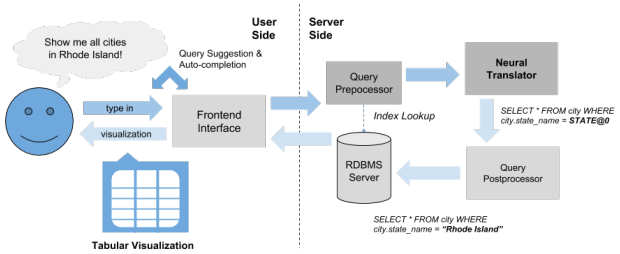

Figure 2: An overview of the DBPal system.

spectrum of pairs of SQL and natural language queries. To further extend the spectrum of natural language variations, we rely on existing language models learned from large text corpora such as Wikipedia to automatically paraphrase and add noise to the training set.

Interactive Query Auto-Completion. We provide real-time auto-completion and query suggestion to help users who may be unfamiliar with the database schema or the supported query features. This helps our system improve translation accuracy by leading the user to enter less ambiguous queries. Consider a user exploring a US geographical information database and starting to type *"show me the names "* — at this point, the system suggests possible completions such as *of states*, *of rivers*, or *of cities* to make the user aware of the different options she has. The core of the auto-completion feature is a language model based on the same sequence-to-sequence model and trained on the same dataset as the query translator.

A screenshot of our prototype of DBPal — which implements the aforementioned features — is shown in Figure 1. We also recommend the readers to watch the video[1] which shows a recording of a representative user session. The remainder of this paper is organized as follows: We first discuss related prior work in Section 2. In Section 3, we introduce the system architecture of DBPal. Then, we describe the system demonstration scenario in 4. Lastly, in Section 5 we discuss some limitations of our current prototype and planned future extensions.

2 RELATED WORK

NLIDBs have been studied in the database research community since the 1990's [1, 7]. Most of this work relied on classical techniques for semantic parsing and used rule-based approaches for the translation into SQL. However, these approaches have commonly shown poor flexibility for the users who use questions with different linguistic styles using paraphrases and thus failed to support complex scenarios.

More recent approaches tackled some of these limitations. For example, the system ATHENA [8] relies on a manually crafted ontology that is used to make query translation more robust by taking different ways of phrasing a query into account. However, since ontologies are domain-specific, they need to be hand-crafted for every new database schema. On the other hand, the system NaLIR [5] relies on an off-the-shelf dependency parser that could also be built on top of a deep model. However, it still implements a rule-based system that struggles with variations in vocabulary and syntax. Our system attempts to solve both of those issues by being both domain independent and robust to grammatical alterations.

[1] https://vimeo.com/user78987383/dbpal

Furthermore, some recent approaches leverage deep models for end-to-end translation similar to our system (e.g., [4]). However, a main difference between our system and [4] is that their approach requires manually handcrafting a training set for each novel schema/domain that consist of pairs of natural language and SQL queries. In contrast, our approach does not require a hand-crafted training set. Instead, inspired by [12], our system generates a synthetic training set that requires only minimal annotations to the database schema. It should be noted that "no need for manual input" does not mean that our approach does not need training when switching to a schema.

Finally, none of the above-mentioned approaches combine their translation pipelines with additional functions such as auto-completion. These features not only make query formulation easier by helping users to phrase questions even without knowing the database schema, but they also help users to write less ambiguous queries that can be more directly translated into SQL.

3 SYSTEM ARCHITECTURE

The DBPal system consists of three general components: a web-based user interface, a novel query translator, and an interactive auto-completion feature to help users to phrase questions as shown in Figure 2. In the following, we first describe the design of each of these components.

3.1 User Interface Design

We designed a web-based interface that allows users to explore the data in a given database via natural language questions or commands. The interface guides users to perform the exploration in two ways: first, it serves the database schema information and its stored records as tabular views to help users be aware of the accessible information. Second, it applies the auto-completion function as users type in their questions into the form. A primary goal of this input suggestion or auto-completion feature is to steer a user to write an English utterance that is more likely to be correctly translated to SQL by our system. Once the natural language query is submitted, the server translates the query in multiple steps.

3.2 Natural Language-to-SQL Translation

The main novelty of DBPal is that the query translation from natural language to SQL is modeled as a language translation problem using a state-of-the art Recurrent Neural Network (RNN) model [9]. A major challenge when using such a model is the need for a large and comprehensive training set mapping natural language to SQL. While existing work already has shown that a manually curated training corpus can be used to train a NL-to-SQL sequence-to-sequence model [4], such an approach imposes a high overhead for every new database schema that needs to be supported.

Our approach is therefore different since it can automatically "generate" a training set for any given database schema automatically as shown in Figure 3 (left hand side). The main idea is that in a first step called *Training Data Instantiation*, we generate a small training set by using a set of simple templates that describe SQL-NL pairs and instantiate them using a given database schema. In the second step called *Training Data Augmentation*, we automatically enrich the simple instantiated SQL-NL pairs by applying different techniques that leverage existing language models to paraphrase but also to introduce noise etc. The goal of the augmentation is to

cover a wide spectrum of ways users might ask questions against the given database.

