Figure 9.1 (right). With SIXPAQ questions, models get better DISAMBIG-F1 performance with shorter inputs. The DISAMBIG-F1 with 10 questions and 3 passages (775.6 tokens) is 28.2, better than the DISAMBIG-F1 of 27.2 with 5 questions and 4 passages (899.9 tokens) and DISAMBIG-F1 of 26.4 with 0 question and 5 passages (1008.8 tokens).

Revised vs. Unrevised Questions. We further ablate our model to investigate the importance of question revision step proposed in §9.3.2. The results are shown in Table 9.1 (right). The model's performance with 10 revised question is consistently better than with unrevised questions.

Question Disambiguation In the next ablation experiment, we experiment with a more straightforward task to investigate whether the additional information retrieved from SIXPAQ help disambiguating questions. Here, we study a question revision sub-task, using auxiliary data provided in ASQA.[10] In this sub-task, the model should revise an ambiguous question using information provided a passage such that it can differentiate the provided answer with others. The task is similar to revising questions when constructing SIXPAQ (§9.3.2), except that the additional information added to the revised question should be contrastive, i.e. it should differentiate its answer with others possible answers to the ambiguous questions. For example, to differentiate the answer "the Michigan Stadium" and "Crisler Center", one should provide the additional information "*basketball team*" vs. "*football team*", but not "*built in 1927*" vs. "*basketball team*".

[10]This information is also available in the question revision sub-task in AmbigQA [104].

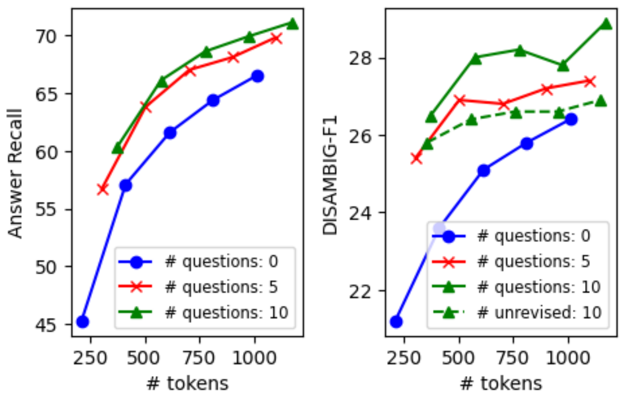


Figure 9.1: Left: Recall of answers from the retrieved results with varying number of passages. Right: DISAMBIG-F1 of predicted long answers with varying number passages. Both figures show that inputs with more questions yield better outputs under certain number of tokens, in both answer recall and DISAMBIG-F1.

A naive model simply takes an ambiguous question, an answer, and the provided passage as input to predict the revised question. We instead retrieve similar questions, along with their answers and conditions from SIXPAQ, and augment the provided passage with the retrieved information, similar to §9.3.2. The additional information should provide background knowledge for models to determine how to revise the question. Again, we finetune a T5 model for this question revision task. The model's output is measured by a metric, "EDIT-F1", proposed by [104], to only evaluate the edits made in the revised questions.[11]

The results are shown in Table 9.5. In addition to the naive baseline of only taking the provided passage as input (passage-only), we experiment with a few other options that can potentially improve the coverage of information for the question revision task. First, we retrieve the top 1 passage from Wikipedia (top-1 Wikipedia). Second, we expand the provided passage with preceding and following tokens to double the length of inputs (adjacent context). Third, we deliberately add a passage which contains different answers of the same question (contrasting passage).[12] Results in Table 9.5 shows that adding questions retrieved from SIXPAQ is the most effective method in revising questions, and therefore justify our claim that questions from SIXPAQ are concise and can provide sufficient background information for models to differentiate answers.

9.5 Related Work

In previous work, [83] constructed a database of 67M probably asked questions (PAQ) from Wikipedia and proposed to answer open-domain questions by retrieving similar questions from PAQ. Later work

[11]Please refer to [104] for more information on "EDIT-F1".
[12]Also available as auxiliary information in ASQA.

	EDIT-F1
oracle passage	22.4
+ adjacent context	22.6
+ top-1 Wikipedia	21.6
+ contrasting passage	23.0
+ top-5 SIXPAQ questions	24.8
+ top-10 SIXPAQ questions	**25.6**

Table 9.5: Results on the question revision task of AmbigQA. We compare different approaches to increase the coverage of information in the inputs.

[20, 173] proposed alternative approaches to using databases of QA pairs in QA systems, proposing pre-training methods that are more statistically-efficient [20] or more computationally efficient [173], and developing multi-hop QA models [20] that retrieve multiple times from a QA memory. However, prior QA-memory based models [83, 20, 173] focused on the traditional QA setting of unambiguous questions with unique answers. This leads to many differences in emphasis: for example, the PAQ dataset purposely removes generated questions with multiple answers.

