Open Question Answering Over Curated and Extracted Knowledge Bases

Anthony Fader[*]
Allen Institute for AI
Seattle, WA
afader@gmail.com

Luke Zettlemoyer
University of Washington
Seattle, WA
lsz@cs.washington.edu

Oren Etzioni
Allen Institute for AI
Seattle, WA
orene@allenai.org

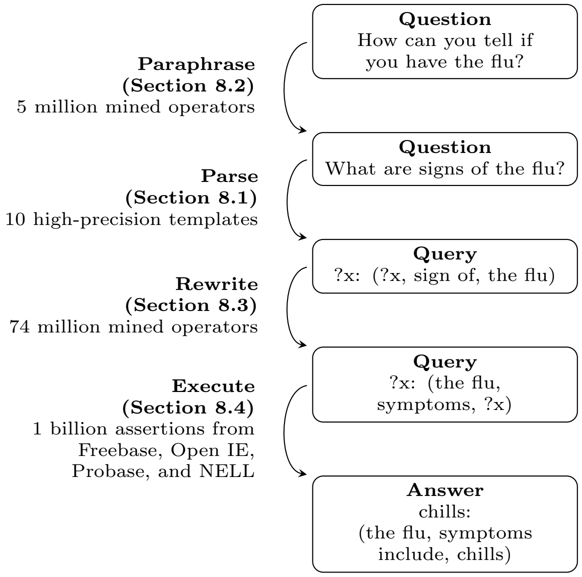
ABSTRACT

We consider the problem of open-domain question answering (Open QA) over massive knowledge bases (KBs). Existing approaches use either manually curated KBs like Freebase or KBs automatically extracted from unstructured text. In this paper, we present OQA, the first approach to leverage both curated and extracted KBs.

A key technical challenge is designing systems that are robust to the high variability in both natural language questions and massive KBs. OQA achieves robustness by decomposing the full Open QA problem into smaller sub-problems including question paraphrasing and query reformulation. OQA solves these sub-problems by mining millions of rules from an unlabeled question corpus and across multiple KBs. OQA then learns to integrate these rules by performing discriminative training on question-answer pairs using a latent-variable structured perceptron algorithm. We evaluate OQA on three benchmark question sets and demonstrate that it achieves up to twice the precision and recall of a state-of-the-art Open QA system.

1. INTRODUCTION

Open-domain question answering (Open QA) is a long-standing problem that has been studied for decades [12, 13]. Open QA systems need broad knowledge to achieve high coverage. Early systems took an information retrieval approach, where question answering is reduced to returning passages of text containing an answer as a substring [24]. Recent advances in constructing large-scale knowledge bases (KBs) [21, 2, 5] have enabled new systems that return an exact answer from a KB [4, 28, 23, 25, 10, 15, 3]. Some such systems have used curated KBs like Freebase,[1] which are high-precision but incomplete. Other systems have used extracted KBs like Open Information Extraction,[2] which have higher coverage but generally lower precision. In this paper, we present OQA, the first Open QA system to leverage both curated and extracted KBs.



Question
How can you tell if you have the flu?
Paraphrase (Section 8.2)
5 million mined operators
Question
What are signs of the flu?
Parse (Section 8.1)
10 high-precision templates
Query
?x: (?x, sign of, the flu)
Rewrite (Section 8.3)
74 million mined operators
Query
?x: (the flu, symptoms, ?x)
Execute (Section 8.4)
1 billion assertions from Freebase, Open IE, Probase, and NELL
Answer
chills:
(the flu, symptoms include, chills)

Figure 1: OQA automatically mines millions of operators (left) from unlabeled data, then learns to compose them to answer questions (right) using evidence from multiple knowledge bases.

A key challenge in Open QA is to be robust to the high variability found in natural language and the many ways of expressing knowledge in large-scale KBs. OQA achieves this robustness by decomposing the full QA problem into smaller sub-problems that are easier to solve. Figure 1 shows an example of how OQA maps the question "How can you tell if you have the flu?" to the answer "chills" over four steps. The first step rewrites the input question to "What are signs of the flu?" using a paraphrase operator mined from a large corpus of questions. The second step uses a hand-written template to parse the paraphrased question to the KB query "?x : (?x, signs of, the flu)." These two steps are synergistic; paraphrase operators effectively reduce the variance of the input questions, allowing OQA to use a small set of high-precision parsing rules while maintaining recall. The third step uses a query-rewrite operator to reformulate the query as "?x : (the flu, symptoms, ?x)." Query-rewrite operators are automatically mined from the KB, and allow the vocabulary mismatch between question words and KB

[*] Work completed while at the University of Washington.

[1] `http://freebase.com`

[2] `http://openie.cs.washington.edu`

KDD'14, August 24–27, 2014, New York, New York, USA.
Copyright 2014 ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623677 .

symbols to be solved independent of parsing. Finally, the fourth step executes the rewritten query against the KB, returning the final answer.

The operators and KB are noisy, so it is possible to construct many different sequences of operations (called derivations), very few of which will produce a correct answer. OQA learns from a small amount of question-answer data to find the best derivations. Because the derivations are unobserved in the training data, we use a latent-variable structured perceptron algorithm [31, 17, 22]. OQA uses a small set of general features that allow it to generalize from a limited number of training examples. Experiments on three benchmark question sets show that OQA outperforms the state-of-the-art Open QA system PARALEX [10], achieving twice the precision and recall.

In summary, we make the following contributions:

- We introduce OQA, the first Open QA system to leverage multiple, large-scale curated and extracted KBs.
- We describe an inference algorithm for deriving high-confidence answers (Section 6) and a hidden-variable structured perceptron algorithm for learning a scoring function from data (Section 7).
- We present algorithms for automatically mining paraphrase operators from a question corpus (Section 8.2) and KB-query rewrite operators (Section 8.3) from multiple KBs.
- We provide an empirical evaluation (Section 9), showing the relative contributions of different KBs and different system components across three question sets. We also compare OQA to the state-of-the-art QA systems PARALEX [10] and SEMPRE [3].
- We release the code and data from this work.[3]

In Section 2, we describe related work in more detail before moving on to the description of OQA (Sections 3–8) and experiments (Section 9).

2. RELATED WORK

Early work in Open QA used search engines as a source of background knowledge and relied on hand-written templates to map questions to search engine queries [16, 1]. In contrast, OQA utilizes a set of KBs, which enable it to combine knowledge extracted from Web text with curated knowledge. The KB abstraction also allows OQA to join multiple pieces of evidence to arrive at an answer, a technique that is not possible using just a search engine.

