Efficient Deployment of
Conversational Natural Language Interfaces over Databases

Anthony Colas[*], **Trung Bui**[†], **Franck Dernoncourt**[†], **Moumita Sinha**[†], **Doo Soon Kim**[†]
University of Florida[*], Adobe Research[†]
`acolas1@ufl.edu`[*],
`{bui, franck.dernoncourt, mousinha, dkim}@adobe.com`[†]

Abstract

Many users communicate with chatbots and AI assistants in order to help them with various tasks. A key component of the assistant is the ability to understand and answer a user's natural language questions for question-answering (QA). Because data can be usually stored in a structured manner, an essential step involves turning a natural language question into its corresponding query language. However, in order to train most natural language-to-query-language state-of-the-art models, a large amount of training data is needed first. In most domains, this data is not available and collecting such datasets for various domains can be tedious and time-consuming. In this work, we propose a novel method for accelerating the training dataset collection for developing the natural language-to-query-language machine learning models. Our system allows one to generate conversational multi-term data, where multiple turns define a dialogue session, enabling one to better utilize chatbot interfaces. We train two current state-of-the-art NL-to-QL models, on both an SQL and SPARQL-based datasets in order to showcase the adaptability and efficacy of our created data.

1 Introduction

Chatbots and AI task assistants are widely used today to help users with their everyday needs. One use for these assistants is asking them questions on various areas of knowledge or how to accomplish different tasks (Braun et al., 2017; Cui et al., 2017). Because data is usually stored in a structured database, in order to answer a user's questions, it is essential that the system should first understand the question, and convert it into a structured language query, such as SQL or SPARQL, to fetch the correct answer.

While much research has focused on translating natural languages into query lan-

Figure 1: Example illustrating a three-turn dialogue, featuring the natural language (first column) and query language (second column) representations.

guages (Ngonga Ngomo et al., 2013; Braun et al., 2017; Dubey et al., 2016; Giordani and Moschitti, 2009; Finegan-Dollak et al., 2018; Giordani, 2008; Xu et al., 2017; Zhong et al., 2017), the state-of-the-art systems typically involve a large amount of training data. Therefore, in order to fully utilize these models that translate a natural language (NL) question into query language (QL), one would need to collect large amounts of both NL-QL pairs. Although there are works which involve the collection of NL-QL pairs in different domains (Hemphill et al., 1990; Zelle and Mooney, 1996; Zhong et al., 2017; Yu et al., 2018, 2019b), data is still not available in most domains, and thus this collection process can be both time-consuming and expensive.

In this work, we address the problem of having insufficient data collection methodologies by proposing a novel approach that accelerates the data collection process for use in NL-to-QL models. Additionally, our approach focuses on generating conversation data, where the context of a dialogue turn is used to generate a subsequent pair. In this way, we better simulate the data necessary for real world chatbots and voice assistants, as exemplified in Figure 1. Our contributions are as follows:

- We develop a novel approach that accelerates the creation of NL-to-QL data pairs. Primarily, our approach tackles the problem in the conversational domain.

- We showcase our data collection system on two different QLs, SQL and SPARQL, demonstrating the flexibility of our system.

- Finally, we demonstrate the use of current single-turn state-of-the-art approaches on these two domains to prove the adaptability of our system to current models.

Though our data collection implementation focuses on conversational data, the models we deploy are single-turn. Our main focus here is to give a demonstration of the generated data. Section 3 and Section 4 show the adaptability of our data collection scheme to these kinds of models.

The rest of this paper is structured as follows: Section 2 surveys prior work in both the NL-to-QL and data collection space, Section 3 details our novel conversational data collection approach, Section 4 walks through examples in both the SQL and SPARQL domain, Section 5 describes the current models we have trained and tested on the generated data, Section 6 gives the results on the data and models, and Section 7 concludes our work.

2 Related Work

In the field of natural language interfaces for structured data there are bodies of work that 1) focus on translating natural language to a specific query language and that 2) relate to collecting semantic parsing data for natural language interfaces.

