DBPal: Weak Supervision for Learning a Natural Language Interface to Databases

Nathaniel Weir, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bushan, Ugur Cetintemel
Brown University, USA

Prasetya Utama, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Carsten Binnig
TU Darmstadt, Germany

ABSTRACT

This paper describes DBPal, a new system to translate natural language utterances into SQL statements using a neural machine translation model. While other recent approaches use neural machine translation to implement a Natural Language Interfaces to Databases (NLIDB), existing approaches rely on supervised learning with manually curated training data, resulting in a high overhead for supporting each new database schema. In order to avoid this issue, DBPal implements a novel training pipeline based on weak supervision that synthesizes all training data from a given database schema. In our evaluation, we show that DBPal can outperform existing rule-based NLIDBs while achieving comparable performance to other NLIDBs that leverage deep neural network models but rely on manually curated training data for each database schema.

PVLDB Reference Format:
Nathaniel Weir, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bushan, Ugur Cetintemel, Prasetya Utama, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Carsten Binnig. DBPal: Weak Supervision for Learning a Natural Language Interface to Databases. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Motivation: Structured query language (SQL), despite its expressiveness, may hinder users with little or no relational database knowledge from exploring and making use of the data stored in a DBMS. In order to effectively analyze such data, users are required to have prior knowledge of the syntax and semantics of SQL. These requirements set "a high bar for entry" for democratized data exploration and have therefore triggered new efforts to develop alternative interfaces that allow non-technical users to explore and interact with their data more conveniently. While visual data exploration tools (e.g., Tableau, Vizdom [2]) have recently gained significant attention, Natural Language Interfaces to Databases (NLIDBs) arose as highly promising alternatives as they enable users to pose questions in a concise and intuitive manner. For example, imagine that a doctor at a hospital and wants to learn about the age distribution of patients with the longest stays in the hospital. This question typically requires the doctor — when using a standard database interface directly — to write a complex SQL query.

Contributions: Understanding natural language questions and translating them accurately to SQL is a complicated task, and thus NLIDBs have not yet made their way into commercial products. In this paper, we introduce DBPal, a natural language interface to SQL databases with improved robustness towards language variations. In order to provide a more robust NL interface, DBPal leverages a deep neural network model, which has become a standard for machine translation tasks. While there are other recent efforts that use deep models to implement NLIDB solutions [4, 9, 10], these commonly rely on supervised learning approaches that require substantial amounts of training data, particularly for sophisticated neural architectures such as deep sequence-to-sequence models.

The aforementioned approaches have largely ignored this problem and assumed the availability of manually-curated training sets (e.g., via crowdsourcing). As such, additional manual effort is needed for each new database schema, severely limiting the portability and ready applicability of these approaches to new domains. In order to address this fundamental limitation, we have built DBPal as a complete system that enables users to build robust NL-interfaces for different databases with low manual overhead.

At its core, DBPal implements a novel training pipeline for NLIDBs that synthesizes its training data using the principle of *distant (or weak) supervision* [1, 3]. The basic idea of distant supervision is to leverage various heuristics and existing datasets to automatically generate large (and potentially noisy) training data instead of handcrafting them.

In its basic form, only the database schema is required as input in order to generate a large collection of pairs of natural language queries and their corresponding SQL statements that can be used to train our language translation model. In order to maximize our coverage across natural language variations, we use additional input sources to automatically augment the training data using a collection of techniques. One such augmentation step, for example, is an automatic paraphrasing process using an off-the-shelf paraphrasing database [6].

Outline: The remainder of this paper is organized as follows. In Section 2, we first introduce the overall system

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

architecture of DBPal. Afterwards, in Section 3, we describe the details of the novel training pipeline of DBPal based on distant supervision. In order to show the efficiency of DBPal as well as its robustness, we present initial results of our evaluation using benchmarks in Section 4. Finally, we describe promising future directions in Section 5.

2. OVERVIEW

Figure 1 shows an overview of the architecture of DBPal. At the core of DBPal is a neural machine translation model (i.e., a sequence-to-sequence model).

The most important aspect of DBPal is the novel training pipeline based on weak supervision that automatically generates training data that is used to train the translation model. The basic flow of the training pipeline is shown on the left-hand-side of Figure 1. In the following, we describe the training pipeline and focus in particular on the data generation framework. The details of the full training pipeline will be explained in Section 3. In the first step, the *Generator* uses the database schema along with a set of seed templates that describe typical NL-SQL pairs to generate an initial training set of NL-SQL pairs. In the second step, *Augmentation*, the training data generation pipeline then automatically adds to the initial training set of NL-SQL pairs by leveraging existing general-purpose data sources and models to linguistically modify the NL part of each pair.

Furthermore, the runtime phase is comprised of multiple components such as a *Query Preprocessor* and a *Query Postprocessor* as shown on the right-hand-side of Figure 1. The pre-processor, for example, is responsible for replacing the constants in the input NL query with placeholders to make the translation model independent from the actual database content and avoid needing to retrain the model if the database is updated. For example, for the input query shown in Figure 1 `"What are the cities whose state is California"`, the pre-processor replaces "California" by the appropriate schema element using a placeholder `"@STATE"`. The neural model then works on these (so called) anonymized NL input queries and creates output SQL queries, which also contain placeholders. In the example in Figure 1, the output of the neural translator is `"SELECT * FROM city WHERE city _state_name=@California")`. The task of the post-processor is then to replace the placeholders again by the actual constants such that the SQL query can be executed over the database.