A major research thread in QA has been scaling up semantic parsing systems from small, single-domain KBs [30, 31, 26, 18, 7] to larger, multi-domain KBs like YAGO2 [28], DBpedia [23], and Freebase [4, 3, 15]. Curated KBs like Freebase are attractive for QA because they allow systems to reason over high-precision knowledge and return accurate answers. However, these systems have limited recall due to the inherent incompleteness of curated KBs. This phenomenon can be understood as a power-generality tradeoff: QA systems can rely on the accuracy and conciseness of a curated KB, but incomplete knowledge limits their generality.

The PARALEX system [10] was the first Open QA system to operate over a noisy, extracted KB. The biggest difference between PARALEX and OQA is how they decompose the QA problem. PARALEX uses self-labeled data to learn templates that directly map questions to queries—essentially performing paraphrasing, parsing, and query-rewriting in one step. OQA treats these as separate problems, which allows it to combine high-recall data mining techniques (for paraphrasing and query rewriting) with high-precision, hand-written rules (for parsing).

OQA's feature representation also differs from previous work. Previous systems use a large number of lexicalized features—those involving specific lexemes or KB symbols. OQA uses unlexicalized features that operate on the level of function words, part-of-speech tags, and corpus statistics. We found that the an unlexicalized feature representation generalizes better to questions involving relationships that were never seen during training.

3. TASK DEFINITION AND OVERVIEW

In this section, we define the QA task and give a high-level outline of OQA and our experiments.

Task and Metrics: We focus on the task of factoid QA, where the system takes a natural language question like "How can you tell if you have the flu?" as input and returns a short string answer like "chills" from a KB, or "no answer." We use precision (fraction of answered questions that are correct) and recall (fraction of questions that are correctly answered) as our primary evaluation metrics.

Knowledge Base: OQA uses a simple KB abstraction where ground facts are represented as string triples (argument1, relation, argument2). We use triples from curated and extracted knowledge sources (Section 4.1) and provide a lightweight query language to access the KB (Section 4.2).

Operators and Scoring Function: OQA models QA as a process where answers are derived from questions using operators Ops (Section 5.1). A sequence of operators linking a question to an answer is called a derivation. OQA computes the confidence of an answer derivation d using a linear scoring function score($d|\mathbf{f}, \mathbf{w}$), which is parameterized by a feature function $\mathbf{f}$ and feature weights $\mathbf{w}$ (Section 5.2).

Inference: In practice, the space of derivations defined by Ops is too large to enumerate. OQA uses heuristic search over partial derivations, guided by score($d|\mathbf{f}, \mathbf{w}$), to generate high-scoring candidate answers for an input question (Section 6).

Learning: OQA learns the weights $\mathbf{w}$ from a small set of question-answer pairs. Because annotated answer derivations are difficult to obtain, we use a latent-variable structured perceptron algorithm that treats answer derivations as unobserved variables in the training data (Section 7).

Operators and Features: OQA uses four types of operators: a small set of parsing operators (Section 8.1), a large set of paraphrase operators mined from a question corpus (Section 8.2), a large set of query-rewrite rules mined from the KB (Section 8.3), and an execution operator that interfaces with the KB (Section 8.4). Each operator is paired with a small set of features used to compute $\mathbf{f}$. Using a small feature set results in better generalization from limited training data.

[3] https://github.com/afader/oqa

Source	Type	# Triples	# Relation Phrases
Freebase	Curated	300M	18K
Open IE [9]	Extracted	500M	6M
Probase [27]	Extracted	200M	1
NELL [5]	Extracted	2M	300

Table 1: Knowledge bases used by OQA.

System Evaluation: We evaluate OQA using three question sets. We compare OQA to the Open QA system PARALEX and the Freebase QA system SEMPRE. We then test the contributions of each knowledge source and system component via ablation.

4. KNOWLEDGE BASE

This section describes where OQA's knowledge comes from and the query language it uses to access the knowledge.

4.1 Knowledge Base Sources

Table 1 summarizes the knowledge sources in OQA. OQA uses one curated KB (Freebase) and three extracted KBs (Open IE, Probase, and NELL).

Freebase is an open-domain, collaboratively edited KB. Freebase has relatively comprehensive coverage of certain domains like film or geography, but does not contain informal assertions like "chicken is high in protein." Freebase maintains canonical string representations of its entities and relations, which we use to coerce facts into string triples.[4]

Open IE [2, 9] is a family of techniques used to extract binary relationships from billions of web pages containing unstructured text. Open IE has the unique property that its relations are unnormalized natural language, which results in over two orders of magnitude more relation phrases than Freebase. The Open IE assertions are noisy and lack the comprehensive domain coverage found in Freebase. However, Open IE contains many of the informal assertions that are not found in curated KBs. For example, Open IE produces assertions like (pepper, provides a source of, vitamins a and c). Open IE triples are annotated with metadata including extractor confidence and corpus frequency, and some triple arguments are linked to Freebase entities [20].

Probase [27] is an extracted KB containing "is-a" relations, *e.g.*, (paris, is-a, beautiful city) or (physicist, is-a, scientist). Probase triples are annotated with statistical metadata that measure the confidence of each extraction.

NELL [5] is an extracted KB that contains approximately 300 relation phrases. NELL generally has high precision, but low recall.

The union of these KBs forms a single resource containing a billion noisy, redundant, and inconsistent assertions. While there is a vast body of literature exploring the problem of data-integration [8], these techniques require a target schema, which does not exist for our knowledge sources. Instead of making an offline commitment to a single schema, at runtime OQA hypothesizes many interpretations for each question. These hypotheses are encoded using the query language described in the next section.

4.2 Query Language

The query language used in OQA provides a lightweight interface between natural language and KB assertions. In contrast to the semantic parsing literature [30], a OQA query is not intended to represent the complete, formal semantic interpretation of a question. Instead, the query language is used to separate the parsing problem (identifying predicate-argument structure) from the vocabulary-matching problem (matching natural language symbols to KB symbols) [12, 15]. This factorization is at the core of the OQA approach, which uses different operators to solve each problem.

What fruits are a source of vitamin C?
?x : (?x, is-a, fruit) (?x, source of, vitamin c)

```
SELECT t0.arg1 FROM triples AS t0, triples AS t1
  WHERE
    keyword-match(t0.rel, "is-a")        AND
    keyword-match(t0.arg2, "fruit")      AND
    keyword-match(t1.rel, "source of")   AND
    keyword-match(t1.arg2, "vitamin c")  AND
    string-similarity(t0.arg1, t1.arg1) > 0.9
```

t0.arg1	t0.rel	t0.arg2	t1.arg1	t1.rel	t1.arg2
Lychee	is a	fruit	Lychees	good source of	vitamin c
star-fruit	is a	tropical fruit	starfruit	source of	vitamin c
pepper	is a	fresh fruit	pepper	provides a source of	vitamins c and a

Figure 2: Top: An example question and query used by OQA. Middle: The query semantics expressed as SQL. Bottom: The results when executed against a knowledge base (answers highlighted).