2.1 NL-to-QL

NL-to-QL models have worked to transform natural language queries into their respective logical form (LF) representations (Dong and Lapata, 2016), SQL queries (Xu et al., 2017; Zhong et al., 2017; Finegan-Dollak et al., 2018; Cai et al., 2018), or SPARQL queries (Ngonga Ngomo et al., 2013; Dubey et al., 2016). While work in the SPARQL domain first normalize and match the queries, state-of-the-art work in translating NL to SQL involves neural architectures. Dong and Lapata (2016) utilize and encoder-decoder framework to translate NL questions into their LF representation. Xu et al. (2017) propose a sketch-based model where a neural network predicts each slot of the sketch. The ar-

chitecture built by Zhong et al. (2017) uses policy-based reinforcement learning in order to translate NL to SQL. While Finegan-Dollak et al. (2018)'s main takeaway is how different evaluations effect the generalization problem in translating NL to SQL, they approach the problem with a seq2seq model. Because of the volume of data needed to fully utilize these models, it can be difficult to adapt to different domains.

In the multi-turn domain, Saha et al. (2018) first approach the problem of complex sequential question-answering (CSQA) by first building a large-scale QA dataset made to answer questions found in Wikidata [1]. However, their data collection process was extremely laborious, as their process required in-house annotators, crowdsourced workers, and multiple iterations. Additionally, their approach was end-to-end, meaning the output was an expected answer. Nevertheless, because their approach incorporate the query representation, we plan to further incorporate their approach into our data collection process in future work. Yu et al. (2019a) also develop the first general-purpose DB querying dialogue system. However, their system dialogues focus on clarifying a NL question for user verification, before returning an answer. Our work focuses on generating conversational data about specific database entities and properties.

2.2 Data Collection for Semantic Parsing

NL question semantic parsers have been developed for single-turn QA in order to translate simple NL questions into their respective LFs (Wang et al., 2015). In their approach, Wang et al. (2015) first begin with a *domain*, building a seed lexicon of that domain. Next, they find the LF and canonical utterance templates corresponding based on the lexicon. Wang et al. (2015) then paraphrase their canonical utterances via crowd-sourcing. Iyer et al. (2017) learn a semantic parser via an encoder-decoder model by using NL/SQL templates. This model is tuned through user feedback, where incorrect queries are annotated by crowd-workers. Paraphrasing is accomplished through the Paraphrasing Database (PPDB) (Ganitkevitch et al., 2013).

While the two previously mentioned works are single-turn semantic parsers, Shah et al. (2018) develop a multi-turn semantic parser. Their approach begins with a task schema and API which is used to create dialogue outlines for the provided domain.

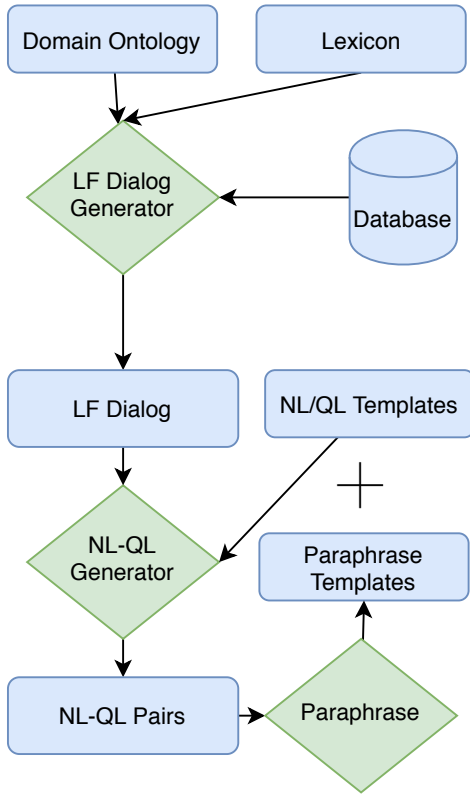[1] https://www.wikidata.org/wiki/Wikidata:Main_Page

Figure 2: An overview of our conversational data collection deployment system. Blue shapes denote the input/output data at each stage, while green diamonds denote the processes of the system. The "plus" sign denotes the concatenation of both seed templates and paraphrase templates.

