Schema2QA: High-Quality and Low-Cost Q&A Agents for the Structured Web

Silei Xu Giovanni Campagna Jian Li† Monica S. Lam
Computer Science Department
Stanford University
Stanford, CA, USA
{silei, gcampagn, jianli19, lam}@cs.stanford.edu

ABSTRACT

Building a question-answering agent currently requires large annotated datasets, which are prohibitively expensive. This paper proposes Schema2QA, an open-source toolkit that can generate a Q&A system from a database schema augmented with a few annotations for each field. The key concept is to cover the space of possible compound queries on the database with a large number of in-domain questions synthesized with the help of a corpus of generic query templates. The synthesized data and a small paraphrase set are used to train a novel neural network based on the BERT pretrained model.

We use Schema2QA to generate Q&A systems for five Schema.org domains, restaurants, people, movies, books and music, and obtain an overall accuracy between 64% and 75% on crowdsourced questions for these domains. Once annotations and paraphrases are obtained for a Schema.org schema, no additional manual effort is needed to create a Q&A agent for any website that uses the same schema. Furthermore, we demonstrate that learning can be transferred from the restaurant to the hotel domain, obtaining a 64% accuracy on crowdsourced questions with no manual effort. Schema2QA achieves an accuracy of 60% on popular restaurant questions that can be answered using Schema.org. Its performance is comparable to Google Assistant, 7% lower than Siri, and 15% higher than Alexa. It outperforms all these assistants by at least 18% on more complex, long-tail questions.

ACM Reference Format:
Silei Xu, Giovanni Campagna, Jian Li, and Monica S. Lam. 2020. Schema2QA: High-Quality and Low-Cost Q&A Agents for the Structured Web. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411974

1 INTRODUCTION

The adoption of virtual assistants is increasing at an unprecedented rate. Companies like Amazon and Google are rapidly growing their proprietary platforms of third-party skills, so consumers can access different websites and IoT devices by voice. Today, Alexa has 100,000 skills [13] and Google claims 1M actions. Websites can make themselves accessible to each of the assistant platforms by supplying a skill containing many sample natural language invocations. Virtual assistant platforms then use proprietary technology to train the linguistic interfaces. Such interfaces are unavailable outside the platforms, and reproducing them requires a prohibitively high investment, which is not affordable for all but the largest companies. In the meantime, the existing proprietary linguistic interfaces can only accurately answer simple, popular questions.

We envision a future where cost-effective assistant technology is open and freely available so every company can create and own their voice interfaces. Instead of submitting information to proprietary platforms, companies can publish it on their website in a standardized format, such as the Schema.org metadata. The open availability of the information, together with the open natural language technology, enables the creation of alternative, competitive virtual assistants, thus keeping the voice web open.

Towards this goal, we have developed a new methodology and a toolkit, called Schema2QA, which makes it easy to create natural language Q&A systems. Developers supply only their domain information in database schemas, with a bit of annotation on each database field. Schema2QA creates an agent that uses a neural model trained to answer complex questions. Schema2QA is released as a part of the open-source Genie toolkit for virtual assistants[1].

1.1 A Hypothesis on Generic Queries

Today's assistants rely on manually annotating real user data, which is prohibitively expensive for most companies. Wang et al. previously suggested a new strategy where canonical natural language questions and formal queries are automatically generated from a database schema [24]. These questions are then paraphrased to create a natural language training set. Unfortunately, this approach does not yield sufficiently accurate semantic parsers. We proposed substituting the single canonical form with developer-supplied templates to increase the variety of synthesized data, and training a neural network with both synthesized and paraphrased data [5]. Good accuracy has been demonstrated on event-driven commands connecting up to two primitive functions. However, such commands are significantly simpler than database queries. A direct application of this approach to Q&A is inadequate.

This research focuses on questions that can be answered with a database. Database systems separate *code* from *data*: with just a few operators in relational algebra, databases can deliver all kinds

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10.
https://doi.org/10.1145/3340531.3411974

[1] https://github.com/stanford-oval/genie-toolkit

† Jian Li conducted this research at Stanford while visiting from the Computer Science and Engineering Department of the Chinese University of Hong Kong

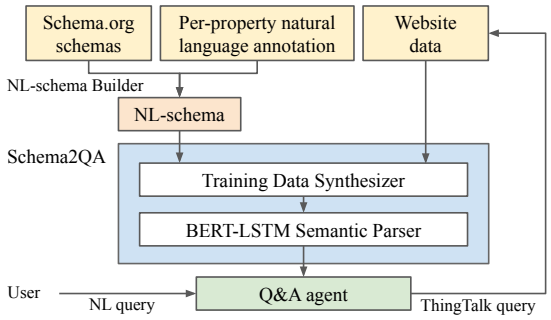

Figure 1: The Schema2QA system architecture.

of answers by virtue of the data stored in the various fields of the tables. Would there be such a separation of code from data in natural language queries that can be answered with a database? We define the part of the query that corresponds to the "code" as *generic*. For example, "what is the *field* of the *table*" is a generic question that maps to the projection of a given *table* onto a given *field*.

We hypothesize that *by factoring questions into generic queries and domain-specific knowledge, we can greatly reduce the incremental effort needed to answer questions on new domains of knowledge.* We are *not* saying that every question can be derived by substituting the parameters in generic queries with the ontology of a domain. We only wish to reduce the need for annotated real data in training with a large corpus of synthesized sentences for each domain. The data synthesizer can generate hundreds of thousands of combinations of database operations and language variations to (1) provide better functional coverage, and (2) teach the neural network compositionality. To handle the full complexity and irregularity in natural language, we rely on the neural network to generalize from the combination of synthesized and human-produced sentences, using pretraining to learn general natural language.

1.2 Research Methodology

Our research methodology is to design a representation that separates the generic and the domain-specific parts of a question. We extend the ThingTalk virtual assistant programming language [5] to include the primitives necessary for querying databases. We create Schema2QA, a tool that automatically generates a Q&A agent from an *NL-schema*, a database scheme whose fields are annotated with ways they are referred to in natural language.

Schema2QA builds the skill by training a novel neural semantic parser, based on the BERT pretrained model [6], with mostly synthesized data. It synthesizes pairs of natural-language sentences and ThingTalk code, from generic query templates and the domain-specific field annotations. For example, Schema2QA can synthesize "Show me the address of 5-star restaurants with more than 100 reviews" and this corresponding ThingTalk code:

$$[address] \texttt{ of Restaurant}, aggregateRating.ratingValue = 5 \;\&\&\; aggregateRating.reviewCount \geq 100$$

It augments the synthesized data and a small number of paraphrased sentences with real parameter values. At inference time, the parser translates questions into ThingTalk, which is executed to retrieve the answer from a database, which may be updated dynamically.