OQA's query language is capable of representing conjunctive queries [6]. Because our KB is unnormalized and contains only strings, OQA uses keyword matching and string similarity as primitive operations. Figure 2 shows how the question "What fruits are a source of vitamin C?" can be represented as the query ?x : (?x, is-a, fruit) (?x, source of, vitamin c). This particular query represents one possible mapping of the question to a predicate-argument structure. The middle box of Figure 2 shows how the semantics of the query can be interpreted as a SQL expression over a single table `triples` with string columns `arg1`, `rel`, and `arg2`. A OQA query consists of a projection variable (*e.g.*, ?x) and a list of conjuncts. Each conjunct contains a mix of string literals (*e.g.*, fruit) and variables. String literals correspond to keyword-matching constraints on the table columns, while variables correspond to string-similarity join constraints.

Having keyword matching and string similarity incorporated into the query semantics leads to another useful factorization. The query language provides a general, high-recall solution to the problem of minor surface-form variations (*e.g.*, joining "star-fruit" with "starfruit" or matching "source of" with "provides a source of" in Figure 2). OQA can then increase precision by computing question- or KB-specific features as soft constraints on the output. For example, it uses a feature that checks whether two join keys are linked to the same Freebase entity, if this information is available. This lets OQA maintain a simple data model (entity-linking is not required) while allowing for domain knowledge to be modeled via features.

5. DERIVING AND SCORING ANSWERS

OQA factors question answering into a set of smaller, related problems including paraphrasing, parsing, and query

[4] We discard Freebase facts that are not binary relations.

reformulation. The solutions to each of these sub-problems can then be applied in sequence to give a complete mapping from question to answer. Figure 3 shows example derivations for the question “How can you tell if you have the flu?” Our approach consists of two parts: (1) derivation operators, which define the space of possible answers for a given question, and (2) a scoring function, which returns a real-valued confidence for a derivation.

5.1 Derivation Operators

More formally, we model question answering as the process of mapping a question q to an answer a by applying operators from some set Ops. Each operator $o \in \mathsf{Ops}$ takes a state object $s \in \mathsf{States}$ as input and returns a set $o(s) \subseteq \mathsf{States}$ of successor states as output. State objects encode intermediate values that are used during the question-answering procedure. In Figure 3, the intermediate question “What are signs of the flu?” is a state object that is related to the query “?x : (?x, sign of, flu)” via a parsing operator. We use three types of states: question states, query states, and answer states.

Operations can be chained together into a derivation. A single derivation step encodes the process of applying an operator o to some state s and picking a successor state $s' \in o(s)$ from the output. A derivation $d = (\mathbf{o}, \mathbf{s}, k)$ consists of a sequence of k operators $\mathbf{o} = (o_1, \ldots, o_k)$ and a sequence of $k+1$ states $\mathbf{s} = (s_0, s_1, \ldots, s_k)$ satisfying $s_i \in o_i(s_{i-1})$ for all $1 \leq i \leq k$.[5]

An answer a is derivable from a question q under the operator set Ops if there exists some derivation $(\mathbf{o}, \mathbf{s}, k)$ such that $s_0 = q$ and $s_k = a$. We use the notation $\mathsf{Derivs}(q, \mathsf{Ops})$ to represent the space of all possible derivations from the question q under the operations Ops ending at answer a.

In our implementation of OQA, the operator set Ops contains millions of operators, combining both hand-written operators and operators learned from data. These operators are noisy: incorrect answers can be derived from most questions. Thus, estimating the confidence of a derivation is necessary for returning answers with high precision.

5.2 Scoring Function

To compute the confidence of a derivation, OQA uses a scoring function. The scoring function computes a real value for a given derivation, where large, positive scores are assigned to high-confidence derivations.

We make two assumptions about the form of the scoring function. First, we assume that the score is a linear function over features computed from a derivation. This will allow us to use familiar algorithms to learn the function from data. Second, we assume that the feature function decomposes over derivation steps. This allows us to use the scoring function to score partial derivations, which is useful for searching over $\mathsf{Derivs}(q, \mathsf{Ops})$ (discussed further in Section 6).

Under these assumptions, the score of a derivation $d = (\mathbf{o}, \mathbf{s}, k)$ can be written as

$$\text{score}(d|\mathbf{f}, \mathbf{w}) = \sum_{i=1}^{k} \mathbf{w} \cdot \mathbf{f}(s_0, s_{i-1}, o_i, s_i), \qquad (1)$$

where $\mathbf{f}$ is an n-dimensional feature function that maps a derivation step into $\mathbb{R}^n$ and $\mathbf{w}$ is an n-dimensional weight vector. Because s_0 (the input question) is a constant in all derivations, we pass it as an argument to the feature function. This allows features to compute properties relating derivation steps to the input question.