These dialogue outlines involve a user and system bot that simulate a scenario. The dialogues are then paraphrased via crowd-sourcing. However, Shah et al. (2018) use the logical-form representation of the utterances rather than their query language representation. In our work, we re-incorporate the paraphrases into the dialogue generation phase.

3 Data Collection System

Our conversational data collection strategy is developed to efficiently collect NL/QL pairs for training data in models which translate the NL into QL in a multi-turn setting. Because domain data is required when training a chatbot to query a database when converting from NL to QL, our approach is generalized so that one can easily collect data for their respective domain.

3.1 Overview

Our approach in collecting data is made of the four following steps: 1) First we generate the dialogue represented as LFs, forming the abstract represen-

tations of NL questions, 2) Next, we convert the LFs into an NL template and QL templates 3) We then collect paraphrases of the natural language templates, and 4) Finally, we use these paraphrases to further develop our dialog generator. In generating our dialogue, the context of each previous turn is taken in order to develop the current turn. Figure 2 presents our data deployment system. We divide and expand upon the steps further in the next sections.

3.2 Definitions

We first define the following notations in our data collection system:

- U_n: an utterance in the dialogue.

- LF_n: the LF n in the dialogue.

- NL_n: the NL utterance corresponding to LF_n.

- QL_n: the QL utterance corresponding to LF_n.

3.3 Input Module

The input to our data collection system consists of a domain ontology, lexicon, and database. These should be provided by the user and vary depending on the type of data one requires. The domain ontology defines the <*object, relation, property*> triples of a given dataset, where each object has a set of properties connected through a relation, e.g. <*ACL 2020, has_location, Seattle*>. The lexicon file defines each data field, along with its NL and QL representation, important in the NL-QL Generator step. The database is the data in structured form.

3.4 Logical Form Dialogue Generator

In order to appropriately simulate a conversation between a user and chatbot, the synthetic dialogue must first be generated. This is done by first outlining the dialog via LFs, where the system generates, LF_{1-n}. These outlines are an abstract but understandable representation of the dialogue taking into account the type, entity, and relation of a question. Thus, our parser builds a dialogue based on a domain ontology, lexicon, and domain database.

The LFs take the form of three predicates: *Retrieve-Objects*, *Inquire-Property*, and *Compute*, each taking on their own arguments. For the *Retrieve-Objects* predicate, the LF fetches an instance that satisfies a condition. As arguments,

Retrieve-Objects takes an *entity type*, $\mathbf{t_n^i}$ from the ontology, a boolean *condition* $\mathbf{c_n^i}$, and a *property value*, $\mathbf{p_n^i}$, from the DB. For the *Inquire-Property* predicate, given an *anchor entity* $\mathbf{ae_n^i}$, *target instance*, $\mathbf{ti_n^i}$, and an *inference path* $\mathbf{ip_n^i}$ from the entity to that instance, the LF finds the property in that path of the anchor entity. The *Compute* predicate denotes a *computation* $\mathbf{comp_n^i}$ over a set of given objects, thus its arguments are comprised of *Retrieve-Objects* arguments and an operation to be performed. [2]. For our work, we focus on using the *COUNT* aggregate function. Future work can easily adapt more aggregate functions into our model such as *MAX* or *MIN* depending on the values contained in the database.

More formally, each LF can be described as follows:

$$LF_n \rightarrow \{Retrieve - Objects(t_n^i, c_n^i, p_n^i),$$
$$Inquire - Property(ae_n^i, ti_n^i, ip_n^i),$$
$$Compute(comp_n^i, t_n^i, c_n^i, p_n^i)\} \quad (1)$$

At the start of a dialogue, a random LF predicate is selected, given the database schema, lexicon, and domain ontology. The subsequent turns in the dialogue are built conditionally on the previous turn. Therefore, given a LF_{n-1}, when generating LF_n the context of LF_{n-1} is further taken into consideration including its arguments, type, and answer. The subsequent predicate is also chosen at random, however its values are conditional on the arguments and answer(s) of the current predicate. For example, if LF_{n-1} is an *Retrieve-Objects* predicate and another *Retrieve-Objects* predicate is chosen as LF_n, this LF can further filter the answer of LF_{n-1} by using an additional condition. Table 1 summarizes the types of LFs, along with an explanation and example of each both in LF and NL, which we discuss in the next section.