One important ontology is Schema.org, a commonly used representation for structured data in web pages. We create an NL-schema Builder tool that converts the RDF-based Schema.org representation into an NL-schema, which is openly available in Thingpedia [4]. Once manual annotations and paraphrases are available for a domain, a Q&A agent can be generated automatically for any website in that domain using its Schema.org metadata. The agent can also be entered in Thingpedia, an open skill repository used currently by the open-source privacy-preserving Almond assistant [4]. The architecture of Schema2QA is shown in Fig. 1.

1.3 Contributions

The contributions of this paper include:

- A new methodology and a Schema2QA toolkit, that significantly reduce the cost in creating Q&A systems for databases by leveraging a set of generic question templates. The toolkit synthesizes a large training set cost-effectively using a template-based algorithm, and trains a novel BERT-based neural semantic parser.
- We demonstrated that Schema2QA is applicable to Schema.org on six domains: restaurants, people, books, movies, music, and hotels. We constructed a large training set of complex web queries, annotated in ThingTalk, with more than 400,000 questions in each domain. This dataset also includes 5,516 real-world crowdsourced questions. This test set can serve as a benchmark for future Q&A systems[2].
- Experimental results show that Q&A systems built with Schema2QA can understand a diverse set of complex questions, with an average query accuracy of 70%. We also show that we can transfer the restaurant skill to the hotel domain with no manual effort, achieving 64% accuracy on crowdsourced questions.
- Schema2QA achieves an accuracy of 60% on popular restaurant questions that can be answered using Schema.org. The accuracy is comparable to Google Assistant, 7% lower than Siri and 15% higher than Alexa. In addition, Schema2QA outperforms all these assistants by at least 18% on more complex, long-tail questions.

1.4 Outline

We first discuss related work in Section 2. We present how we synthesize varied questions along with their ThingTalk representation, the application to Schema.org, and our neural model in Sections 3 through 5, respectively. Lastly, we present experimental results and conclude.

2 RELATED WORK

Question answering. Question answering (QA) is a well-studied problem in natural language processing. A subset of the QA field is to build Natural language Interfaces to Databases (NLIDB). Early proposed systems to solve this problem use rule-based approaches, which are not robust to variations in natural language [8, 16, 19].

[2]The data can be downloaded from https://oval.cs.stanford.edu/releases/

More recently, neural semantic parsing has been widely adopted for question answering [7, 12]. However, this technique requires a large corpus of annotated questions, which is expensive. Previous work has proposed crowdsourcing paraphrases to bootstrap new semantic parsers [24]. Our work on Genie further suggested training with data synthesized from manually tuned templates, based on the constructs in a programming language where each skill provides domain-specific templates mapping to website-specific APIs [5]. DBPal [25] uses a similar approach to augment existing datasets with synthesized data. In this paper, we propose a more varied set of templates, covering not only the variety in functionality, but also the variety in natural language. This avoids the need for domain-specific templates and existing datasets.

Semantic parsing. Recent work in semantic parsing has focused on generating SQL directly from natural language queries [26, 27, 30, 31]. Most of the work employs neural encoder-decoder frameworks [7, 11, 17] with attention. Later work on the WikiSQL dataset [31] leveraged the constrained space of questions in that dataset and developed syntax-specific decoding strategies [23, 28, 29]. Often, these models use pre-trained models as embeddings to a larger, SQL-aware encoder [9]. Hwang et al. [10] explored different combinations of the BERT encoder with LSTM decoders: a vocabulary-only decoder and a pointer-only one. After finding both to be ineffective, they proposed an application-specific decoder. Our model instead uses a pointer-generator controlled by a switch probability. This is effective in our task, and it is also simpler (thus easier to train) and more general.

Using Schema.org in virtual assistants. The Schema.org vocabulary is in active use by commercial virtual assistants for their builtin skills, for example in the Alexa Meaning Representation Language [14] and in Google Assistant. Compositional queries based on Schema.org require expert annotation on large training sets [18]. Furthermore, because of the annotation cost, compositional query capabilities are not available to third-parties, which are limited to an intent classifier [15]. Our approach only requires a small amount of developer effort, and the effort can be shared among all websites using the same Schema.org properties. Furthermore, each website can own their generated semantic parser and improve it for their own use case, instead of relying on a proprietary one.

3 NATURAL LANGUAGE TO THINGTALK

The principle behind neural semantic parsers is to let the neural network generalize from many possible sentences annotated with their meaning in a formal representation. However, because languages are compositional, the neural network must see many possible combinations in order not to overfit. In contrast, compositionality is baked into grammar-based parsers, but they fail to handle the irregularities and contextual variations in natural language. *We propose to use templates to generate a large variety of perfectly annotated natural language sentences to provide coverage and to teach the neural network compositionality, and we use a small number of paraphrases to expose the network to more variety in natural language.* Our approach allows the network to generalize and understand natural language with significantly less manually labeled data.

Table t	tn | sel | pr | agg | cmp | $sort$ | idx | $join$
Selection sel	t, f
Projection pr	`[`fn^+`]` `of` t
Aggregation agg	`aggregate` $aggop$ `of` t
Computation cmp	`compute` $expr$ {`as` fn}$^?$ `of` t
Sorting $sort$	`sort` fn {`asc` | `desc`} `of` t
Indexing idx	t `[`v`]` | t `[`v `:` v`]`
Join $join$	t `join` t
Filter f	`true` | `false` | `!`f | f `&&` f | f `||` f | v $cmpop$ v | t `{` f `}`
Expression $expr$	v | $expr$ `+` $expr$ | $expr$ `-` $expr$ | $expr$ `*` $expr$ | $expr$ `/` $expr$ | `distance(`$expr$, $expr$`)` | $aggop$`(`v`)`
Comparison $cmpop$	`=` | `≥` | `≤` | `=~` | `contains` | `in_array` | ...
Agg. operator $aggop$	count | sum | avg | min | max
Table name tn	identifier
Field name fn	identifier
Value v	literal | fn | `lookup(`literal, tn`)`

Figure 2: The formal definition of ThingTalk.

Here we first define ThingTalk, the formal target language of our semantic parser. We then describe the natural language sentences we synthesize with (1) canonical templates to cover all the possible queries and (2) generic query templates to provide linguistic variety.