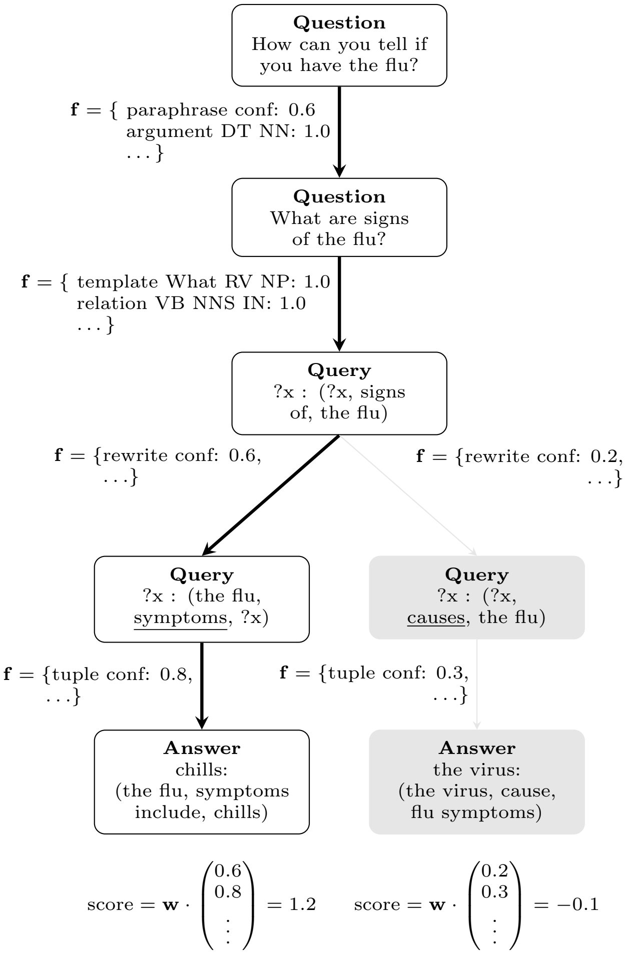

Question
How can you tell if you have the flu?
f = { paraphrase conf: 0.6
argument DT NN: 1.0
...}
Question
What are signs of the flu?
f = { template What RV NP: 1.0
relation VB NNS IN: 1.0
...}
Query
?x : (?x, signs of, the flu)
f = {rewrite conf: 0.6,
...}
f = {rewrite conf: 0.2,
...}
Query
?x : (the flu, symptoms, ?x)
Query
?x : (?x, causes, the flu)
f = {tuple conf: 0.8,
...}
f = {tuple conf: 0.3,
...}
Answer
chills:
(the flu, symptoms include, chills)
Answer
the virus:
(the virus, cause, flu symptoms)
score = w · (0.6, 0.8, ...) = 1.2
score = w · (0.2, 0.3, ...) = −0.1

Figure 3: OQA compute operator-specific features to discriminate between correct derivations (left) and incorrect derivations (right).

Figure 3 shows an example of how the scoring function can discriminate between a correct answer (left) and an incorrect answer (right). The derivation on the left rewrites the original query using a high-confidence rule and uses high-confidence evidence returned from the KB. The derivation on the right uses a low-confidence rewrite rule and low-confidence evidence—and is thus assigned a lower score.

In our implementation of OQA, we learn $\mathbf{w}$ from a small set of question-answer pairs (Section 7) and define $\mathbf{f}$ to compute operator-specific features (Section 8).

6. INFERENCE

We focus on the task of finding a single answer with the highest confidence under the scoring function, which amounts to solving the following equation for an input question q:

$$d^* = \underset{d \in \mathsf{Derivs}(q, \mathsf{Ops})}{\arg\max} \; \text{score}(d|\mathbf{f}, \mathbf{w}). \qquad (2)$$

[5] In OQA, $2 \leq k \leq 4$: parsing and execution steps are required to derive an answer, and we limit derivations to have at most one paraphrase step and one query-rewrite step.

The underlying knowledge base and set of operators are both large enough that exhaustively enumerating $\mathsf{Derivs}(q, \mathsf{Ops})$ is not feasible. Instead, OQA uses beam search informed by the scoring function to explore $\mathsf{Derivs}(q, \mathsf{Ops})$. We refer to the beam search routine as DeriveAnswers.

The algorithm takes a question q as input and returns a set of derivations $D \subseteq \mathsf{Derivs}(q, \mathsf{Ops})$. The output set D is constructed by iteratively growing partial derivations starting at the initial question state $s_0 = q$. The algorithm maintains a beam of partial derivations, each scored by the function $\text{score}(d|\mathbf{f}, \mathbf{w})$. At every iteration, a partial derivation is selected to be extended. Extending a derivation $d = (\mathbf{o}, \mathbf{s}, k)$ amounts to computing successors to the state s_k and appending the successors to construct new derivations. This process is repeated until there are no partial derivations left to extend or until a time limit is reached.

In practice, the scoring function $\text{score}(d|\mathbf{f}, \mathbf{w})$ generally assigns higher scores to short derivations, *e.g.* derivations that do not use a query-rewrite operator. We found that this bias will flood the beam with high-scoring partial derivations occurring early in the search, and later options will not be considered. To avoid this problem, we maintain separate beams for each state-type in the search, similar to the decoding algorithms used in statistical machine translation [14]. OQA uses beam search both at runtime and during learning, which we describe in the next section.

7. LEARNING

A key challenge in learning $\text{score}(d|\mathbf{f}, \mathbf{w})$ is that obtaining labeled answer derivations requires expert annotators and is extremely time consuming. Following recent work in semantic parsing [7, 3, 15], we use question-answer pairs as indirect supervision and treat answer derivations as unobserved variables in the training data. Question-answer pairs like q = "How can you tell if you have the flu?", A = {"chills", "fever", "aches"} are significantly easier to obtain and do not require expert annotators.

We use the latent-variable structured perceptron algorithm [31, 17, 22] to learn $\mathbf{w}$ from example question-answer pairs. Figure 4 shows the pseudocode for the LearnWeights algorithm.

The algorithm takes as input a set of N pairs (q_i, A_i) for $i = 1, \dots, N$, where A_i is a set containing string answers to q_i. For each training example, the algorithm calls DeriveAnswers to generate a candidate set of answer derivations D. The algorithm then chooses a derivation $\hat{d}$ that has the highest score according to the current weights and makes the prediction $\hat{a} = \text{answer}(\hat{d})$. If $\hat{a}$ is correct (*i.e.*, it is in A_i), the algorithm proceeds to the next example. If $\hat{a}$ is incorrect, then the learner picks the highest scoring derivation d^* such that $\text{answer}(d^*)$ is in A_i. The algorithm then performs an additive update $\mathbf{w} = \mathbf{w} + \mathbf{f}(d^*) - \mathbf{f}(\hat{d})$. If there are no derivations with a correct answer in D, then the learner immediately proceeds to the next example without performing an update. Finally, the algorithm returns the average value of $\mathbf{w}$ over all iterations, which improves generalization [11].

```
function LearnWeights
    Inputs:
      Number of iterations $T$
      $N$ example questions with answers $(q_1, A_1), \dots, (q_N, A_N)$
      Initial model $(\mathsf{Ops}, \mathbf{f}, \mathbf{w})$ (Defined in Section 5)
      Function DeriveAnswers (Defined in Section 6)
  Output:
      Learned weights $\mathbf{w}$
  for $t = 1, \dots, T$
      for $i = 1, \dots, N$
          $D = \text{DeriveAnswers}(q_i, \mathsf{Ops}, \mathbf{f}, \mathbf{w})$
          $\hat{d} = \arg\max_{d \in D} \text{score}(d|\mathbf{f}, \mathbf{w})$
          if $\text{answer}(\hat{d}) \notin A_i$
              $D^* = \{d \in D : \text{answer}(d) \in A_i\}$
              $d^* = \arg\max_{d \in D^*} \text{score}(d|\mathbf{f}, \mathbf{w})$
              $\mathbf{w} = \mathbf{w} + \mathbf{f}(d^*) - \mathbf{f}(\hat{d})$
  return average of $\mathbf{w}$ over all iterations
```

Figure 4: The weight-learning algorithm.