3.5 NL-QL Generator

Once the LF generator is complete, the data collection system generates an NL utterance along with its corresponding QL. To generate such pairs, the NL-QL generator takes in each LF from the LF Dialog as input. Based on the predicate type, an NL-QL pair is selected and filled with corresponding arguments of the predicate. Thus, the system uses NL seed templates for the *Retrieve-Objects*,

Inquire-Property, and *Compute* predicates to create the initial training data for the conversational dialogue. For example, one NL template for turns after NL_1 can be *"How about <entity>?"*

The aforementioned seed templates are hand-crafted based on the type of data and are thus left to the user to create. These data are hand-crafted to increase the quality of the seed templates in terms of coherency and utility, important features not only for quality training data, but also when performing the paraphrase task. Because we hand-crafted the query language templates, we also guarantee that the queries are executable for their corresponding QLs, SQL or SPARQL in this work. For the QL, we fill in slots for field names, aliases, and values, utilizing the information in the domain ontology, lexicon, and database schema. Note, 'field' refers to column names in relational DBs (queried with SQL) and type names in graph DBs (queried with SPARQL). To reiterate, the NL-QL generator takes each LF_n, with its respective arguments, and seed templates as input, and outputs a $NL_n - QL_n$ pair, where $U_n \rightarrow (NL_n, QL_n)$. Section 4 goes through detailed examples of various NL-QL pairs.

3.6 Paraphrase

The final step involves the paraphrasing of the seed NL templates given in the NL-QL Generator step. To paraphrase the seed NL templates, we first provide crowdworkers from Amazon Mechanical Turk (AMT) [3] with the instantiated templates, the output from the first iteration of the NL-QL generator. We ask the workers to paraphrase the seed templates while keeping the meaning/intent of the original questions. After collecting these paraphrased questions, we further abstract them and link them to their respective predicate representation. In this way, the paraphrases can be utilized in further iterations of the NL-QL Generator step and instantiated when generating new dialogues for training data. While abstracting the templates, we manually scan them for quality control purposes. Furthermore, we ran multiple trial runs in presenting the problem to the AMT workers. Previous work (Wang et al., 2015; Shah et al., 2018) also use similar crowdsourcing techniques in order to paraphrase their templates. Via AMT, Wang et al. (2015) paraphrase canonical utterance, natural language representations to single-turn LFs, while Shah et al. (2018) paraphrase dialogue outlines as their final step.

---

[2]$\mathbf{n}$ refers to the dialogue turn, while $\mathbf{i}$ refers to the number of dialogue generated.

[3]https://www.mturk.com/

Predicate	Explanation	Example	LF
Retrieve-Objects	Gets objects from DB	Which employees have building no. equal to 5?	Retrieve-Objects(employee(ALL), (employee.building_no,'=',5))
Inquire-Property	Gets an object's property	What is the office of James?	Inquire-Property(James, office)
Compute	CompuAggregate function	How many employees have hire year equal to 2010?	Compute(COUNT, employee(ALL), [('hire_year', 2010)])

Table 1: LF predicate summary with an explanation and example of each, both in NL and LF.

Similarly to Shah et al. (2018), we input the paraphrases back into our NL-QL generation step. Figure 2 illustrates this through the "+" symbol, signifying that the paraphrases are appended to the seed templates when mapping to LF and creating the final NL-QL pairs. This approach can take multiple iterations, as the user sees fit to the NL question generation task in their data domain.

4 Data Examples

In this section we will showcase examples in both the SQL and SPARQL domain and traverse through each stage of our Data Collection System. We first begin with SQL, used to query relational databases, and then demonstrate our system with a graph querying language, SPARQL. By doing so, we show the extendability of our approach to various structured QLs. Moreover, we confirm the importance of generating executable queries in a conversational data collection system.