3.1 The ThingTalk Query Language

Our target language is an extension of ThingTalk, a previously proposed programming language optimized for translation from natural language [5]. Our extensions make ThingTalk a functional subset of SQL. In this paper, we focus on the query aspects of ThingTalk; readers are referred to previous work for the design of the rest of the language. ThingTalk is designed with a relational database model. ThingTalk queries have the form:

$$\left[fn^+ \texttt{ of}\right]^? \ \ table \ \ [, filter]^? \left[\texttt{join} \ table \ [, filter]^?\right]^*$$

where *table* is the type of entity being retrieved (similar to a table name in SQL), *filter* applies a selection predicate that can make use of the fields in the table, and *fn* is an optional list of field names to project on. The full grammar is shown in Fig. 2. ThingTalk queries support the standard relational algebra operators commonly used by natural language questions. These include sorting a table, indexing & slicing a table, aggregating all results, and computing a new field for each row. The `join` operator produces a table that contains the fields from both tables; fields from the second table shadow the fields of the first table. All the operators can be combined compositionally.

ThingTalk uses a static type system. It has native support for named entities, such as people, brands, countries, etc. Every table includes a unique *id* field, and ThingTalk uses the "lookup" operator to look up the ID of a specific entity by name. ThingTalk introduces array types to express joins at a higher level, and to avoid the use of glue tables for many-to-many joins. ThingTalk also includes common types, such as locations, dates and times, and also includes user-relative concepts such as "here" and "now". The latter allows the parser to translate a natural language sentence containing those words to a representation that does not change with the current location or current time.

For example, the `distance` operator can be used to compute the distance between two locations. Combined with sorting and indexing, we can express the query "find the nearest restaurant" as:

(`sort` *distance* `asc of comp` distance(*geo*, `here`) `of` Restaurant)[1]

Table 1: Canonical templates mapping natural language to ThingTalk code.

Operator	Natural language template	ThingTalk	Minimal Example
Selection	table := t : *table* with f : *fname* equal to v : *value*	$t, f = v$	[restaurants] with [cuisine] equal to [Chinese]
	| t : *table* with f : *fname* greater than v : *value*	$t, f \geq v$	[restaurants] with [rating] greater than [3.5]
	| t : *table* with f : *fname* less than v : *value*	$t, f \leq v$	[restaurants] with [rating] less than [3.5]
	| t : *table* with f : *fname* containing v : *value*	t, contains(f, v)	[people] with [employers] containing [Google]
Projection	fref := the f : *fname* of t : *table*	f of t	the [cuisine] of [restaurants]
Join	table := the t_1 : *table* of t_1 : *table*	(t_2 join t_1), in_array(id, t_2)	[reviews with ...] of [restaurants with ...]
Aggregation	fref := the number of *table*	aggregate count of t	the number of [restaurants]
	| the *op* f : *fname* in t : *table*	aggregate *op* f of t	the [average] [rating] of [restaurants]
Ranking	table := the t : *table* with the min f : *fname*	(sort f asc of t)[1]	the [restaurants] with the min [rating]
	| the t : *table* with the max f : *fname*	(sort f desc of t)[1]	the [restaurants] with the max [rating]
	| the n : *number* t : *table* with the min f : *fname*	(sort f asc of t)[1 : n]	the [3] [restaurants] with the min [rating]
	| the n : *number* t : *table* with the max f : *fname*	(sort f desc of t)[1 : n]	the [3] [restaurants] with the max [rating]
Quantifiers	table := t_1 : *table* with t_2 : *table*	t_1, exists(t_2, in_array(id, t_1))	[restaurants with ...] with [reviews with ...]
	| t_1 : *table* with no t_2 : *table*	t_1, !exists(t_2, in_array(id, t_1))	[restaurants with ...] with [no reviews with ...]
Row-wise function	fref := the distance of t : *table* from l : *loc*	compute distance(*geo*, l) of t	the distance of [restaurants] from [here]
	| the number of f : *fname* in t : *table*	compute count(f) of t	the number of [reviews] in [restaurants]

The query reads as: select all restaurants, compute the distance between the field *geo* and the user's current location (and by default, store it in the *distance* field), sort by increasing distance, and then choose the first result (with index 1).

ThingTalk is designed such that the clauses in the query compose in the same way as the phrases in English. This helps with synthesis. For example, the query "who wrote the 1-star review for Shake Shack?" is expressed as:

[*author*] of ((Restaurant, *id* = lookup("shake shack"))
join(Review, *reviewRating.ratingValue* = 1)),
in_array(*id*, *review*)

The query reads as "search the restaurant 'Shake Shack', do a cross-product with all 1-star reviews, and then select the reviews that are in the list of reviews of the restaurant; of those, return the author". The "1-star review" phrase corresponds to the "Review" clause of the query, and "Shake Shack" corresponds to the "Restaurant" clause. The combination "the 1-star review for Shake Shack" corresponds to the join and "in_array" selection expression. Adding "who wrote" to the sentence is equivalent to projecting on the "author" field.

3.2 Canonical Templates

Schema2QA uses the previously proposed concept of *templates* to associate ThingTalk constructs with natural language [5]. Templates are production rules mapping the grammar of natural language to a *semantic function* that produces the corresponding code. Formally, a template is expressed as:

$$nt := [v : nt \mid \text{literal}]^+ \Rightarrow sf$$

The non-terminal *nt* is produced by expanding the terminals and non-terminals on the right-hand side of the := sign. The bound variables v are used by the semantic function *sf* to produce the corresponding code.

For each operator in ThingTalk, we can construct a *canonical template* expressing that operator in natural language. The main canonical templates are shown in Table 1. The table omits logical connectives such as "and", "or", and "not" for brevity. These canonical templates express how the composition of natural language is reflected in the composition of corresponding relational algebra operators to form complex queries.

With the canonical templates, we can cover the full functionality of ThingTalk, and thus the full span of questions that the system can answer. However, these synthesized sentences do not reflect how people actually ask questions. In the next section, we discuss how we can increase the variety in the synthesized set with a richer set of generic templates.

3.3 Generic Query Templates

Sentence Types. Let's start with the concept of sentence types. In English, there are four types: declarative, imperative, interrogative, and exclamatory. In Q&A, we care about declarative ("I am looking for . . ."), imperative ("Search for . . .") and interrogative ("What is . . ."). We have generic query templates to generate all these three different sentence types.