8. OPERATORS AND FEATURES

In this section, we describe the operators that OQA uses to derive answers. The operators factor the end-to-end QA problem into smaller subproblems:

Parsing operators (Section 8.1) are responsible for interfacing between natural language questions and the KB query language described in Section 4.2. OQA uses a small number of high-precision templates to map questions to queries.

Paraphrase operators (Section 8.2) are responsible for rewording the input question into the domain of a parsing operator. In an offline process, OQA mines 5 million lexicalized paraphrase-templates from an unlabeled corpus of open-domain questions.

Query-rewrite operators (Section 8.3) are responsible for interfacing between the vocabulary used in the input question and the internal vocabulary used by the KBs. OQA implements its query-rewrite operators by mining a set of 75 million relation-entailment pairs from the knowledge bases described in Section 4.1.

The execution operator (Section 8.4) is responsible for fetching and combining evidence from the KB, given a query.

For each operator, OQA computes general, domain independent features that are used in the scoring function. These features are *unlexicalized* in the sense that they do not compute any values associated with content words in either the question or the KB.

In the following subsections, we describe each type of operator in detail and describe the operator-specific features used by the scoring function.

8.1 Parsing Operators

To map questions to queries, we use the set of 10 handwritten operators shown in Table 2. Each operator consists of a question pattern and a query pattern.

A question pattern is expressed as a regular expression over part-of-speech (POS) tags and function words. To detect noun phrases (NPs), we use a POS pattern that matches a sequence of nouns, determiners, and adjectives. We use the ReVerb pattern [9] to detect relation phrases. Each question pattern uses named capture-groups to select substrings from the input question.

A query pattern consists of a query (defined in Section 4.2) containing pointers to capture groups from the question pattern. When a question pattern matches a question, the captured strings are substituted into the query pattern to generate a complete query. For example, the question

Question Pattern	Query Pattern	Example Question	Example Query
Who/What RV_{rel} NP_{arg}	(?x, rel, arg)	Who invented papyrus?	(?x, invented, papyrus)
Who/What Aux NP_{arg} RV_{rel}	(arg, rel, ?x)	What did Newton discover?	(Newton, discover, ?x)
Where/When Aux NP_{arg} RV_{rel}	(arg, rel in, ?x)	Where was Edison born?	(Edison, born in, ?x)
Where/When is NP_{arg}	(arg, is in, ?x)	Where is Detroit?	(Detroit, is in, ?x)
Who/What is NP_{arg}	(arg, is-a, ?x)	What is potassium?	(potassium, is-a, ?x)
What/Which NP_{rel2} Aux NP_{arg} RV_{rel1}	(arg, rel1 rel2, ?x)	What sport does Sosa play?	(Sosa, play sport, ?x)
What/Which NP_{rel} is NP_{arg}	(arg, rel, ?x)	What ethnicity is Dracula?	(Dracula, ethnicity, ?x)
What/Who is NP_{arg}'s NP_{rel}	(arg, rel, ?x)	What is Russia's capital?	(Russia, capital, ?x)
What/Which NP_{type} Aux NP_{arg} RV_{rel}	(?x, is-a, type) (arg, rel, ?x)	What fish do sharks eat?	(?x, is-a, fish) (sharks, eat, ?x)
What/Which NP_{type} RV_{rel} NP_{arg}	(?x, is-a, type) (?x, rel, arg)	What states make oil?	(?x, is-a, states) (?x, make, oil)

Table 2: High-precision parsing operators used to map questions to queries. Question templates are expressed using noun phrases (NP), auxiliary verbs (Aux), and ReVerb patterns (RV). Subscripts denote regex-style capture groups.

Source Template	Target Template
How does _ affect your body?	What body system does _ affect?
What is the latin name for _?	What is _'s scientific name?
Why do we use _?	What did _ replace?
What to use instead of _?	What is a substitute for _?
Was _ ever married?	Who has _ been married to?

Table 3: Example paraphrase operators that extracted from a corpus of unlabeled questions.

Source Query	Target Query
(?x, children, ?y)	(?y, was born to, ?x)
(?x, birthdate, ?y)	(?x, date of birth, ?y)
(?x, is headquartered in, ?y)	(?x, is based in, ?y)
(?x, invented, ?y)	(?y, was invented by, ?x)
(?x, is the language of, ?y)	(?y, languages spoken, ?x)

Table 4: Example query-rewrite operators mined from the knowledge bases described in Section 4.1.

pattern in the first row of Table 2 matches "Who invented papyrus?" and captures the substrings {rel → invented, arg → papyrus}. These are substituted into the query pattern (?x, rel, arg) to produce the output (?x, invented, papyrus).

Features: Because some question patterns may be more reliable than others, we include an indicator feature for each question pattern. We also include indicator features for the POS sequence of the capture groups and the POS tags to the left and right of each capture group.

8.2 Paraphrasing Operators

The parsing operators in Table 2 have high precision but low recall. To increase recall, we use paraphrase operators to map questions onto the domain of the parsing operators. Each paraphrase operator is implemented as a pair of paraphrase templates like the examples in Table 3.

Each paraphrase template consists of a natural language string with a slot that captures some argument. For example, the first source template in Table 3 matches the question "How does nicotine affect your body?" This question can then be paraphrased by substituting the argument "nicotine" into the target template, yielding the new question "What body system does nicotine affect?"

We follow the work of Paralex and automatically mine these source/target template pairs from the WikiAnswers[6] paraphrase corpus [10]. The WikiAnswers paraphrase corpus consists of 23 million question-clusters that WikiAnswers users have grouped as synonymous. Each question cluster contains an average of 25 questions. In general, the clusters have low precision due to mistakes or users grouping related, but non-synonymous questions (*e.g.*, "How to say shut up in french?" is grouped with "Is it nice to say shut up?").

We extracted 200,000 templates that occurred in at least 10 question clusters. For each pair of templates (t, t'), we define the co-occurrence count $c(t, t')$ to be the number of clusters where t and t' both occur with the same argument. For example, if a cluster contains the questions "Why do we use computers?" and "What did computers replace?" we would increment the count c(Why do we use _?, What did _ replace?) by 1. For each template pair (t, t') such that $c(t, t') \geq 5$, we define paraphrase operators $t \to t'$ and $t' \to t$. This generates a set of 5 million paraphrase operators. During inference, all possible paraphrases of a question q are computed by considering all substrings of q (up to 5 tokens) as the argument.

Features: The paraphrase operators are automatically extracted from a noisy corpus, and are not always reliable. We compute statistical and syntactic features to estimate the confidence of using the operator $t \to t'$ to paraphrase a question q to a new question q' using argument a. The statistical features include the pointwise mutual information (PMI) between t and t' in the WikiAnswers corpus and a language model score of q'. The syntactic features include the POS sequence of the matched argument a, and the POS tags to the left and right of a in q.