4.1 SQL

Through our data collection system for conversational QA, we are able to produce contextual dependent NL-SQL pairs. For the SQL example, suppose a user wants to produce data for an employee directory relational database. Figure 3 gives an example of possible input files needed to produce this kind of conversational data with our data collection system, including a domain ontology with two entities *Employee* and *Department*, a lexicon to map NL and QL instances, and a database containing *Employee* and *Department* data.

Thus, given the input files in Figure 3, possible LF_n values with each predicate are:

(i) Retrieve-Object(employee(ALL), (employee.dept_name,'=', Marketing))

(ii) Inquire-Property(James,dept_name)

(iii) Computation(COUNT,employee(ALL), [('works_in', 'IT')])
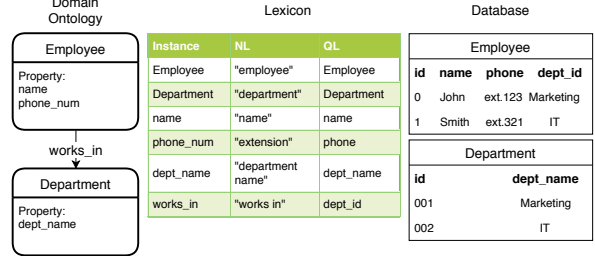
Figure 3: Example ontology schema, lexicon, and database. The two tables in the Database are used throughout our SQL example.

In (i), the logical form represents a retrieval of employee objects who work in the Marketing department. (ii) asks about the department name of James. (iii) computes the total number of employees who work in the IT department. During the generation of LF_1, one of these LFs can be generated. Then for LF_2 - LF_n, the context is passed along to generate the LFs. The *n* denotes the number of turns a dialogue can take. As an example, given LF_1 is (1) from the aforementioned LFs, LF_2 can be *Inquire-Property(Answer,phone_num)*, where *Answer* denotes the objects returned by LF_1. Our dialogue generation system allows one to tune the number of turns and number of dialogues generated from the given input.

For the NL-QL step, our input includes the dialogues represented as LFs along with NL-QL seed templates described in Section 3.5. Possible templates are given in Table 2. Note, that we refer to a column in a relational DB as a field. Taking our previous *Retrieve-Objects* example, the filled seed template would read: "Which employee have department equal to Marketing?" The Lexicon from Figure 3 is utilized here, as the instance name is mapped to its NL name. Similarly, its QL name (table name) is mapped in the SQL query.

Finally, in the final step, as explained in 3.5, the NL seed templates are paraphrased via crowdsourcing, e.g. "Which employee have department equal to Marketing?" can be paraphrased into "Who works in the marketing department?".

Predicate	Template
Retrieve-Object	Which <entity>have <field name> equal to <instance>?
Inquire-Property	What is the <field name> of <entity value>?
Computation	How many <entity>have <field name>equal to <instance>?

Table 2: Examples of seed templates with their respective predicates. <entity>refers to an entity type. <field name>corresponds to a column in a relational DB or a relation in a graph DB. <instance>refers to the value of that field in the DB. <entity value>is an instance of an entity in the DB.

Figure 4: An example of a subgraph in the Photoshop Knowledge Graph. The Layer object (red node), can be seen connected to its objects (blue nodes) through relations. Here we can see that the Layer entity is connected to the various actions connected to "Photoshop Layers", such as "flatten", "lock", and "use", where the object nodes show how they can be performed.

4.2 SPARQL

SPARQL is used to query graph databases, where entities are linked together through relations. These graph databases usually take the form of triples in the form: <*subject,relation,object*>. Because both LF-Generator and NL-QL Generator remain the same as in Section 4.1, here we examine the main differences in the system data when utilizing SPARQL instead of SQL. As a guide, we refer to the example give in Figure 4.