Another efficient way to store world knowledge is to build knowledge bases (KB) or knowledge graphs (KG), such as Freebase and Wikidata, where information is stored with entities and relations as triples, e.g. ("Charles Darwin", "author of", "On the Origin of Species"). Knowledge bases have been commonly used in many knowledge intensive tasks due to its structured nature [149, 102]. Knowledge bases, however, lack the expressiveness in representing complex relationships that involve multiple pieces of information, and often do not contain information in a format that naturally reflects users' questions.

To resolve this problem, [46, 151] proposed to construct virtual knowledge bases that are not restricted to pre-defined vocabularies of entities and relations. [46] proposed to store entity-centric information as vectors and build a large database of vectors by iterating through passages in Wikipedia. [151] encoded pair-wise relationship between entities as vectors. Both methods support a similar reasoning process as regular knowledge bases. Others have argued for use of entity-linked QA pairs as a formalism for storing knowledge, as a representation that is more aligned with users' information needs, but still are closely related to traditional AI representations like KBs and KGs [19].

Recent interest in QA for ambiguous questions poses new challenges for retrieving and representing knowledge. In such datasets, models are required to not only find one of the correct answers, but also comply with additional requirements associated with the need to choose between multiple answers. For example, Temp-LAMA [43] requires models to answer time-sensitive questions under given time constraints; [183] contains questions with geographic and temporal constraints; ConditionalQA [145] contains constraints based on user scenarios; and ROMQA [185] requires QA subject to different combinations of constraints.

This work builds especially on the AmbigQA dataset [104], which contains NQ questions that have multiple interpretations, and the ASQA dataset [141]. The ASQA dataset contains ambiguous questions with long-form answers, where ther answers explain in text what the alternative interpretations of the original question are, and what the answer is for each interpretation.

9.6 Discussion

In this chapter, we discussed a novel method of disambiguating questions in the open domain, i.e. answers and conditions must be retrieved from a text corpus. We tackle two challenges encountered by existing QA models in answering ambiguous questions.

- Previous methods which first retrieve passages from text corpora then extract answers lack diversity in the retrieved information. In this work, we proposed to instead retrieve from a database of probably asked questions, SIXPAQ, constructed from Wikipedia. SIXPAQ contains 127 million questions which covers more than 80% of questions in three benchmark QA datasets. Experiments show that retrieving from SIXPAQ is easier than from Wikipedia, leading to higher answer coverage and diversity.
- We additionally propose a novel question revision task to create a list of conditions for the ambiguous question from similar questions retrieved from SIXPAQ and passages associated with the retrieved questions. The list of conditions can be directly returned as conditions for answers of the ambiguous questions, or can be combined with passages to predict novel conditions.

Our proposed method significantly improves the performance of baseline models in answer recall and in question disambiguation. However, the performance of our model is still limited, i.e. more than 20 points lower than human performance under several metrics. We argue disambiguating questions and answering ambiguous questions with conditions are still challenging tasks.

Recent Large Language Models (LLMs), e.g. LaMDA and GPT-4, also face similar challenges in answering ambiguous questions. They often predict the best known answers rather than all possible answers under different interpretations of the questions. For example, the output from one of the famous LLMs to the example question above, “The home stadium of the Michigan Wolverines is Michigan Stadium, also known as ‘The Big House’. It is located in Ann Arbor, Michigan”, assumes the sport is football – one of the most famous sports that the Michigan Wolverines play. Training LLMs to predict diverse answers while keep faithful to existing knowledge is a challenging but interesting research problem.

Chapter 10

Conclusions

In this thesis, we studied reasoning tasks commonly seen in Question Answering, a challenging task in NLP research. We proposed state-of-the-art methods to model the reasoning procedures in Question Answering. In particular, we started by discussing two query language methods, NQL and EmQL. Both methods worked on symbolic knowledge bases and can accurately perform several reasoning tasks, such as relation following, intersection, union, negation, etc. However, NQL and EmQL were restricted by the coverage of information in symbolic knowledge bases. To resolve this problem, we proposed OPQL which operated on virtual knowledge bases (VKBs) constructed from text corpora. Next, we showed that the proposed query languages can be easily injected to language models (LMs), e.g. FILM and OPQL-LM, improving LM's awareness of factual knowledge as well as enabling modifying learned knowledge at inference time. So far, the reasoning tasks we studied fall into the category of deductive reasoning where complete information of questions is provided to obtain deterministic answers. In the last two chapters, we instead studied abductive reasoning where questions are incomplete and answers only apply under certain conditions. We worked on two settings in abductive reasoning: (1) candidate conditions are provided in the context, and (2) conditions need to be retrieved, e.g. from text corpora. We proposed a challenging dataset, ConditionalQA, and two state-of-the-art methods, T-Reasoner and SIXPAQ, to show challenges and advances in abductive reasoning.

10.1 Summary of Contributions

In this thesis, we studied the reasoning tasks from a few perspectives:

- *What types of reasoning?* We focused on a few types of reasoning tasks commonly seen in Question Answering, including relation following, intersection, union, negation, etc. We proposed a collection of methods, such as NQL, EmQL, and OPQL. In the later parts of the thesis, we extended our research to abductive reasoning, and proposed the ConditionalQA dataset and T-Reasoner and SIXPAQ-based methods.