8.3 Query-Rewrite Operators

To handle the mismatch between natural language vocabulary and the KB vocabulary, we mine query rewrite rules. We focus on handling the mismatch between relation words in the question and relation words in the KB.[7] Table 4 lists example query rewrite rules. Each rule encodes a translation from one relation phrase to another, with a possible re-ordering of the arguments. For example, the first row in Table 4 is an operator that allows the relation phrase "children" to be rewritten as "was born to^{-1}," where the superscript denotes inverted argument ordering.

We follow previous work on mining equivalent relations [19, 29] and count the number of shared argument-pairs between two relation phrases. For example, the tuples (hermann einstein, children, albert einstein) and (albert einstein, was born to, hermann einstein) both appear in the KB, so "children" and "was born to^{-1}" share the argument pair

[6] `http://wiki.answers.com`

[7] Rewriting arguments is future work.

(hermman einstein, albert einstein). We construct a query rewrite operator for each pair of relation phrases (r, r') that share at least 10 argument pairs. This results in a set of 74 million (r, r') pairs that we include as operators.

Features: As with the paraphrase templates (Section 8.2), we compute the PMI for each pair of relation phrases as a feature.

8.4 Execution Operator

The execution operator takes a query as input and returns a set of tuples, as shown in Figure 2. We store the KB (arg1, relation, arg2) triples in an inverted index[8] that allows for efficient keyword search over the three triple fields. We implemented a simple query optimizer that performs joins over triples by making multiple queries to the KB. Due to the size of the KB, we limit each keyword search over the triples to return the top 100 hits. The output of the execution operator is an answer state, containing a string answer and a joined tuple of evidence.

Features: We use features to estimate the reliability of the KB output. The features examine properties of the query, the returned tuple evidence, and the answer.

We measure the keyword similarity between two strings by lemmatizing them, removing stopwords, and computing the cosine similarity. We then include the keyword similarity between the query and the input question, the keyword similarity between the query and the returned evidence, and an indicator feature for whether the query involves a join.

The evidence features compute KB-specific properties. Extracted triples have confidence scores, which are included as features. We compute the join-key string similarity measured using the Levenshtein distance. We also include indicator features for the source of each triple (*e.g.*, whether the triple is from Open IE or Freebase).

The answer features compute conjunctions of properties of the input question and the answer. We compute whether the question begins with some common prefixes (*e.g.*, Who, What, When, How many, *etc.*). For the answer, we compute word-shape features (*e.g.*, “Kansas” has the word shape “Aaaa” and “December 1941” has the word shape “Aaaa 1111”). This allows the system to learn that features like *question starts with When* $\wedge$ *answer has shape 1111* are indicative of a correct answer.

9. EXPERIMENTAL SETUP

We are interested in answering three questions: (1) How does oqa compare to the state-of-the-art systems Paralex and Sempre? (2) How do the different knowledge sources affect performance? (3) How do the different system components affect performance?

We investigate these questions by comparing performance on three question sets. Given a question q, each system returns an answer a with confidence $c \in \mathbb{R}$ or “no answer.” We then measure the precision (correct answers/answers returned), recall (correct answers/questions), and F1 score (harmonic mean of precision and recall). We also compute precision-recall curves that show how precision is traded for recall as the minimum confidence to return an answer is varied. We describe the three question sets in Section 9.1 and the system settings in Section 9.2.

WebQuestions 3,778 train 2,032 test	who influenced wolfgang amadeus mozart? who won the super bowl xliv 2010? where was nicki minaj born? what is in liverpool england? who is the leader of france 2012?
TREC 962 train 517 test	What other countries do Kurds live in? What year was Barry Manilow born? What format was VHS's main competition? Who is the chairman of WWE? What is Muscular Dystrophy?
WikiAnswers 1,334 train 7,310 test	What is Matthew henson's mothers name? Who is a retired gay nfl football player? Do beluga whales eat penguins? Why were the conquistadors important? How does psychiatry benefit society?

Table 5: The three question sets used in our experiments.

9.1 Question Sets

In our experiments, we use three question sets: WebQuestions, TREC, and WikiAnswers. Figure 5 shows statistics and example questions from each set.

WebQuestions was introduced by the authors of the Sempre system [3]. The questions were generated from Google Autocomplete using a seed set of Freebase entities. Amazon Mechanical Turk users then provided answers in the form of Freebase concepts. Questions that could not be answered using Freebase were filtered out. Out of the three test sets, WebQuestions has the unique property that the questions are known *a priori* to be answerable using Freebase.

TREC was introduced for the purpose of evaluating information retrieval QA systems [24]. We re-purpose the TREC questions to test our KB-based Open QA systems. While the TREC questions were designed to be answerable using a small collection of test documents, they are not guaranteed to be answerable using any of the KBs described in Section 4.1.

WikiAnswers is a set of questions that were randomly sampled from a crawl of WikiAnswers. The WikiAnswers question set is completely disjoint from the corpus used to extract the paraphrasing operators described in Section 8.2. WikiAnswers questions are very challenging and ambiguous, and are not necessarily answerable by any KB.

WebQuestions and TREC both have gold-standard answer sets for each question, and WikiAnswers questions often have no answers available. However, due to the open-domain nature of our experiments, the gold-standard answer sets are incomplete. If a system's top answer was not already included in the provided gold-standard sets, we manually tagged the answers as correct or incorrect.

9.2 System Settings

OQA: We examine two different training settings for oqa. In the first setting, we trained oqa on each question set independently, resulting three different sets of weights. In the second setting, we trained oqa on the union of the WebQuestions, TREC, and WikiAnswers training sets, resulting in one set of weights.

For inference, we used a beam capacity of 1,000 and a search time limit of 20 seconds. For learning, we initialized

[8] `https://lucene.apache.org/solr/`

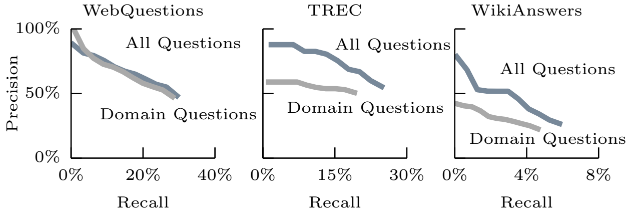


Figure 5: Training OQA on questions from all question sets leads to greater precision and recall than training on domain questions only.