Figure 4 gives an example of a subgraph found in the Photoshop Knowledge Graph (KG). This KG contains the various tools, dialogs, shortcuts, and options found in Photoshop, connected to their options and definitions through relations. The KG is extracted from the Photoshop Wiki. Similarly to the SQL example above, we input a domain on-

tology, lexicon, and database to the conversational data collection system. However, in the case of a graph database, the entities found in the ontology are more clearly defined in a graph database. Additionally, instead of a table structure, the database is in the form of <*subject,relation,object*> triples, where each entity belongs to a *type* defined in the ontology.

While the the types of LFs generated in the LF-Generator are equivalent, a *property* now refers to the relation found in the triple, while a property refers to the object of a KB triple. For example, an entity such as the one found in figure 4 may have various properties, including "has_shortcut" and "has_option". When generating NL-QL pairs, the generator again takes from the out of the LF-Generator, lexicon, and seed templates, where the QL template is SPARQL-based instead of SQL-based. Paraphrases are collected in the same way. Thus, an example Photoshop Retrieve-Object LF template question, and paraphrase may look like: "LF: Retrieve-Objects(tool(ALL), (tool.has_shortcut, '=', 'H))", "Template: Which ⟨ entities⟩ have ⟨ relation⟩ equal to ⟨ object⟩?", and "Paraphrase: What's the tool with the H shortcut?"

5 Experiments

We will now examine our experiments with a relational and graph database setting. We first briefly discuss the data used in constructing the conversational dataset and then describe the various models utilized in translating the NL questions into their respective structured queries.

5.1 Data

For our experiments involving SQL data, we construct an NL-QL conversational dataset on data based on a proprietary web analytics tool. In our results table, we refer to this dataset as *Web-Analytics*. For the graph-database, we construct an NL-QL conversational dataset based on the Photoshop KB, as the one exemplified in Section 4.1. As previously noted, this KB contains various entities found in Photoshop, connected to their properties, through predicates which define the properties. In total, the KB contains 15,381 triples, with 3,410 triples that correspond to how-to type queries.

After running our conversational data collection system on both set of data, we collected 288 and 73 NL-QL pairs of templates for the Photoshop

	Photoshop	Web-Analytics
Templates	288	73

Table 3: Number of templates for each dataset, where the Photoshop dataset is SPARQL-based and Web-Analytics dataset is SQL-based.

and Web-Analytics datasets, respectively. Table 3 summarizes these statistics. Additionally, we configured our system to give 3 turn dialogues.

5.2 Models

In our experiments we utilize single-turn NL-QL models. Specifically, we utilize the baselines defined by Finegan-Dollak et al. (2018).

The first baseline is a seq2seq model with attention-based copying, originally proposed by Jia and Liang (2016). This model takes an NL utterance as input and outputs a structured query. Included in the output is a COPY token, which signifies the copying of an input token. In the copying mechanism model, the loss is calculated based on the accumulation of both the probability of distribution of the tokens in the output and the probability of copying from an input token. This copying probability is calculated as the categorical cross entropy of the distributed attention scores across the input's tokens, where the token with the max attention score is chosen as the output token.

The second baseline is a template-based model developed by Finegan-Dollak et al. (2018). This model takes in natural language questions, along with query templates to train. Since our data collection system directly utilizes templates to generate the data, this model is easily adaptable to our setting. We simply use the templates we collect from both the seed-templates and paraphrasing tasks, as well as the slot values extracted from the source DB when creating the dialogue data to train the model. In the template-based model, there are two decisions being made. First the model selects the best template to choose from the input. This is done by passing the final hidden states of a bi-LSTM through a feed-forward neural network. Next, the model selects the words in an input NL-question which can fill the template slots. Again, the same bi-LSTM is used to predict whether an input token is used in the output query or not. Thus, given a natural language question, the model jointly learns the best template from the given input, as well as the values that fill the template's slots. Please note, that while this model is best fitted for our dataset,


```
SELECT ?entity ?property where {
?entity rdf:type ontology:ps_entity .
?entity ontology:sharpen ?property .
?entity rdfs:label "ps_entity0"@en}
```
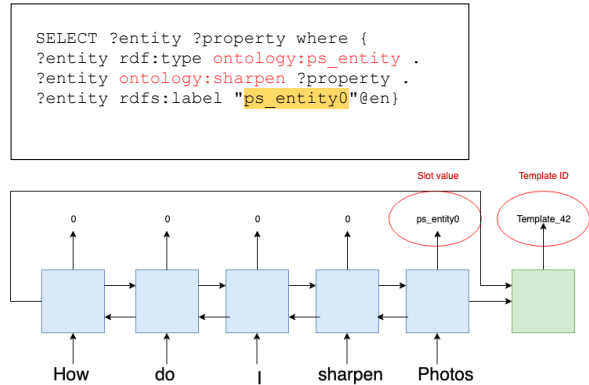
Figure 5: The template-based model developed by Finegan-Dollak et al. (2018), where the blue boxes represent LSTM cells and the green box represents a feed-forward neural network. 'Photos' is classified as a slot value, while the template chosen (Tempalte_42), is depicted above the model. In the template, the entity slot is highlighted in yellow and the properties which make the template unique are in red.