Based on the type of the object in question, different interrogative pronouns can be used, in lieu of the generic "what". "Who" maps to "what is the person that"; "where" maps to "what is the location of"; "when" maps to "what is the time that". The distinction between persons, animals/inanimate objects, locations, and time is so important to humans that it is reflected in our natural language. To create natural-sounding sentences, we create generic query templates for these different "W"s and select the one to use according to the declared types of the fields in sentence synthesis.

In addition, we can ask *yes-no questions* to find out if a stated fact is true, or *choice* questions, where the answer is to be selected from given candidates.

Question Structure. There is a great variety in how questions can be phrased, as illustrated by questions on the simple fact that "Dr. Smith is Alice's doctor" in Table 2. We create templates to combine the following three factors to achieve variety in question structure.

- Two-way relationships. Many important relationships have different words to designate the two parties; e.g. doctor and patient, employer and employee.
- Parts of speech. A relationship can be expressed using phrases in different parts of speech. Besides has-a noun, is-a noun, active verb, and passive verb, as shown in Table 2, we can also have adjective and prepositional phrases. Not all fields can be expressed using all the different parts of speech, as shown for "servesCuisine" and "rating" in Table 3.

Table 2: Variety in question structure given the fact that Dr. Smith is Ann's doctor

Relation	Part-of-Speech	Statement	Unknown: Ann	Unknown: Dr. Smith
Doctor	has-a noun	Ann has Dr. Smith as a doctor.	Who has Dr. Smith as a doctor?	Who does Ann have as a doctor?
	is-a noun	Dr. Smith is a doctor of Ann.	Who is Dr. Smith a doctor of?	Who is a doctor of Ann?
	active verb	Dr. Smith treats Ann.	Whom does Dr. Smith treat?	Who treats Ann?
	passive verb	Ann is treated by Dr. Smith.	Who is treated by Dr. Smith?	By whom is Ann treated?
Patient	has-a noun	Dr. Smith has Ann as a patient.	Who does Dr. Smith have as a patient?	Who has Ann as a patient?
	is-a noun	Ann is a patient of Dr. Smith.	Who is a patient of Dr. Smith?	Who is Ann a patient of?
	active verb	Ann consults with Dr. Smith.	Who consults with Dr. Smith?	With whom does Ann consult?
	passive verb	Dr. Smith is consulted by Ann.	By whom is Dr. Smith consulted?	Who is consulted by Ann?

Table 3: Example annotations for fields servesCuisine, rating, and alumniOf in different parts of speech.

POS	servesCuisine : String	rating : Number	alumniOf : Organization	Example sentence for alumniOf
has-a noun	*value* cuisine, *value* food	rating *value*, *value* star	*value* degree, alma mater *value*	who have a Stanford degree?
is-a noun	×	×	alumni of *value*, *value* alumni	who are alumni of Stanford?
active verb	serves *value* cuisine	×	studied at *value*, attended *value*	who attended Stanford?
passive verb	×	rated *value* star	educated at *value*	who are educated at Stanford?
adjective	*value*	*value*-star	*value*	who are Stanford people?
prepositional	×	×	from *value*	who are from Stanford?

Table 4: Type-specific comparison words.

Type	Comparative
Time	earlier, before, later, after
Duration	shorter, longer
Distance	closer, nearer, farther, more distant
Length	shorter, longer
Currency	cheaper, more expensive
Weight	lighter, smaller, heavier, larger
Speed	slower, faster
Temperature	colder, hotter

- Information of interest. We can ask about either party in a relationship; we can ask for Alice's doctor or Dr. Smith's patient.

In our knowledge representation, the rows in each table represent unique entities, each with a set of fields. Suppose we have a table of patients with "doctors" as a field; asking for Alice's doctor maps to a projection operation, whereas asking for Dr. Smith's patient maps to a selection operation. On the other hand, with a table of doctors with "patients" as a field, the converse is true. To avoid having multiple representations for the same query, our semantic parser assumes the existence of only one field for each relationship. It is up to the implementation to optimize execution if multiple fields exist.

Types and Measurements. The encoding of types in language carries over to measurements. Mathematical operators such as "$a < b$", "$a > b$", "min", "max" translate to different words depending on the kind of measurements. For example, for weight we can say "heavier" or "lighter". The list of type-specific comparison words is shown in Table 4. For each type, there are corresponding superlative words ("heaviest", "lightest", etc.). There is a relatively small number of commonly used measurements. By building these varieties in the training data synthesis, we do not have to rely on manually annotated data to teach the neural networks these concepts.

Types are also expressed in certain common, shortened question forms. For example, rather than saying "what is the distance between here and X", we can say "how far is X"; the "distance" function and the reference point "here" are implicit. Similarly, instead of saying "the restaurant with the shortest distance from here", we can say "the nearest restaurant". These questions are sufficiently common that it is useful to include them in the generic set.

Connectives. In natural language, people use connectives to compose multiple clauses together, when they wish to search with multiple filters or joins. Clauses can be connected with an explicit connective such as "and", "but", and "with". However, the connective can also be implicit, when composing natural short phrases based on its part-of-speech category. For example, multiple adjective phrases can be concatenated directly: people say "5-star Italian restaurants" rather than "5-star and Italian restaurants".

3.4 Supplying Domain-Specific Knowledge

The domain-specific information necessary to generate questions is provided by the developer in the form of annotations on each field. Table 3 shows examples of annotations for 3 fields in the database. The developer is asked to provide common variations for each property, and they can iterate the annotations based on error analysis. Even for the same POS category, there may be alternatives using different words. For example, "alumniOf" can have two different verb phrases "studied at" and "attended". The examples in this table also show that not every field can be referred to using the 6 POS modifiers. For restaurants, we can say "has French food", "serves French food", "French restaurant", but not "restaurant is a French", "served at French", or "a restaurant of French". Similarly, for ratings, we can say "has 5 stars", "rated 5 stars", "5-star restaurant", but not "restaurant is a 5 rating", "restaurant rates 5", or a "restaurant of 5 stars". We rely on the domain expert to annotate their fields with appropriate phrases for each POS category.

3.5 Template-Based Synthesis

To synthesize data for the neural semantic parser, Schema2QA uses the Genie template-based algorithm [5]. Genie expands the templates by substituting the non-terminals with the previously generated derivations and applying the semantic function. Because the expansion is exponential, only a limited fraction of the possible combinations are sampled. The expansion continues recursively, up to a configurable depth.

A small portion of the synthesized data is paraphrased by crowdsource workers. Both the synthetic and paraphrased sets are then augmented by replacing the field values with the actual values in the knowledge base. This allows us to expose more real values to the neural network to learn how they match to different fields.