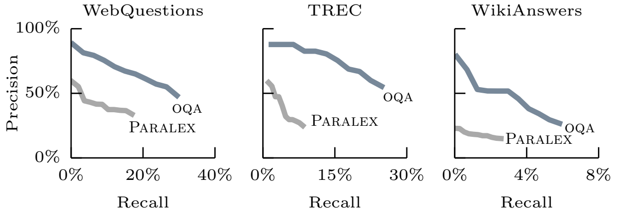


Figure 6: OQA has higher precision and recall than the Open QA system Paralex.

10 of the feature weights to be +1/-1 based on whether the features are indicative of good derivations (*e.g.*, PMI scores) or bad derivations (*e.g.*, verbs as paraphrase-template arguments). We set the number of perceptron iterations (between 1 and 5) using a fraction of held-out training data. For the first perceptron iteration, we interactively trained the system by providing the set D^* in Figure 4.

Paralex: The authors of Paralex provide a learned model.[9] We used Paralex to parse questions to queries, and then execute them against the same KB as oqa. Paralex provides a score for each query. For each answer that the query returns, we use the score from the query as a measure of confidence.

Sempre: The authors of Sempre also make it available for download.[10] Sempre comes with a model trained on the WebQuestions question set. We attempted to train Sempre with questions from TREC and WikiAnswers, but found that the WebQuestions model had higher performance on held-out development questions, so we use the WebQuestions model in our experiments.

10. EXPERIMENTAL RESULTS

Figure 5 shows the precision-recall curves comparing oqa under the two different training settings. Training the scoring function on questions from all three question sets resulted in higher precision and recall on TREC and WikiAnswers, but had no effect on WebQuestions performance. This is likely due to the fact that oqa is unable to derive correct answers for many of the questions in TREC and WikiAnswers, so the effective number of training examples is smaller than WebQuestions.

Figure 6 shows the precision-recall curves of oqa and Paralex on the test questions. oqa achieves both higher precision and recall than Paralex across all three question sets. oqa's scoring function was able to avoid many of the errors made by Paralex. For example, Paralex made a systematic error confusing "Where" and "When" questions, *e.g.*, it was unable to tell that "1985" is an unlikely answer to a question that begins with "Where." In contrast, oqa was able to compute features of the full derivation (including the answer), which allowed it to learn not to make this type of error.

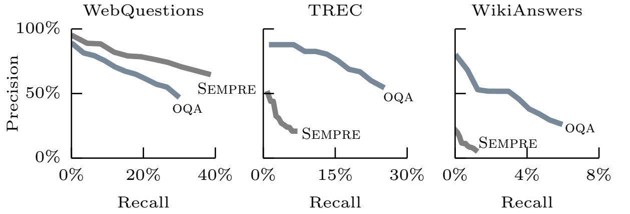


Figure 7: Sempre has higher precision and recall on WebQuestions, which are known to be answerable in Freebase. However, OQA outperforms Sempre on TREC and WikiAnswers, which were not developed for any particular KB.

Figure 7 shows the precision-recall curves of oqa and Sempre on the test questions. In this case, Sempre is has higher precision and recall than oqa on WebQuestions. Sempre performs better on WebQuestions through its use of lexicalized features, *e.g.*, there is a single feature indicating that the string "see in" corresponds to the Freebase relation "tourist attraction." These features allow Sempre to better fit the distribution of relations and entities in WebQuestions. In contrast, oqa uses only unlexicalized features like POS tags and corpus statistics like PMI, which limit oqa's ability to fit the WebQuestions training data.

However, oqa performs significantly better on TREC and WikiAnswers for two reasons. First, oqa is uses both extracted and curated knowledge sources, so it is more likely to have an answer in its KB. Second, Sempre requires significant lexical overlap in its training and testing set, which is not satisfied in the TREC and WikiAnswers questions.

Figure 8 shows the effects of removing different components from oqa. The weight-learning algorithm significantly improves performance over the default weights defined in the experimental setup.

The paraphrase operators improve performance on WebQuestions and WikiAnswers, but not on TREC. We found that many TREC questions were covered by the parser operators in Table 2, so the paraphrase operators did not add much. In contrast, the WebQuestions and WikiAnswers questions exhibit much more lexical and syntactic variation, so the paraphrase operators were more useful.

The query-rewrite operators led to only a slight improvement on the TREC question set, and had at best no effect on WebQuestions and WikiAnswers. We examined the output and found some high-confidence examples where the query-rewrite operators helped, *e.g.*, the question "When did the Big Dig begin?" was answered by rewriting "(big dig, begin in, ?x)" to "(big dig, broke ground in, ?x)." However, most derivations that used a query-rewrite operator were assigned low confidence, and had limited effect on recall.

Figure 9 shows the effects of removing a knowledge source from oqa on system performance. Removing Open IE from the KB lowers the F1 score across all test sets. Freebase

[9] `http://knowitall.cs.washington.edu/paralex/`
[10] `https://github.com/percyliang/sempre`

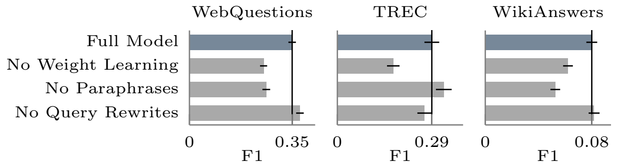


Figure 8: The relative contributions of each system component depend on the distribution of test questions. (Error bars represent one standard deviation from the mean, computed over 10,000 bootstrap samples of test data.)

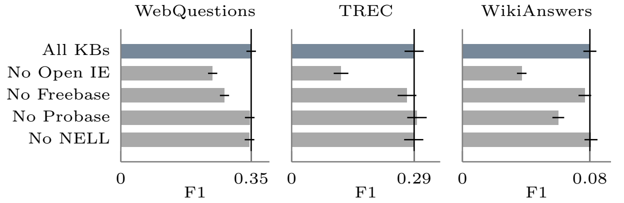


Figure 9: OQA performs best using multiple knowledge sources, in particular Open IE, Freebase, and Probase.

helps the most on the WebQuestions set (which was designed specifically for Freebase), but is less useful for TREC and WikiAnswers. Probase is most useful for WikiAnswers, which contains many "What is..." questions that can be answered using Probase's is-a relations. NELL is largely a subset of the other KBs, so it had no effect on OQA's performance.

11. DISCUSSION

The experimental results in the previous section exemplify the power-generality tradeoff discussed in the introduction: OQA uses a small number of general, unlexicalized features, which provided better generalization. However, this limits OQA's ability to take full advantage of the training data. For example, OQA was unable to answer questions like "What time zone is in South Africa have?" despite seeing several nearly-identical questions in the WebQuestions training data. A challenge for the future is to engineer OQA to take advantage of lexical cues when they are available. Extending OQA to induce higher-precision operators during discriminative training may be one way.