- *Which knowledge source?* We worked on three popular sources: text corpora, knowledge bases, and databases of probably asked questions. Knowledge bases contains symbolic data and are thus easy to use for reasoning tasks. Methods, such as NQL and EmQL, were proposed over symbolic KBs. However, KBs are always incomplete, bringing limits to questions that can be answered. Text corpora, on the other hand, covers enormous information about the world. Popular text corpora are news articles, books, Wikis, etc. However, textual data is hard to interpret due to its diversity in syntactics and semantics. Therefore, we proposed virtual KB methods, such as OPQL, and text plus KB methods, such as FILM and OPQL-LM to accurately use text to answer questions. Finally, we proposed SIXPAQ, a database of 127M probably asked questions generated from Wikipedia. SIXPAQ is considered another rich source of knowledge represented as question and answer pairs (QA pairs).

- *Deductive or abductive?* Many previous QA tasks focused deductive reasoning, i.e. questions are well defined such that unique answers exist. We showed models like NQL can accurately model such reasoning tasks. Then, we focused on another type of reasoning, abductive reasoning, where questions must be answered with conditions. We proposed T-Reasoner for the abductive reasoning tasks. For abductive reasoning tasks in open-domain, we propose methods consuming a database of questions, SIXPAQ, to obtain conditions for question disambiguation.

10.2 Future Work

This thesis has studied several important aspects of reasoning for answering complex questions. While we have made much progress, we realize more effort is needed. We highlight a few limitations to address in future research.

Other Types of Reasoning in Question Answering

A few other challenging reasoning tasks have not been covered in this thesis. We will leave them as open questions for future research.

- *Comparative Reasoning.* In comparative reasoning, models have to compare two or more entities from specified perspectives. For example, "Does monkeys have longer lifespan than butterflies?". It is challenging because it not only need to find the values of the requested attributes, for example, the lifespan of monkeys and butterflies, but also compare the values, e.g. "10-30 years" vs. "15-29 days". The comparison involves many other challenges, such as understanding ranges beyond single values and interpreting units.

- *Superlative Reasoning.* Similar to comparative reasoning, superlative reasoning also require models to compare values to find extrema, for example, "what monkeys have the longest lifespan?" It is more challenging than comparative reasoning, however, because the comparison needs to be performed over multiple values, usually more than two. Furthermore, models need to find an exclusive list of candidates for comparison, e.g. all species of monkeys, before performing the comparison.

- *Numerical Reasoning.* As discussed above, numerical reasoning is already a crucial component in comparative and superlative reasoning. It is also important in broader tasks when calculations are needed, for example, "time of exercise to burn 2000 calories" with knowing the amount of calories burn in 1 hour.

Question Answering in Other Settings

In this thesis, we mainly focused on answering questions about factual knowledge in an open-domain setting. However, questions can be asked in different circumstances. We did not cover those cases in this thesis, but we would like to point them out for future research.

- *Conversational Question Answering.* People ask and answer questions during conversations. Answering questions in conversations is different because they have rich contextual information in conversation histories. Models have to capture such information when making predictions. Sometimes, the dependency can be long. Conversations are also difficult because of interruptions, correferences, pragmatic reasoning in spoken languages, etc.

- *Personalized Question Answering.* Some questions from users require personalized answers. It requires deep understanding of the users' interests and needs from a large amount of data about the users, such as their past questions, search histories, social media activities, etc. Furthermore, users' interests and needs change over time and the system needs to rapidly adapt to these changes.

- *Domain-specific Question Answering.* Many questions are raised in profession domains, such as medical and financial domains. Domain-specific questions are challenging because of the technical languages used in questions and context can be hard to understand. Furthermore, training data for domain-specific questions are limited, making it a more challenging task than answering factual questions. So far, limited research has been conducted in domain-specific questions. It is not clear whether models which work well for factual questions also apply to specialized domains.

Reasoning beyond Question Answering

Methods proposed in this thesis mainly focus on Question Answering. Other NLP tasks also require reasoning abilities to perform them well. For example, in fact checking, models need to verify if claims can be derived from existing facts. Some claims need to be supported by several facts combined. Therefore, models need to reason over multiple facts to determine their correctness. For another example, in commonsense reasoning, models should learn the relationship between objects using commonsense knowledge that all humans are expected to know. Commonsense reasoning is challenging because some commonsense knowledge is hard to be expressed with entities and relations, and some commonsense knowledge is rarely expressed in text, e.g. “lawn is wet in rainy days”. Reasoning steps associated with these tasks are different, but still challenging. We expect more research in the future to develop more accurate and generalized models to perform more different reasoning tasks and to improve the performance of NLP as a whole.