Because synthesized data is used in training, Schema2QA training sets cover a lot more questions than previous methods based only on paraphrasing. Schema2QA does not need to rely on the distribution of sentences in the training set to match the test set. Instead, it only needs to generalize minimally from sentences that are present in the training set. As long as the training set has good coverage of real-world questions, this ensures high accuracy. Additionally, developers can refine their skills with more templates.

4 Q&A FOR SCHEMA.ORG

Schema2QA is applicable to any *ontology*, or database schema, that uses strict types with a type hierarchy consisting of the base classes of people, things, places, events. To answer questions on the structured web, we develop NL-schema Builder to convert the RDF-based Schema.org ontology into an NL-schema.

4.1 Data Model of Schema.org

Schema.org is a markup vocabulary created to help search engines understand and index structured data across the web. It is based on RDF, and uses a graph data model, where nodes represent objects. Nodes are connected by *properties* and are grouped in *classes*. Classes are arranged in a multiple inheritance hierarchy where each class can be a subclass of multiple other classes, and the "Thing" class is the superclass of all classes.

Each property's domain consists of one or more classes that can possess that property. The same property can be used in separate domains. For example, "Person" and "Organization" classes both use the "owns" property. Each property's range consists of one or more classes or primitive types. Additionally, as having any data is considered better than none, the "Text" type (free text) is always implicitly included in a property's range. For properties where free text is the recommended or only type expected, e.g. the "name" property, "Text" is explicitly declared as the property type.

4.2 NL-schema Builder

To adapt the Schema.org ontology for use by Schema2QA, the NL-schema Builder (1) converts the graph representation into database tables, (2) defines their fields and assigns the types, and (3) provides annotations for each field.

Creating tables. We categorize classes in Schema.org as either *entity* or *non-entity* classes. Entity classes are those that refer to well-known identities, such as people, organizations, places, events, with names that the user can recognize. The Builder identifies entity classes using the hierarchy in Schema.org, and converts each class into a table in the Schema2QA representation. The properties of entity classes are converted into fields of the table. Given a target website for the Q&A system, the Builder only includes those properties used by the website to eliminate irrelevant terms.

Non-entity classes can only be referred to as properties of entity classes. The Builder inserts properties of non-entity classes directly into the entity class that refers to them, giving them unique names with a prefix. This design eliminates tables that cannot be referred to directly in natural language, hence simplifying synthesis. It also reduces the number of joins required to query the database, hence simplifying translation.

Assigning field types. The Schema.org class hierarchy provides most of the information needed by Schema2QA. However, the union type in Schema.org is problematic: it is not supported in Schema2QA to reduce ambiguity when parsing natural language. Here, the Builder trades off precision for ease of translation. For each property, the Builder simply picks among the types in its range the one with the highest priority. The types in decreasing order of priority are: record types, primitive types, entity references, and finally strings. All the website data are cast to the chosen type.

Schema2QA needs to know the cardinality of each property, but that is not provided by Schema.org. The Builder uses a number of heuristics to infer cardinality from the property name, the type, as well as the documentation. Empirically, we found the heuristics work well in the domains we evaluated, with the exception of properties of the "Thing" class, such as "image" and "description", which are described as plural in the documentation, but have only one value per object in practice.

Generating per-property natural language annotations. The Builder automatically generates one annotation for each property, based on the property name and type. Camel-cased names are converted into multiple words and redundant words at the beginning and end are removed. The Builder uses a POS tagger and heuristics to identify the POS of the annotation. While the Builder will create a basic annotation for each property, developers are expected to add other annotations to improve the quality of synthesized sentences.

5 NEURAL SEMANTIC PARSING MODEL

Schema2QA uses a neural semantic parser to translate the user's question to an executable query in ThingTalk. Our model is a simple yet novel architecture we call BERT-LSTM. It is an encoder-decoder architecture that uses the BERT pretrained encoder [6] and an LSTM decoder with attention and a pointer-generator [1, 12, 20]. This section introduces the model and the rationale for its design. The overall architecture of the model is shown Fig. 3.

5.1 Encoding

Our model utilizes the BERT model [6] as the sentence encoder. We designed our encoder with the minimal amount of additional parameters on top of BERT, so as to make the most use of pretraining.

BERT is a deep Transformer network [21]. It encodes the sentence by first splitting it into *word-piece* subtokens, then feeding

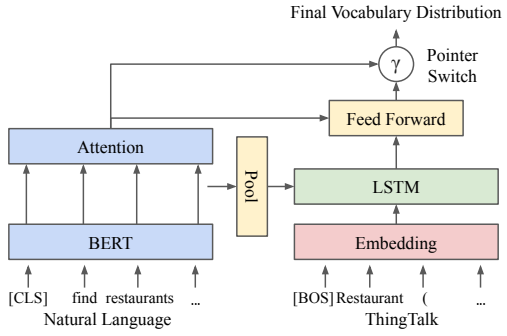

Figure 3: The BERT-LSTM model in Schema2QA.

them to a 12-layer Transformer network, to compute the final contextualized dense representations of each token h_{E}. BERT is pretrained on general English text using the *masked language model* objective. We fine-tune it on our semantic parsing task.

To compute a single vector representation $\bar{h}_{\mathrm{E}}$ for the whole sentence, the token representations produced by BERT are averaged, then fed to a one-layer feed-forward network:

$$h_{\mathrm{E}} = \mathrm{BERT}(x)$$
$$\bar{h}_{\mathrm{E}} = W_{\mathrm{pool,out}}\mathrm{relu}(W_{\mathrm{pool,in}}\mathrm{avg}(h_{\mathrm{E}}) + b_{\mathrm{pool}}),$$

where x is the input sentence and W and b are learnable parameters.

5.2 Decoding

During decoding, the model produces one token of the executable query y_t at time t, given the previously produced token y_{t-1}. There are no syntax-specific decoder components, and the model can generalize to any target language, including future extensions of ThingTalk, depending on the training data.