Training Data Instantiation. The observation is that SQL – as opposed to natural language – has a much limited expressivity. We therefore use query templates to instantiate different possible SQL queries that a user might phrase against a given database schema such as:

Select {Att}(s) From {Table} Where {Filter}

The SQL templates cover a variety of different types of queries from simple select-from-where queries up to more complex aggregate-grouping queries and some simple nested queries.
For each SQL template, we define one or more natural language (NL) templates as counterparts for direct translation such as:

{SelectPhrase} the {Att}(s) {FromPhrase} {Table}(s) {WherePhrase} {Filter}

Reflecting the larger expressivity of spoken language versus SQL, our NL templates contain slots for speech variation (e.g., *SelectPhrase*, *FromPhrase*, *WherePhrase*) in addition to the slots for database items (Tables, Attributes, ...) present in the SQL templates. To instantiate the initial training set, the generator repeatedly instantiates each of our natural language templates by filling in their slots. Table, column and filter slots are filled using the schema information of the database, while a diverse array of natural language slots is filled using a manually-crafted dictionary of synonymous words and phrases. For example, the *SelectPhrase* can be instantiated using *What is* or *Show me*. Thus, an instantiated SQL-NL pair might look like this:

SELECT name FROM patient WHERE age=20 and
Show me the name of all patients with age 20

It is also of importance to balance the training data when instantiating the slots with possible values. If we naively replace the slots of a query template with all possible combinations of slot instances (e.g., all attribute combinations of the schema), then instances that result from templates with more slots would dominate the training set and add bias to the translation model. An imbalance of instances can result in a biased training set where the model would prefer certain translations over other ones only due to the fact that certain variations appear more often.
Finally, in the current prototype, for each initial NL template, we additionally provide some manually curated paraphrased NL templates that follow particular paraphrasing techniques as discussed in [11], covering categories such as syntactical, lexical or morphological paraphrasing. Although they are crafted manually, these templates are all database schema independent and can be applied to instantiate NL-SQL pairs for any possible schema without additional manual effort. Moreover, instantiated paraphrased SQL-NL pairs will still be fed into the automatic paraphrasing that is applied next during automatic augmentation.

Training Data Augmenation. In order to make the trained deep model more robust, we apply the following post-processing steps after the template instantiation phase:

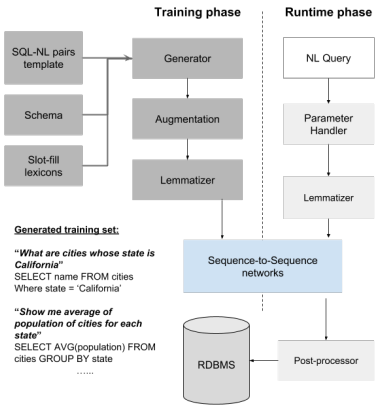

Figure 3: An overview of our Learned Query Translator

First, we augment the training set by duplicating NL-SQL pairs, but randomly selecting words of a sentence and paraphrase them using The Paraphrase Database (PPDB) [6] as the lexical resource. Second, another problem is missing or implicit information in queries. For example, a user might query the "patients with flu" instead of the "patients diagnosed with flue" and thus the information about the table column might be missing in a user query. To make the translation more robust against missing information, we copy individual NL-SQL pairs, then randomly select words and remove them from the natural language utterances. Third, for each resulting natural language query pair, we lemmatize the words. Finally, constants such as numbers or strings are replaced by special tokens.
In the future, we plan on extending our augmentation techniques; e.g., one avenue is to enhance our automatic paraphrasing using PPDB by paraphrasing multi-word phrases and clauses. We also plan to investigate the idea of using an off-the-shelf part-of-speech tagger to enrich each word in a given query and make the model more robust towards syntactic paraphrasing.

Training the Model. We follow a similar architecture of a sequence-to-sequence (seq2seq) model from [9] for machine translation tasks. Our model maps the natural language input directly to SQL output queries, which could be seen as the target language. The model itself consists of two recurrent neural networks, namely, the encoder and decoder. We use a bidirectional encoder-decoder architecture as proposed by [2]. During training, we also apply a dropout on embedding layer to avoid over-fitting to our training corpus, which only consists of a small vocabulary (i.e., SQL keywords as well as schema information of the given database).

Runtime Usage of Model. Our learned model is used at runtime to translate an input natural language query coming from a user to an executable SQL query as shown in Figure 3 (right-hand side). The runtime pipeline comprises also of multiple steps: First, we handle input parameters (e.g., "New York") and replace them by special tokens (e.g., "CITY_NAME"). The reason is that we want to be able to translate queries for unseen constants correctly. Second, a complex query handler tries to split a natural language query into multiple simple queries that are translatable by our model and combines them using SQL operators such as nesting or joining. Third, after translation, the special tokens are again replaced by their real values in the SQL query in the postprocessing phase. For

the first two tasks we rely on another set of deep models. Describing the details of these runtime-models, however, is beyond the scope of the demo paper.