Reasoning in Large Language Models

Large language models (LLMs) such as LaMDA and GPT-4 have been recently developed and have showed outstanding ability in generating text that is coherent and contextually appropriate. LLMs are based on statistical modeling. They are trained on vast amounts of text from a variety of sources to learn patterns and relationships between words, phrases, and concepts. Therefore, LLMs are good at tasks that requires contextual understanding, such as summarization and reading comprehension. However, since LLMs are based on statistical modeling, it is challenging for LLMs to follow rules and explicit representations of knowledge to perform precise reasoning steps. In contrast, symbolic methods can be used to precisely perform logical reasoning, for example in symbolic question answering to derive answers, but lack the ability in resolving ambiguity or understanding contextual information.

One can combine symbolic reasoning methods with LLMs to build more powerful machine learning models. In this thesis, we presented a few methods to perform symbolic reasoning in an embedding space and to incorporate symbolic reasoning into language models to answer factual questions. Better methods can be proposed in the future to create reasoning systems that are more accurate, robust, and adaptable to a wide range of tasks in the real world.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Wasi Ahmad, Jianfeng Chi, Yuan Tian, and Kai-Wei Chang. PolicyQA: A reading comprehension dataset for privacy policies. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 743–749, Online, November 2020. Association for Computational Linguistics.

[3] Sungjin Ahn, Heeyoul Choi, Tanel Pärnamaa, and Yoshua Bengio. A neural knowledge language model. *arXiv preprint arXiv:1608.00318*, 2016.

[4] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. Etc: Encoding long and structured inputs in transformers, 2020.

[5] Joshua Ainslie, Santiago Ontañón, Chris Alberti, Philip Pham, Anirudh Ravula, and Sumit Sanghai. ETC: encoding long and structured data in transformers. *CoRR*, abs/2004.08483, 2020.

[6] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016.

[7] Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. Leveraging linguistic structure for open domain information extraction. 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 344–354, 2015.

[8] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[9] Livio Baldini Soares, Nicholas FitzGerald, Jeffrey Ling, and Tom Kwiatkowski. Matching the blanks: Distributional similarity for relation learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2895–2905, Florence, Italy, July 2019. Association for Computational Linguistics.

[10] Saptarashmi Bandyopadhyay, Shraman Pal, Hao Zou, Abhranil Chandra, and Jordan Boyd-Graber. Improving question answering with generation of nq-like questions. *arXiv preprint arXiv:2210.06599*, 2022.

[11] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[12] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, 2013.

[13] Tarek R Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C Lamb, Daniel Lowd, Priscila Machado Vieira Lima, et al. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*, 2017.

[14] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 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. AcM, 2008.

[15] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[16] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[17] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, 2017.

[18] Qian Chen, Xiaodan Zhu, Zhen-Hua Ling, Diana Inkpen, and Si Wei. Neural natural language inference models enhanced with external knowledge. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2406–2417, 2018.

[19] Wenhu Chen, William W Cohen, Michiel De Jong, Nitish Gupta, Alessandro Presta, Pat Verga, and John Wieting. Qa is the new kr: Question-answer pairs as knowledge bases. *arXiv preprint arXiv:2207.00630*, 2022.

[20] Wenhu Chen, Pat Verga, Michiel de Jong, John Wieting, and William Cohen. Augmenting pre-trained language models with qa-memory for open-domain question answering. *arXiv preprint arXiv:2204.04581*, 2022.

[21] Wenhu Chen, Xinyi Wang, and William Yang Wang. A dataset for answering time-sensitive questions, 2021.

[22] Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Wang. Hybridqa: A dataset of multi-hop question answering over tabular and textual data, 2021.

[23] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555, 2020.

[24] Peter Clark, Oyvind Tafjord, and Kyle Richardson. Transformers as soft reasoners over language. *CoRR*, abs/2002.05867, 2020.

[25] William W Cohen. Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*, 2016.

[26] William W. Cohen, Matthew Siegler, and R. Alex Hofer. Neural query language: A knowledge base query language for Tensorflow. *CoRR*, abs/1905.06209, 2019.

[27] William W Cohen, Haitian Sun, R Alex Hofer, and Matthew Siegler. Scalable neural methods for reasoning with a symbolic knowledge base. *arXiv preprint arXiv:2002.06115*, 2020. Appeared in ICLR-2020.

[28] William W Cohen, Fan Yang, and Kathryn Rivard Mazaitis. Tensorlog: Deep learning meets probabilistic dbs. *arXiv preprint arXiv:1707.05390*, 2017.

[29] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[30] Amit Daniely, Nevena Lazic, Yoram Singer, and Kunal Talwar. Sketching and neural networks. *arXiv preprint arXiv:1604.05753*, 2016.

[31] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851*, 2017.

[32] Rajarshi Das, Arvind Neelakantan, David Belanger, and Andrew McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. *arXiv preprint arXiv:1607.01426*, 2016.

[33] Pradeep Dasigi, Matt Gardner, Shikhar Murty, Luke Zettlemoyer, and Eduard Hovy. Iterative search for weakly supervised semantic parsing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2669–2680, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[34] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers, 2021.