One problem that has gone unaddressed by all of the discussed QA systems is modeling whether a given question is answerable with the given KB and operators. For example, OQA currently has no way to answer truth-false questions like "Are dogs mammals?" Yet OQA systematically chains low-confidence operators together to derive incorrect answers for them, which hurts precision. A measure of answerability would be useful in scenarios where high-precision is required.

Operator	State
Input	Who did Michael J Fox marry?
Parse	?x: (Michael J Fox, marry, ?x)
Rewrite	?x: (Michael J Fox, has wife, ?x)
Execute	**Tracy Pollan:** (Michael J. Fox, has wife, Tracy Pollan)
Input	What are brake pads made of?
Paraphrase	What material are brake pads made of?
Parse	?x: (?x, is-a, material) (brake pads, made of, ?x)
Execute	**copper:** (copper, is-a, material) (The brake pads, were made of, copper)
Input	What are some examples of building maintenance jobs?
Parse	?x: (?x, example of, building maintenance jobs)
Rewrite	?x: (?x, is-a, building maintenance jobs)
Execute	**changing light bulb:** (changing light bulb, is-a, small building maintenance job)

Table 6: Examples from the test data where OQA derives a correct answer.

Operator	State
Input	What animal represents California?
Paraphrase	What are California's symbols?
Parse	?x: (california, symbols, ?x)
Execute	**CWT:** (California Water Service, Trading symbol, CWT)
Input	Who did George Washington admire?
Parse	?x: (George Washington, admire, ?x)
Execute	**presidents and generals:** (George Washington, was admired by, presidents and generals)

Table 7: Example derivations from the test data where OQA derives an incorrect answer.

Table 6 shows examples from the test data where OQA derives correct answers. The first example shows a query rewrite operator that modifies the relation "marry" to "has wife." The second example shows a paraphrase operator that maps "What are _ made of?" to "What material are _ made of?" In this case, the paraphrase operator introduces a type constraint that does not appear in the input question, which is beneficial for selecting the correct answer. The third example highlights the benefit of extracted knowledge, which contains obscure assertions like "(changing light bulb, is-a, small building maintenance job)."

Table 7 shows examples where OQA derives incorrect answers. The first example shows that the paraphrase operators can be too general, in this case overgeneralizing "animal" to "symbol." This combines with "California Water Service" incorrectly matching "California," resulting in the incorrect answer "CWT." The second example shows that better features are needed to prevent errors like matching an active voice ("admire") with the passive voice ("was admired by").

12. CONCLUSION

We introduced OQA, a novel Open QA system that is the first to leverage both curated and extracted knowledge. We described inference and learning algorithms that OQA uses to derive high-confidence answers. Our experiments demonstrate that OQA generalizes well to unseen questions and makes significant improvements over a state-of-the-art Open

QA baseline. The data and code for this work is available at `https://github.com/afader/oqa`.

Acknowledgements

This research was supported in part by ONR grant N00014-11-1-0294, DARPA contract FA8750-13-2-0019, and ARO grant W911NF-13-1-0246, and was carried out at the University of Washington's Turing Center.

13. REFERENCES

[1] M. Banko, E. Brill, S. Dumais, and J. Lin. AskMSR: Question answering using the worldwide web. In *2002 AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases*, 2002.

[2] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In *IJCAI*, 2007.

[3] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on Freebase from question-answer pairs. In *EMNLP*, 2013.

[4] Q. Cai and A. Yates. Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. In *ACL*, 2013.

[5] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. H. Jr., and T. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

[6] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, 1977.

[7] J. Clarke, D. Goldwasser, M.-W. Chang, and D. Roth. Driving Semantic Parsing from the World's Response. In *CoNLL*, 2010.

[8] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[9] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *EMNLP*, 2011.

[10] A. Fader, L. Zettlemoyer, and O. Etzioni. Paraphrase-Driven Learning for Open Question Answering. In *ACL*, 2013.

[11] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Mach. Learn.*, 37(3):277–296, 1999.

[12] B. J. Grosz, D. E. Appelt, P. A. Martin, and F. C. N. Pereira. TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces. *Artificial Intelligence*, 32(2):173–243, 1987.

[13] B. Katz. Annotating the World Wide Web using Natural Language. In *RIAO*, pages 136–159, 1997.

[14] P. Koehn. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In *AMTA*, Lecture Notes in Computer Science, pages 115–124. Springer, 2004.

[15] T. Kwiatkowski, E. Choi, Y. Artzi, and L. Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *EMNLP*, 2013.

[16] C. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. *ACM Trans. Inf. Syst.*, 19(3):242–262, 2001.

[17] P. Liang, A. Bouchard-Côté, D. Klein, and B. Taskar. An end-to-end discriminative approach to machine translation. In *ACL*, 2006.

[18] P. Liang, M. Jordan, and D. Klein. Learning Dependency-Based Compositional Semantics. In *ACL*, 2011.

[19] D. Lin and P. Pantel. DIRT – Discovery of inference rules from text. In *KDD*, 2001.

[20] T. Lin, Mausam, and O. Etzioni. Entity linking at web scale. AKBC-WEKEX, 2012.

[21] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant Supervision for Relation Extraction Without Labeled Data. In *ACL*, 2009.

[22] X. Sun, T. Matsuzaki, D. Okanohara, and J. Tsujii. Latent variable perceptron algorithm for structured classification. In *IJCAI*, 2009.

[23] C. Unger, L. Bühmann, J. Lehmann, A.-C. N. Ngomo, D. Gerber, and P. Cimiano. Template-Based Question Answering over RDF Data. In *WWW*, 2012.

[24] E. M. Voorhees and D. M. Tice. Building a question answering test collection. In *SIGIR*, 2000.

[25] S. Walter, C. Unger, P. Cimiano, and D. Bär. Evaluation of a Layered Approach to Question Answering over Linked Data. In *ISWC*, 2012.

[26] Y. W. Wong and R. J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *ACL*, 2007.

[27] W. Wu, H. Li, H. Wang, and K. Q. Zhu. Probase: a probabilistic taxonomy for text understanding. In *SIGMOD*, 2012.

[28] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural Language Questions for the Web of Data. In *EMNLP*, 2012.

[29] A. Yates and O. Etzioni. Unsupervised methods for determining object and relation synonyms on the web. *JAIR*, 34:255–296, March 2009.

[30] J. M. Zelle and R. J. Mooney. Learning to Parse Database Queries Using Inductive Logic Programming. In *AAAI*, 1996.

[31] L. S. Zettlemoyer and M. Collins. Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In *UAI*, 2005.