it does not generalize well to data outside of the trained domain due to the template selection task. Figure 5, inspired by Finegan-Dollak et al. (2018), shows an example of the template-based model with our own input in the SPARQL domain.

Although our dataset collection system generates multi-turn data, because of the immaturity of multi-turn NL-to-QL models, we leave the use of multi-turn models for future work. We do however, mention the model developed by Saha et al. (2018), which answers complex sequential natural language questions over KBs, which can be further integrated in future work.

5.3 Settings

We experimented with both the seq2seq and template-based models on the SQL-based and SPARQL-based datasets previously discussed. For the Photoshop SPARQL dataset, we generated *2,100* single-turn data pairs utilizing our data collection system, while generating *3,504* single-data pairs for the web-analytics dataset. Experiments all used a 90/10 train/validation set split.

6 Results

We evaluated the models on our generated datasets for exact-match accuracy of the SQL/SPARQL output queries. The results (shown in Table 4) indicate that in both cases the seq2seq model outperforms the template-based model. While the seq2seq gives an accuracy of .726 and .738, the template-based
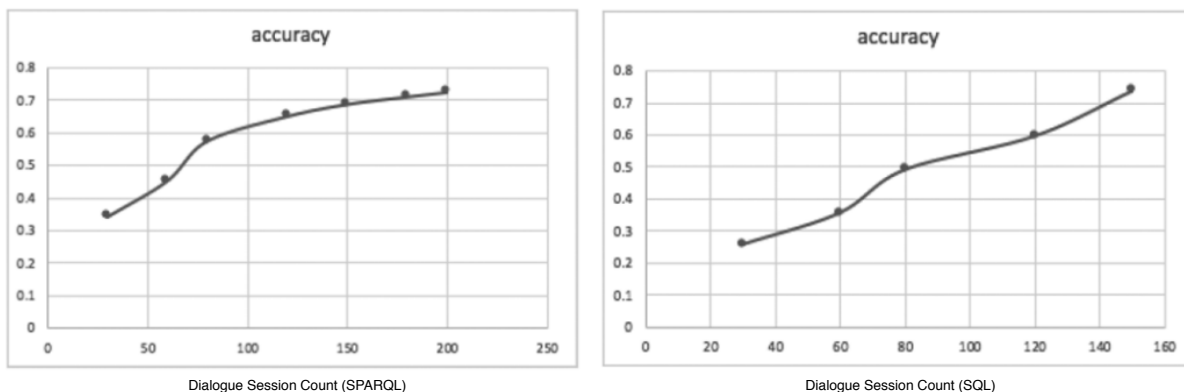
Figure 6: The above graphs show that as the dialogue session count increases for both the Photoshop SPARQL (left) and Web-Analytics SQL (right) dataset, the accuracy also increases. The y-axis of each graph marks the accuracy, while the x-axis marks the number of dialogue sessions for each dataset.