[35] H Leo H de Penning, Artur S d'Avila Garcez, Luís C Lamb, and John-Jules C Meyer. A neural-symbolic cognitive agent for online learning and reasoning. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[36] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, pages 2462–2467. Hyderabad, 2007.

[37] Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Lifted rule injection for relation embeddings. *arXiv preprint arXiv:1606.08359*, 2016.

[38] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[39] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[40] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[41] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[42] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[43] Bhuwan Dhingra, Jeremy Cole, Julian Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William Cohen. Time-aware language models as temporal knowledge bases. *Transactions of the Association for Computational Linguistics*, 10:257–273, 2022.

[44] Bhuwan Dhingra, Jeremy R. Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W. Cohen. Time-aware language models as temporal knowledge bases, 2021.

[45] Bhuwan Dhingra, Manzil Zaheer, Vidhisha Balachandran, Graham Neubig, Ruslan Salakhutdinov, and William W Cohen. Differentiable reasoning over a virtual knowledge base. In *International Conference on Learning Representations*, 2020.

[46] Bhuwan Dhingra, Manzil Zaheer, Vidhisha Balachandran, Graham Neubig, Ruslan Salakhutdinov, and William W Cohen. Differentiable reasoning over a virtual knowledge base. *arXiv preprint arXiv:2002.10640*, 2020.

[47] Luna Dong. Amazon product graph, Jun 2017.

[48] Oren Etzioni, Michele Banko, Stephen Soderland, and Daniel S Weld. Open information extraction from the web. *Communications of the ACM*, 51(12):68–74, 2008.

[49] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the 2011 conference on empirical methods in natural language processing*, pages 1535–1545, 2011.

[50] James Ferguson, Matt Gardner, Hannaneh Hajishirzi, Tushar Khot, and Pradeep Dasigi. Iirc: A dataset of incomplete information reading comprehension questions, 2020.

[51] Thibault Févry, Livio Baldini Soares, Nicholas FitzGerald, Eunsol Choi, and Tom Kwiatkowski. Entities as experts: Sparse memory access with entity supervision. *Conference on Empirical Methods in Natural Language Processing*, 2020.

[52] Catherine Finegan-Dollak, Jonathan K Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-SQL evaluation methodology. *arXiv preprint arXiv:1806.09029*, 2018.

[53] Yifan Gao, Chien-Sheng Wu, Shafiq Joty, Caiming Xiong, Richard Socher, Irwin King, Michael Lyu, and Steven C.H. Hoi. Explicit memory tracker with coarse-to-fine reasoning for conversational machine reading. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 935–945, Online, July 2020. Association for Computational Linguistics.

[54] Yifan Gao, Chien-Sheng Wu, Jingjing Li, Shafiq R. Joty, Steven C. H. Hoi, Caiming Xiong, Irwin King, and Michael R. Lyu. Discern: Discourse-aware entailment reasoning network for conversational machine reading. *CoRR*, abs/2010.01838, 2020.

[55] Google. Introducing the knowledge graph: things, not strings, May 2012.

[56] Georg Gottlob and Christos Papadimitriou. On the complexity of single-rule datalog queries. *Information and Computation*, 183(1):104–122, 2003.

[57] Nitish Gupta and Mike Lewis. Neural compositional denotational semantics for question answering. *CoRR*, abs/1808.09942, 2018.

[58] Swapnil Gupta, Sreyash Kenkre, and Partha Talukdar. CaRe: Open knowledge graph embeddings. 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 378–388, Hong Kong, China, November 2019. Association for Computational Linguistics.

[59] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020.

[60] Kelvin Guu, John Miller, and Percy Liang. Traversing knowledge graphs in vector space. *arXiv preprint arXiv:1506.01094*, 2015.

[61] Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. In *Advances in Neural Information Processing Systems*, pages 2026–2037, 2018.

[62] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*, 2016.

[63] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*, 2020.

[64] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. *CoRR*, abs/2007.01282, 2020.

[65] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering, 2021.

[66] Kelvin Jiang, Dekun Wu, and Hui Jiang. Freebaseqa: A new factoid qa data set matching trivia-style question-answer pairs with freebase. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 318–323, 2019.

[67] Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William W. Cohen, and Xinghua Lu. Pubmedqa: A dataset for biomedical research question answering, 2019.

[68] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[69] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, 2017.

[70] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[71] Nora Kassner, Benno Krojer, and Hinrich Schütze. Are pretrained language models symbolic reasoners over knowledge? *arXiv preprint arXiv:2006.10413*, 2020.

[72] Nora Kassner and Hinrich Schütze. Bert-knn: Adding a knn search component to pretrained language models for better qa. *arXiv preprint arXiv:2005.00766*, 2020.

[73] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[74] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[75] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc

Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 2019.

[76] John E Laird, Christian Lebiere, and Paul S Rosenbloom. A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. *Ai Magazine*, 38(4):13–26, 2017.