3.3 Interactive Auto-Completion

To improve the search performance of users, we provide an auto completion mechanism on queries. This well-known paradigm not only enhances the search behavior, but also helps to increase translation accuracy by leading the user into less ambiguous queries. Considering the abstraction between user and the schema, auto-completion becomes even more crucial. The core of the auto completion system is comprised of a generic deep learning model and a modified search algorithm that gives priority to database specific elements. Given an input sequence, the model performs a breadth-first search in the candidate space, and returns the most probable completions of potential user queries.

4 DEMONSTRATION

The demonstration presents our first protoype of DBPal that allows users to select from two data sets: the ParaphraseBench[2] that consists of a a simple one-table dataset of hospital patients and tests the robustness against linguistic variations, and Geo880 (called 'Geo' in this paper), a benchmark dataset of geographical information of the United States that has been commonly tested upon by prior NLIDBs such as [4] and [8]. We will allow the user to query both the Patients and Geo datasets with increasingly difficult utterances as shown below. All queries can be formulated with and without the auto-completion feature being enabled.

4.1 Simple SQL-like Queries

We first provide the user with a list of straightforward functions that line up with the functionality of single-table SQL queries such as simple filter queries and aggregations (e.g. MAX, MIN, COUNT) and GROUP BY statements. We allow the user to play around with the canonical utterances that are syntactically similar to their SQL counterparts for single table in both the Patients and Geo contexts. Example queries include:

- *"show me the distinct names of patients where diagnosis is flu"*
- *"for each gender, what is the minimum length of stay of patients?"*
- *"what is the name and capital of states where name is not Mississippi and population is less than 5000000"*
- *"return the count of mountains where state name is Colorado"*

4.2 Paraphrased and Fragmented Queries

After working with simple queries, we then allow users to alter their earlier queries to make them less syntactically and semantically aligned with a corresponding SQL query. This can involve restructuring the word syntax of the query, paraphrasing words and phrases with ones that are semantically equivalent, or removing words from the query that are superfluous and/or that do not alter the semantics of the query. Example for this part of the demo are:

- *"distinct names of flu patients"*
- *"find the shortest patient length of stay from each gender"*
- *"enumerate Colorado's mountains"*

[2]https://datamanagementlab.github.io/ParaphraseBench/

4.3 Complex Queries

In the last part, we give the user freedom to query the system with whatever utterances they can come up with. These may include contextual questions or more nuanced requests; for example:

- *"Who is the oldest patient?"*
- *"Show the flu patient with the longest length of stay"*
- *"Select the highest point and area from all states"*

5 LIMITATION AND FUTURE WORK

The current prototype of DBPal already shows a significant improvement over other state-of-the-art-systems such as NaLIR [5] or ATHENA [8] when dealing with paraphrasing and other linguistic variations. Its main limitation, however, is the lack of coverage for more complicated queries such as different forms of nesting in SQL. However, this is not necessarily a limitation upon the translation model itself, but rather of the training set generation. We are currently extending the training data instantiation phase with additional templates and lexicons as well as the augmentation phase to add more complex natural language queries. Another future avenue of development is allowing users to incrementally build queries in a chatbot-like interface, where the system can ask for clarifications if the model can not translate a given input query directly. We expect that this feature will also be especially helpful for handling nested queries. Finally, integration with other deep models (e.g., for Question-Answering) seems to be another promising avenue for future exploration.

6 ACKNOWLEDGEMENTS

This research is funded in part by NSF Awards IIS-1526639 and IIS-1514491, as well as gifts from Google.

REFERENCES

[1] I. Androutsopoulos et al. Natural language interfaces to databases - an introduction. *Natural Language Engineering*, 1:29–81, 1995.
[2] D. Bahdanau et al. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.
[3] A. Crotty et al. Vizdom: Interactive analytics through pen and touch. *PVLDB*, 8(12):2024–2035, 2015.
[4] S. Iyer et al. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 963–973, 2017.
[5] F. Li et al. Constructing an interactive natural language interface for relational databases. *PVLDB*, 8:73–84, 2014.
[6] E. Pavlick et al. Simple PPDB: A paraphrase database for simplification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers*, 2016.
[7] A.-M. Popescu et al. Towards a theory of natural language interfaces to databases. In *IUI*, 2003.
[8] D. Saha and othersa. Athena: An ontology-driven system for natural language querying over relational data stores. *Proc. VLDB Endow.*, 9(12):1209–1220, Aug. 2016.
[9] I. Sutskever et al. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014. MIT Press.
[10] P. Terlecki et al. On improving user response times in tableau. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, SIGMOD '15, pages 1695–1706, 2015.
[11] M. Vila et al. Paraphrase concept and typology. A linguistically based and computationally oriented approach. *Procesamiento del Lenguaje Natural*, 46:83–90, 2011.
[12] Y. Wang et al. Building a semantic parser overnight. In *ACL*, 2015.