	Photoshop	Web-Analytics
Seq2seq	.726	.738
Template-based	.305	.641

Table 4: Results on the accuracy of the NL-to-QL task on the generated single-turn Photoshop and Web-Analytics datasets.

model results in .305 and .641 accuracy. Furthermore, the template-based model performs better on the Web-Analytics SQL-based dataset. This may be because the number of templates contained in the SQL dataset is almost four times greater than the number of templates contained in the Photoshop SPARQL dataset, 73 compared to 288.

We also investigate how the accuracy of the models increase, as the number of samples generated by our data collection system increase. Figure 6 shows that for our best performing model (seq2seq), as the number of dialogue sessions (or data points) increases, the accuracy increases. While this is expected, it also shows that through out dialog creation system, one can improve their NL-to-QL application's performance by configuring the data creation system with more dialogues and templates.

Though the models use synthetic data generated by our system, our system allows one to accelerate the data collection process and quickly deploy an NL-to-QL system that gives reasonably accurate results. This deployed system can then later collect data collected from real application users, where the application logs where a correct or incorrect response may have been returned. Iyer et al. (2017) explore this kind of work which learns from user feedback, where users marked utterances as cor-

rect or incorrect, and the accuracy of the semantic parser increased as a result.

7 Conclusion

In this work, we propose a conversational data collection system which accelerates the deployment of conversational natural language interface applications which utilize structured data. We describe the three main processes of our system, including the *LF Dialog Generator*, the *NL-QL Generator*, and the *Paraphrase* component. By taking in a domain ontology, lexicon, and structured database as input, our system generates NL-QL multi-turn pairs which can be used to train systems that translate NL to QL. Each component of our system is examined in both the SQL and SPARQL QL domain. We then validate our data by training state-of-the-art NL to QL models on single-turn utterances. Our experiments show promising results in both the SQL and SPARQL domains, while providing an efficient method to generate data for the development of multi-turn models.

References

Daniel Braun, Adrian Hernandez Mendez, Florian Matthes, and Manfred Langen. 2017. Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 174–185.

Ruichu Cai, Boyan Xu, Zhenjie Zhang, Xiaoyan Yang, Zijian Li, and Zhihao Liang. 2018. An encoder-decoder framework translating natural language to database queries. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3977–3983.

Lei Cui, Shaohan Huang, Furu Wei, Chuanqi Tan, Chaoqun Duan, and Ming Zhou. 2017. SuperAgent: A customer service chatbot for e-commerce websites. In *Proceedings of ACL 2017, System Demonstrations*, pages 97–102, Vancouver, Canada. Association for Computational Linguistics.

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)*, pages 33–43.

Mohnish Dubey, Sourish Dasgupta, Ankit Sharma, Konrad Höffner, and Jens Lehmann. 2016. Asknow: A framework for natural language query formalization in sparql. In *European Semantic Web Conference*, pages 300–316. Springer.

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

Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. Ppdb: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764.

Alessandra Giordani. 2008. Mapping natural language into sql in a nlidb. In *International Conference on Application of Natural Language to Information Systems*, pages 367–371. Springer.

Alessandra Giordani and Alessandro Moschitti. 2009. Semantic mapping between natural language questions and sql queries via syntactic pairing. In *International Conference on Application of Natural Language to Information Systems*, pages 207–221. Springer.

Charles T Hemphill, John J Godfrey, and George R Doddington. 1990. The atis spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

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)*, pages 963–973.

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)*, pages 12–22.

Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, Christina Unger, Jens Lehmann, and Daniel Gerber. 2013. Sorry, i don't speak sparql: translating sparql queries into natural language. In *Proceedings of the 22nd international conference on World Wide Web*, pages 977–988.

Amrita Saha, Vardaan Pahuja, Mitesh M Khapra, Karthik Sankaranarayanan, and Sarath Chandar. 2018. Complex sequential question answering: Towards learning to converse over linked question answer pairs with a knowledge graph. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Pararth Shah, Dilek Hakkani-Tür, Gokhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry Heck. 2018. Building a conversational agent overnight with dialogue self-play. *arXiv preprint arXiv:1801.04871*.

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

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. 2019a. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979.

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*, pages 3911–3921.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, et al. 2019b. Sparc: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries

from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.