[77] Carolin Lawrence, Bhushan Kotnis, and Mathias Niepert. Attending to future tokens for bidirectional sequence generation. 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 1–10, Hong Kong, China, November 2019. Association for Computational Linguistics.

[78] Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6086–6096, 2019.

[79] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[80] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461, 2019.

[81] Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*, 2020.

[82] Patrick Lewis, Pontus Stenetorp, and Sebastian Riedel. Question and answer test-train overlap in open-domain question answering datasets. *arXiv preprint arXiv:2008.02637*, 2020.

[83] Patrick Lewis, Yuxiang Wu, Linqing Liu, Pasquale Minervini, Heinrich Küttler, Aleksandra Piktus, Pontus Stenetorp, and Sebastian Riedel. PAQ: 65 million probably-asked questions and what you can do with them. *Transactions of the Association for Computational Linguistics*, 9:1098–1115, 2021.

[84] Jing Li, Aixin Sun, and Shafiq R Joty. Segbot: A generic neural text segmentation model with pointer network. In *IJCAI*, pages 4166–4172, 2018.

[85] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *arXiv preprint arXiv:1611.00020*, 2016.

[86] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision, 2017.

[87] Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc V Le, and Ni Lao. Memory augmented policy optimization for program synthesis and semantic parsing. In *Advances in Neural Information Processing Systems*, pages 9994–10006, 2018.

[88] Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. Modeling relation paths for representation learning of knowledge bases. *arXiv preprint arXiv:1506.00379*, 2015.

[89] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[90] Jeffrey Ling, Nicholas FitzGerald, Zifei Shan, Livio Baldini Soares, Thibault Févry, David Weiss, and Tom Kwiatkowski. Learning cross-context entity representations from text. *arXiv preprint arXiv:2001.03765*, 2020.

[91] Hanxiao Liu, Yuexin Wu, and Yiming Yang. Analogical inference for multi-relational embeddings. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2168–2178. JMLR. org, 2017.

[92] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[93] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[94] Robert Logan, Nelson F Liu, Matthew E Peters, Matt Gardner, and Sameer Singh. Barack's wife hillary: Using knowledge graphs for fact-aware language modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5962–5971, 2019.

[95] Robert Logan, Nelson F. Liu, Matthew E. Peters, Matt Gardner, and Sameer Singh. Barack's wife hillary: Using knowledge graphs for fact-aware language modeling. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[96] Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems, 2016.

[97] Todor Mihaylov and Anette Frank. Knowledgeable reader: Enhancing cloze-style reading comprehension with external commonsense knowledge. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 821–832, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[98] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[99] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *EMNLP*, 2016.

[100] Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *CoRR*, abs/1606.03126, 2016.

[101] Bonan Min, Ralph Grishman, Li Wan, Chang Wang, and David Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 777–782, 2013.

[102] Sewon Min, Danqi Chen, Luke Zettlemoyer, and Hannaneh Hajishirzi. Knowledge guided text retrieval and reading for open domain question answering. *arXiv preprint arXiv:1911.03868*, 2019.

[103] Sewon Min, Kenton Lee, Ming-Wei Chang, Kristina Toutanova, and Hannaneh Hajishirzi. Joint passage ranking for diverse multi-answer retrieval. *arXiv preprint arXiv:2104.08445*, 2021.

[104] Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. AmbigQA: Answering ambiguous open-domain questions. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5783–5797, Online, November 2020. Association for Computational Linguistics.

[105] Dipendra Misra, Ming-Wei Chang, Xiaodong He, and Wen-tau Yih. Policy shaping and generalized update equations for semantic parsing from denotations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2442–2452, 2018.

[106] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, 2015.

[107] Xiangyang Mou, Chenghao Yang, Mo Yu, Bingsheng Yao, Xiaoxiao Guo, Saloni Potdar, and Hui Su. Narrative question answering with cutting-edge open-domain qa techniques: A comprehensive study, 2021.

[108] Stephen Mussmann and Stefano Ermon. Learning and inference via maximum inner product search. In *International Conference on Machine Learning*, pages 2587–2596, 2016.

[109] Anastasios Nentidis, Anastasia Krithara, Konstantinos Bougiatiotis, Georgios Paliouras, and Ioannis Kakadiaris. Results of the sixth edition of the BioASQ challenge. In *Proceedings of the 6th BioASQ Workshop A challenge on large-scale biomedical semantic indexing and question answering*, pages 1–10, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[110] Allen Newell, J. C. Shaw, and Herbert A. Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, 1959.

[111] Allen Newell and Herbert Simon. The logic theory machine–a complex information processing system. *IRE Transactions on information theory*, 2(3):61–79, 1956.

[112] Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernández Ábrego, Ji Ma, Vincent Y Zhao, Yi Luan, Keith B Hall, Ming-Wei Chang, et al. Large dual encoders are generalizable retrievers. *arXiv preprint arXiv:2112.07899*, 2021.