First, the previous token is embedded using a learned embedding. The embedded token is fed to an LSTM cell to compute the decoder representations $h_{\mathrm{D},t}$. These are then used to compute the attention scores s_t against each token in the encoder, and the attention value vector v_t and the attention context vector c_t:

$$\begin{aligned}
h_{\mathrm{D},0} &= \bar{h}_{\mathrm{E}} \\
c_0 &= \mathbf{0} \\
y_{\mathrm{emb},t} &= W_{\mathrm{emb}} y_{t-1} \\
h_{\mathrm{D},t} &= \mathrm{LSTM}(h_{\mathrm{D},t-1}, \left[y_{\mathrm{emb},t}; c_{t-1}\right]) \\
s_t &= \mathrm{softmax}(h_{\mathrm{D},t} h_{\mathrm{E}}^{\mathrm{T}}) \\
v_t &= \sum_{t'} s_{t,t'} h_{\mathrm{E},t'} \\
c_t &= \tanh(W_{\mathrm{att}} \left[v_t; h_{\mathrm{D},t}\right])
\end{aligned}$$

("emb" denotes embedding and "att" denotes attention).

The model then produces a vocabulary distribution $p_{t,w}$, where w is the word index in the vocabulary. $p_{t,w}$ is either a distribution over tokens in the input sentence (using a *pointer* network [22]), or a distribution over the vocabulary. The use of a pointer network allows the model to be mostly agnostic to specific entity names mentioned in the question, which are copied verbatim in the generated query. This allows the model to generalize to entities not seen in training. Conversely, the generator network allows the model to understand paraphrases and synonyms of the properties, as well as properties that are only mentioned implicitly.

We employ the pointer-generator network previously proposed by See et al. [20]. The choice of whether to point or to generate from the vocabulary is governed by a switch probability γ_t:

$$\begin{aligned}
\gamma_t &= \sigma(W_\gamma \left[y_{\mathrm{emb},t}; h_{\mathrm{D},t}; c_t\right]) \\
p_{t,w} &= \gamma_t \sum_{t', x_{t'}=w} s_{t,t'} + (1-\gamma_t)\mathrm{softmax}(W_{\mathrm{o}} c_t) \\
y_t &= \arg\max_w p_{t,w}
\end{aligned}$$

where W_{o} is the output embedding matrix.

The model is trained to maximize the likelihood of the query for a given question, using teacher forcing. At inference time, the model greedily chooses the token with the highest probability.

The model predicts one token of the query at a time, according to the tokenization rules of ThingTalk. To be able to copy input tokens from the pretrained BERT vocabulary, all words between quoted strings in the ThingTalk query are further split into word-pieces, and the model is trained to produce individual word-pieces.

6 EXPERIMENTAL RESULTS

In this section, we evaluate the performance of Schema2QA with experiments to answer the following: (1) How well does Schema2QA perform on popular Schema.org domains? (2) How does our neural network model compare with prior work? (3) How do our templates compare with prior work? (4) Can the knowledge learned from one domain be transferred to a related domain? (5) How do skills built with Schema2QA compare with commercial assistants?

As Schema2QA generates queries to be applied to a knowledge base, we evaluate on *query accuracy*, which measures if the generated query matches the ThingTalk code exactly. The answer retrieved is guaranteed to be correct, provided the database contains the right data.

In all the experiments, we use the same generic template library, which was refined over time, totally approximately 800 templates. It covers all the varieties described in Section 3, except choice and yes-no questions. Our experiments are conducted in English. Generalization to other languages is out of scope for this paper.

We experiment on five Schema.org domains: restaurant, people, movies, books, and music. We scrape Schema.org metadata from Yelp, LinkedIn, IMDb, Goodreads, and Last.fm, for each domain, respectively. We include (1) all properties of these domains in Schema.org for which data is available, and (2) useful properties of types which no data is needed for training, such as enum and boolean type. The number of properties used per domain is shown in Table 5. We end up writing about 100 annotations per domain, many of which result from error analysis of the validation data.

6.1 Schema2QA on Five Schema.org Domains

The size of the generated training data and the evaluation data is shown in Table 5. No real data are used in training. Furthermore, only less than 2% of the synthesized data are paraphrased by crowdsource workers, resulting in a low data acquisition cost. Realistic data, however, are used for validation and testing, as they have been shown to be significantly more challenging and meaningful than testing with paraphrases [5].

Table 5: Size of training, dev, and test sets.

		Restaurants	People	Movies	Books	Music
# of properties		25	13	16	15	19
# of annotations		122	95	111	96	103
Train	Synthesized	270,081	270,081	270,081	270,081	270,081
	Paraphrase	6,419	7,108	3,774	3,941	3,626
	Augmented	508,101	614,841	405,241	410,141	425,041
Dev	1 property	221	127	140	107	62
	2 properties	219	346	226	222	182
	3+ properties	88	26	23	33	82
	Total	**528**	**499**	**389**	**362**	**326**
Test	1 property	200	232	130	114	44
	2 properties	245	257	264	241	181
	3+ properties	79	11	19	55	63
	Total	**524**	**500**	**413**	**410**	**288**

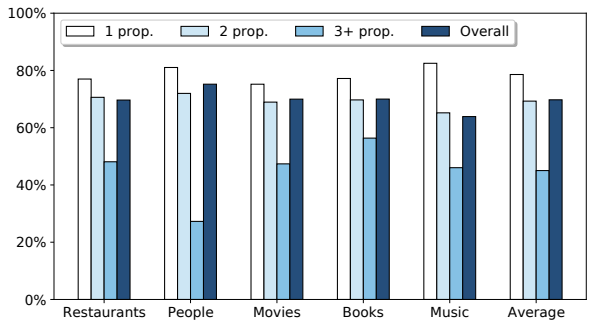

Figure 4: Query accuracy on crowdsourced questions on Schema.org domains.

For the dev and test data, crowdsource workers are presented with a list of properties in the target domain and a few examples of queries, and are asked to come up with 5 questions with either one property or two properties, with equal probability. A property is counted once for every mention in the query. E.g., "rating between 2 and 4" would be counted as two properties, because it is equivalent to "rating less than 4 and rating greater than 2". We observe that crowdsource workers generate questions that refer to three or more properties, despite being instructed to generate queries involving one or two. We do not show the workers any sentences or website data we used for training, and we allow them to freely choose the value of each property. The questions are then annotated by hand with their ThingTalk representation. The author who annotated the test set did not help tune the system afterwards.

We train and evaluate on each domain separately. The accuracy is shown in Fig. 4. On average, Schema2QA has a 69.7% overall accuracy, achieving 78.6%, 69.3%, and 45.0% accuracy on questions with one, two, three or more properties, respectively. Overall, this result, achieved without any real data in training, shows that Schema2QA can build an effective parser at a low cost. Additionally, developers can add more annotations to further increase the accuracy with little additional cost. Schema2QA is able to achieve reasonably high accuracy for complex queries because of the large synthesized training set, including many combinations of properties.

Table 6: Comparison of query accuracy across models.

	Restaurants	People	Movies	Books	Music	Avg
MQAN	65.1%	53.0%	62.0%	59.3%	51.0%	58.1%
BERT-LSTM	**69.7%**	**75.2%**	**70.0%**	**70.0%**	**63.9%**	**69.7%**

Table 7: Query accuracy of BERT-LSTM trained with only data synthesized from SEMPRE and Schema2QA templates.

	Restaurants	People	Movies	Books	Music	Avg
SEMPRE	3.5%	1.6%	4.8%	2.2%	0.7%	2.6%
Schema2QA	**63.2%**	**66.8%**	**63.2%**	**49.8%**	**57.3%**	**60.1%**

6.2 Neural Model Comparison

The previous state-of-the-art neural model capable of training on synthesized data and achieving good accuracy was MQAN (Multi-Task Question Answering Network) [5, 17]. It is an encoder-decoder network that uses a stack of self-attention and LSTM layers, and uses GloVe and character word embeddings. In this section, we evaluate how BERT-LSTM performs in comparison to MQAN.

As shown in Table. 6, the use of BERT-LSTM improves query accuracy by 11.6% on average. This shows that the use of pretraining in BERT is helpful to generalize to test data unseen in training.

6.3 Evaluation of Schema2QA Templates

One of the contributions of Schema2QA is a more comprehensive generic query template set than what was proposed originally by Sempre [24]. Here we quantify the contribution by comparing the two. We reimplement the Sempre templates using Genie, and apply the same data augmentation to both sets. We train with only synthesized data, and evaluate on realistic data. The result is shown in Table 7. On average, training with Sempre templates achieves only 2.6% accuracy. On the other hand, training with only synthesized data produced with Schema2QA templates achieves 59.5% accuracy. This result shows that the synthesized data we generate matches our realistic test questions more closely. Our templates are more tuned to understand the variety of filters that commonly appear in the test. Furthermore, due to the pretraining, the BERT-LSTM model can make effective use of synthesized data and generalize beyond the templates. These two effects combined means we do not need to rely as much on expensive paraphrasing.

6.4 Zero-Shot Transfer Learning to Hotels

Many domains in Schema.org share common classes and properties. Here we experiment with transfer learning from the restaurant domain to the hotel domain. Restaurants and hotels share many of the same fields such name, location, and rating. The Hotel class has additional properties "petsAllowed", "checkinTime", "checkoutTime", and "amenityFeature". For training data synthesis, we have manual annotations for fields that are *common* with restaurants, and for the rest, we use annotations automatically generated by Schema2QA. The paraphrases for the restaurant domain are automatically transferred to the hotel domain by replacing words like

Table 8: Comparing Schema2QA with three major commercial assistants on 10 queries about restaurants, people, and hotels.

Query	Google	Alexa	Siri	Schema2QA
Show restaurants near Stanford rated higher than 4.5	×	×	×	✓
Show me restaurants rated at least 4 stars with at least 100 reviews	×	×	×	✓
What is the highest rated Chinese restaurant in Hawaii?	×	✓	✓	✓
How far is the closest 4 star and above restaurant?	×	×	×	✓
Find a W3C employee that went to Oxford	×	×	×	✓
Who worked for both Google and Amazon?	×	×	×	✓
Who graduated from Stanford and won a Nobel prize?	✓	×	×	✓
Who worked for at least 3 companies?	×	×	×	✓
Show me hotels with checkout time later than 12PM	×	×	×	✓
Which hotel has a swimming pool in this area?	✓	×	×	✓

"restaurant" and "diner" with "hotel". We augment the training sets with data crawled from the Hyatt hotel chain.

We acquire an validation set of 443 questions and a test set of 528 questions, crowdsourced from MTurk, and annotated by hand. 205 of the test questions use one property, 270 use two properties, and 53 use three or more. On the test set, the generated parser achieves an overall accuracy of 64%. Note that about half of the test sentences use properties that are specific to the "Hotel" class. This shows that it is possible to bootstrap a new domain, similar to an existing one, with no manual training data acquisition.

SQA has also been used in a recent transfer-learning experiment[3] on the MultiWOZ multi-domain dialogue dataset [2]. A model for each domain was trained by replacing in-domain real training data with synthesized data and data automatically adapted from other domains. The resulting models achieve an average of 70% of the accuracy obtained with real training data.

6.5 Comparison with Commercial Assistants

For the final and most challenging experiment, we compare Schema2QA with commercial assistants on Restaurant domain. We can only evaluate commercial assistants on their *answer accuracy* by checking manually if the results match the question, since they do not show the generated queries. Answer accuracy is an upper bound of query accuracy because the assistant may return the correct answer even if the query is incorrect.

In the first comparison, we use the test data collected in Section 6.1. As shown in Fig. 5, Schema2QA gets the best result with a query accuracy of 69.7%; Siri has a 51.3% answer accuracy, Google Assistant is at 42.2%, and Alexa is at 40.6%. Unlike Schema2QA, commercial assistants are tuned to answer the more frequently asked questions. Thus, we perform another comparison where we ask crowdsource workers to ask any questions they like about restaurants, without showing them the properties in our database. Note that we only test the systems on questions that can potentially be answered using the Schema.org data. We collected 300 questions each for the dev and test set. Removing questions that cannot be answered with our subset of Schema.org, we have a total of 137 questions for dev and 169 for test. Out of the 169 test questions, 27 use one property, 89 use two, and 53 use three or more. The accuracy of Schema2QA drops from 69.7% in the first test to 59.8% in the second test, because the unprompted questions use a wider vocabulary. Nonetheless, its performance matches that of Google assistant, about 7.1% lower than Siri and 14.7% higher than Alexa.

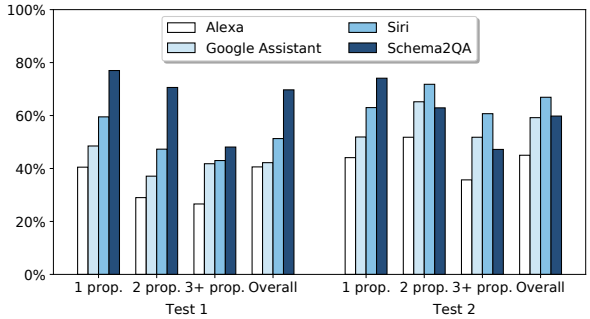

Figure 5: Comparison of commercial assistants (answer accuracy) and Schema2QA (query accuracy) on Restaurant domain. In Test 1, question writers see available properties, and in Test 2 they do not.