[113] Siru Ouyang, Zhuosheng Zhang, and Hai Zhao. Dialogue graph modeling for conversational machine reading. *CoRR*, abs/2012.14827, 2020.

[114] Panupong Pasupat and Percy Liang. Inferring logical forms from denotations. *arXiv preprint arXiv:1606.06900*, 2016.

[115] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, 2018.

[116] Matthew E Peters, Mark Neumann, Robert Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A Smith. Knowledge enhanced contextual word representations. 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 43–54, 2019.

[117] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*, 2019.

[118] Gadi Pinkas. Symmetric neural networks and propositional logic satisfiability. *Neural Computation*, 3(2):282–291, 1991.

[119] Nina Poerner, Ulli Waltinger, and Hinrich Schütze. Bert is not a knowledge base (yet): Factual knowledge vs. name-based reasoning in unsupervised qa. *arXiv preprint arXiv:1911.03681*, 2019.

[120] Peng Qi, Haejun Lee, Oghenetegiri Sido, Christopher D Manning, et al. Retrieve, rerank, read, then iterate: Answering open-domain questions of arbitrary complexity from text. *arXiv preprint arXiv:2010.12527*, 2020.

[121] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[122] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[123] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[124] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

[125] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[126] Pushpendre Rastogi, Adam Poliak, and Benjamin Van Durme. Training relation embeddings under logical constraints. In *KG4IR@ SIGIR*, pages 25–31, 2017.

[127] Abhilasha Ravichander, Alan W Black, Shomir Wilson, Thomas Norton, and Norman Sadeh. Question answering for privacy policies: Combining computational and legal perspectives. 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 4947–4958, Hong Kong, China, November 2019. Association for Computational Linguistics.

[128] Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. *arXiv preprint arXiv:2002.05969*, 2020. Appeared in ICLR-2020.

[129] Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. *arXiv preprint arXiv:2002.05969*, 2020.

[130] Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems*, 33:19716–19726, 2020.

[131] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M Marlin. Relation extraction with matrix factorization and universal schemas. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 74–84, 2013.

[132] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? *arXiv preprint arXiv:2002.08910*, 2020.

[133] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, pages 3788–3800, 2017.

[134] Mohammed Saeed, Naser Ahmadi, Preslav Nakov, and Paolo Papotti. Rulebert: Teaching soft rules to pre-trained language models. *arXiv preprint arXiv:2109.13006*, 2021.

[135] Marzieh Saeidi, Max Bartolo, Patrick Lewis, Sameer Singh, Tim Rocktäschel, Mike Sheldon, Guillaume Bouchard, and Sebastian Riedel. Interpretation of natural language rules in conversational machine reading, 2018.

[136] Marzieh Saeidi, Max Bartolo, Patrick S. H. Lewis, Sameer Singh, Tim Rocktäschel, Mike Sheldon, Guillaume Bouchard, and Sebastian Riedel. Interpretation of natural language rules in conversational machine reading. *CoRR*, abs/1809.01494, 2018.

[137] Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. *ACL*, 2020.

[138] Minjoon Seo, Tom Kwiatkowski, Ankur Parikh, Ali Farhadi, and Hannaneh Hajishirzi. Phrase-indexed question answering: A new challenge for scalable document comprehension. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 559–564, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[139] Peter Shaw, Philip Massey, Angelica Chen, Francesco Piccinno, and Yasemin Altun. Generating logical forms from graph representations of text and entities. *arXiv preprint arXiv:1905.08407*, 2019.

[140] Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, Ryan Sepassi, and Blake A. Hechtman. Mesh-tensorflow: Deep learning for supercomputers. *CoRR*, abs/1811.02084, 2018.

[141] Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. Asqa: Factoid questions meet long-form answers. *arXiv preprint arXiv:2204.06092*, 2022.

[142] Haitian Sun, Andrew Arnold, Tania Bedrax Weiss, Fernando Pereira, and William W Cohen. Faithful embeddings for knowledge base queries. *Advances in Neural Information Processing Systems*, 33:22505–22516, 2020.

[143] Haitian Sun, Andrew O Arnold, Tania Bedrax-Weiss, Fernando Pereira, and William W Cohen. Guessing what's plausible but remembering what's true: Accurate neural reasoning for question-answering. *arxiv preprint*, 2020.

[144] Haitian Sun, Tania Bedrax-Weiss, and William W Cohen. PullNet: Open domain question answering with iterative retrieval on knowledge bases and text. *arXiv preprint arXiv:1904.09537*, 2019.

[145] Haitian Sun, William Cohen, and Ruslan Salakhutdinov. Conditionalqa: A complex reading comprehension dataset with conditional answers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3627–3637, 2022.

[146] Haitian Sun, William W. Cohen, and Ruslan Salakhutdinov. Conditionalqa: A complex reading comprehension dataset with conditional answers. *CoRR*, abs/2110.06884, 2021.

[147] Haitian Sun, William W. Cohen, and Ruslan Salakhutdinov. End-to-end multihop retrieval for compositional question answering over long documents, 2021.