The answer accuracy of commercial assistants improves significantly in the second test. This is because they are tuned to answer popular commands, and can answer the question correctly even with limited understanding of the question. For example, by default they always return restaurants nearby. On the other hand, we measure query accuracy for Schema2QA, where the exact ThingTalk code needs to be predicted. Schema2QA can also apply the same heuristics to the returned answer, and we expect our answer accuracy, if measured on a large knowledge base, would be higher.

6.6 Discussion

Error analysis on the validation set in our experiments of Section 6.1 reveals that better named entity recognition can eliminate about half of the errors. An additional 14% seems to be resolvable with additional generic templates, an example of which is selecting two fields with the same value ("movies produced and directed by ..."). Fixing these two issues can potentially bring the accuracy from 70% to about 90% for in-domain questions. The results confirm our hypothesis that natural-language queries can be factored into a generic and a domain-specific part. Our methodology enables developers to iteratively refine both the generic question templates and domain-specific annotations. Furthermore, by cumulating generic question knowledge in the open-source template library, we enable developers to bootstrap an effective agent for a new domain in just a few days without any real training data.

7 CONCLUSION

This paper presents Schema2QA, a toolkit for building question-answering agents for databases. Schema2QA creates semantic parsers that translate complex natural language questions involving multiple predicates and computations into ThingTalk queries. By capturing the variety of natural language with a common set of generic question templates, we eliminate the need for manually annotated training data and minimize the reliance on paraphrasing. Developers only need to annotate the database fields with their types and a few phrases. Schema2QA also includes an NL-schema Builder tool that adapts Schema.org to Schema2QA, and provides a complete pipeline to build Q&A agents for websites.

Experimental results show that our BERT-LSTM model outperforms the MQAN model by 4.6% to 12.9%. The Q&A agents produced by Schema2QA can answer crowdsourced complex queries with 70% average accuracy. On the restaurant domain, we show a significant improvement over Alexa, Google, and Siri, which can answer at most 51% of our test questions. Furthermore, on common restaurant questions that Schema2QA can answer, our model matches the accuracy of Google, and is within 7% of Siri without any user data. By making Schema2QA available, we wish to encourage the creation of a voice web that is open to every virtual assistant.

ACKNOWLEDGMENTS

We thank Ramanathan V. Guha for his help and suggestions on Schema.org. This work is supported in part by the National Science Foundation under Grant No. 1900638 and the Alfred P. Sloan Foundation under Grant No. G-2020-13938.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473* (2014).

[2] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Ultes Stefan, Ramadan Osman, and Milica Gašić. 2018. MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[3] Giovanni Campagna, Agata Foryciarz, Mehrad Moradshahi, and Monica Lam. 2020. Zero-Shot Transfer Learning with Synthesized Data for Multi-Domain Dialogue State Tracking. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 122–132.

[4] Giovanni Campagna, Rakesh Ramesh, Silei Xu, Michael Fischer, and Monica S. Lam. 2017. Almond: The Architecture of an Open, Crowdsourced, Privacy-Preserving, Programmable Virtual Assistant. In *Proceedings of the 26th International Conference on World Wide Web - WWW '17*. 341–350.

[5] Giovanni Campagna, Silei Xu, Mehrad Moradshahi, Richard Socher, and Monica S. Lam. 2019. Genie: A Generator of Natural Language Semantic Parsers for Virtual Assistant Commands. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2019)*. 394–410.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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)*. 4171–4186.

[7] Li Dong and Mirella Lapata. 2016. Language to Logical Form with Neural Attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 33–43.

[8] Alessandra Giordani. 2008. Mapping Natural Language into SQL in a NLIDB. In *International Conference on Application of Natural Language to Information Systems*. Springer, 367–371.

[9] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 4524–4535.

[10] Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization. *arXiv preprint arXiv:1902.01069* (2019).

[11] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 963–973.

[12] Robin Jia and Percy Liang. 2016. Data Recombination for Neural Semantic Parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 12–22.

[13] Bret Kinsella. 2019. Amazon Alexa Has 100k Skills But Momentum Slows Globally. Here is the Breakdown by Country. https://voicebot.ai/2019/10/01/amazon-alexa-has-100k-skills-but-momentum-slows-globally-here-is-the-breakdown-by-country/. *Voicebot.ai* (October 2019).

[14] Thomas Kollar, Danielle Berry, Lauren Stuart, Karolina Owczarzak, Tagyoung Chung, Lambert Mathias, Michael Kayser, Bradford Snow, and Spyros Matsoukas. 2018. The Alexa Meaning Representation Language. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 3 (Industry Papers)*. 177–184.

[15] Anjishnu Kumar, Arpit Gupta, Julian Chan, Sam Tucker, Björn Hoffmeister, and Markus Dreyer. 2017. Just ASK: Building an Architecture for Extensible Self-Service Spoken Language Understanding. (2017). arXiv:1711.00549

[16] Fei Li and HV Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proceedings of the VLDB Endowment* 8, 1 (2014), 73–84.

[17] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The Natural Language Decathlon: Multitask Learning as Question Answering. *arXiv preprint arXiv:1806.08730* (2018).

[18] Vittorio Perera, Tagyoung Chung, Thomas Kollar, and Emma Strubell. 2018. Multi-Task Learning for parsing the Alexa Meaning Representation Language. In *American Association for Artificial Intelligence (AAAI)*. 181–224.

[19] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towardsa Theory of Natural Language Interfaces to Databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*. 149–157.

[20] Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 1073–1083.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[22] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Advances in Neural Information Processing Systems*. 2692–2700.

[23] Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust Text-to-SQL Generation with Execution-Guided Decoding. *arXiv preprint arXiv:1807.03100* (2018).

[24] Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a Semantic Parser Overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 1332–1342.

[25] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, et al. 2020. DBPal: A Fully Pluggable NL2SQL Training Pipeline. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2347–2361.

[26] Xiaojun Xu, Chang Liu, and Dawn Song. 2017. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. *arXiv preprint arXiv:1711.04436* (2017).

[27] Semih Yavuz, Izzeddin Gur, Yu Su, and Xifeng Yan. 2018. What It Takes to Achieve 100% Condition Accuracy on WikiSQL. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 1702–1711.

[28] Pengcheng Yin and Graham Neubig. 2018. TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 7–12.

[29] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. 588–594.

[30] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 3911–3921.

[31] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv preprint arXiv:1709.00103* (2017).