[148] Haitian Sun, William W. Cohen, and Ruslan Salakhutdinov. End-to-end multihop retrieval for compositional question answering over long documents. *CoRR*, abs/2106.00200, 2021.

[149] Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 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, 2018.

[150] Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William W Cohen. Open domain question answering using early fusion of knowledge bases and text. *EMNLP*, 2018.

[151] Haitian Sun, Patrick Verga, Bhuwan Dhingra, Ruslan Salakhutdinov, and William W Cohen. Reasoning over virtual knowledge bases with open predicate relations. In *International Conference on Machine Learning*, pages 9966–9977. PMLR, 2021.

[152] Karl Svozil. *Quantum logic*. Springer Science & Business Media, 1998.

[153] A. Talmor and J. Berant. The web as a knowledge-base for answering complex questions. In *North American Association for Computational Linguistics (NAACL)*, 2018.

[154] Alon Talmor and Jonathan Berant. The web as a knowledge-base for answering complex questions, 2018.

[155] Alon Talmor, Oyvind Tafjord, Peter Clark, Yoav Goldberg, and Jonathan Berant. Leap-of-thought: Teaching pre-trained models to systematically reason over implicit knowledge. *Advances in Neural Information Processing Systems*, 33:20227–20237, 2020.

[156] Théo Trouillon, Christopher R Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. Knowledge graph completion via complex tensor factorization. *The Journal of Machine Learning Research*, 18(1):4735–4772, 2017.

[157] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080, 2016.

[158] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[159] Pat Verga, Haitian Sun, Livio Baldini Soares, and William W Cohen. Facts as experts: Adaptable and interpretable neural memory over symbolic knowledge. *arXiv preprint arXiv:2007.00849*, 2020.

[160] Patrick Verga, David Belanger, Emma Strubell, Benjamin Roth, and Andrew McCallum. Multilingual relation extraction using compositional universal schema. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 886–896, 2016.

[161] Patrick Verga, Arvind Neelakantan, and Andrew McCallum. Generalizing to unseen entities and entity pairs with row-less universal schema. In *Proceedings of the 15th Conference of the*

European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pages 613–622, 2017.

[162] Nikhil Verma, Abhishek Sharma, Dhiraj Madan, Danish Contractor, Harshit Kumar, and Sachindra Joshi. Neural conversational QA: Learning to reason vs exploiting patterns. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7263–7269, Online, November 2020. Association for Computational Linguistics.

[163] Luke Vilnis, Xiang Li, Shikhar Murty, and Andrew McCallum. Probabilistic embedding of knowledge graphs with box lattice measures. *arXiv preprint arXiv:1805.06627*, 2018.

[164] Luke Vilnis and Andrew McCallum. Word representations via Gaussian embedding. *arXiv preprint arXiv:1412.6623*, 2014.

[165] Meng Wang, Ruijie Wang, Jun Liu, Yihe Chen, Lei Zhang, and Guilin Qi. Towards empty answers in SPARQL: Approximating querying with RDF embedding. In Denny Vrandečić, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web – ISWC 2018*, pages 513–529, Cham, 2018. Springer International Publishing.

[166] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[167] Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Cuihong Cao, Daxin Jiang, Ming Zhou, et al. K-adapter: Infusing knowledge into pre-trained models with adapters. *arXiv preprint arXiv:2002.01808*, 2020.

[168] Dirk Weissenborn, Tomáš Kočiskỳ, and Chris Dyer. Dynamic integration of background knowledge in neural nlu systems. 2017.

[169] Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Constructing datasets for multi-hop reading comprehension across documents, 2018.

[170] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

[171] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[172] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018.

[173] Yuxiang Wu, Yu Zhao, Baotian Hu, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. An efficient memory-augmented transformer for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2210.16773*, 2022.

[174] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*, 2017.

[175] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[176] Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems*, pages 2319–2328, 2017.

[177] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018.

[178] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. 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)*, volume 1, pages 1321–1331, 2015.

[179] Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. 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 1321–1331, Beijing, China, July 2015. Association for Computational Linguistics.

[180] Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 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)*, volume 2, pages 201–206, 2016.

[181] Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 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, Berlin, Germany, August 2016. Association for Computational Linguistics.

[182] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

[183] Michael J. Q. Zhang and Eunsol Choi. Situatedqa: Incorporating extra-linguistic contexts into qa, 2021.

[184] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. Variational reasoning for question answering with knowledge graph. In *AAAI*, 2018.

[185] Victor Zhong, Weijia Shi, Wen-tau Yih, and Luke Zettlemoyer. Romqa: A benchmark for robust, multi-evidence, multi-answer question answering. *arXiv preprint arXiv:2210.14353*, 2022.

[186] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.

[187] Victor Zhong and Luke Zettlemoyer. E3: Entailment-driven extracting and editing for conversational machine reading. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2310–2320, Florence, Italy, July 2019. Association for Computational Linguistics.