3. TRAINING PIPELINE

In the following, we describe the training data generation procedure implemented in DBPal, which is at the core of our weakly supervised training pipeline. In this pipeline, an initial instantiation step first generates a simple set of Natural-Language-to-SQL (NL-SQL) pairs for a given schema. Afterwards, an augmentation step that is based on existing language models (e.g., for automatic paraphrasing) generates further linguistic variants for each NL-SQL pair to cover a wider range of linguistic variances of possible natural language questions.

3.1 Data Instantiation

The main observation of the instantiation step is that SQL, as opposed to natural language, has a much more 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:

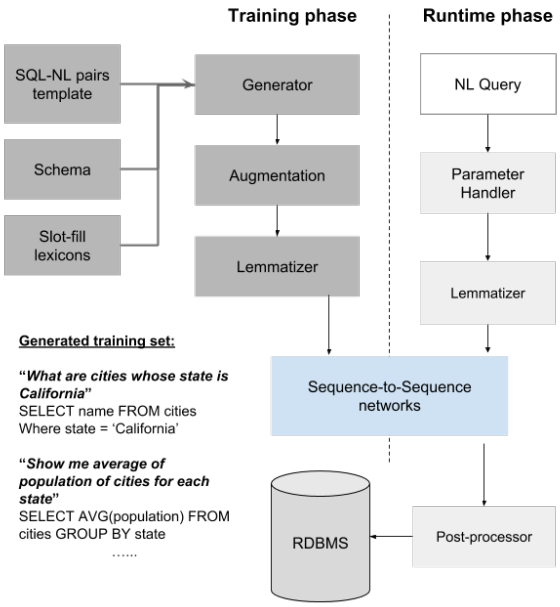

Figure 1: DBPal's Training and Runtime Phase

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

The SQL templates cover a variety of different types of queries from simple `SELECT-FROM-WHERE` queries to more complex group-by aggregation queries as well as 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*}

To take into account the expressivity of spoken language compared to 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. Then, 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 information from the database's schema, while a diverse array of natural language slots are filled using manually-crafted dictionaries of synonymous words and phrases. For example, the *SelectPhrase* can be instantiated using *What is* or *Show me*. Thus, an instantiated NL-SQL pair might look like:

NL: *Show me the name of all patients with age 20*
SQL: *SELECT name FROM patient WHERE age=20*

An important part of training data instantiation is balancing the number of NL-SQL pairs that are instantiated per template. 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 bias the translation model. An imbalance of instances can result in a biased training set where the model would prefer certain translations over others 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 [8], covering categories such as syntactical, lexical, and morphological paraphrasing. Importantly, the paraphrased templates are database independent and can be applied to instantiate the training data for any given schema. Moreover, instantiated paraphrased NL-SQL pairs will still be fed into the automatic paraphrasing that is applied next during automatic data augmentation.

3.2 Data Augmentation

In order to make the query translation model more robust to linguistic variation (i.e., how a user might phrase an input NL query), we apply the following augmentation steps for each instantiated NL-SQL pair:

First, we augment the training set by duplicating NL-SQL pairs, by randomly selecting words and sub-clauses of the NL query and paraphrasing them using the Paraphrase Database (PPDB) [6] as a lexical resource. An example for this is:

Input NL Query:
Show the name of all patients with age @AGE

PPDB Output:
demonstrate, showcase, display, indicate, lay

Paraphrased NL Query:
display the names of all patients with age @AGE

PPDB is an automatically extracted database containing millions of paraphrases in 27 different languages. In its English corpus currently used by DBPal, PPDB provides over 220 million paraphrase pairs, consisting of 73 million phrasal and 8 million lexical paraphrases, as well as 140 million paraphrase patterns, which capture many meaning-preserving syntactic transformations. The paraphrases are extracted from bilingual parallel corpora totalling over 100 million sentence pairs and over 2 billion English words.

In the automatic paraphrasing step, we use PPDB as an index that maps words/sub-phrases to paraphrases and replace words/sub-phrases of the input NL query with available paraphrases. For any given input phrase to PPDB, there are often tens or hundreds of possible paraphrases available. For example, searching in PPDB for a paraphrase of the word *enumerate* which can be used as a part of the question *"enumerate the names of patients with age 80"*, we get suggestions such as *list* or *identify*.

Second, another challenge of input NL queries is missing or implicit information. For example, a user might ask for *patients with flu* instead of *patients diagnosed with flu* and thus the information about the referenced attribute might be missing in a user query.

Therefore, to make the translation more robust against missing information, we duplicate individual NL-SQL pairs and select individual words/sub-clauses that are removed from the NL query. Similar to paraphrasing, an interesting question is which words/sub-clauses should be removed and how aggressively the removal should be applied. Currently, we follow a similar approach as the paraphrasing process by randomly selecting words in the NL query and removing them in a duplicate. Again, aggressively removing words from query copies increases the training data size since more variations are generated. On the other hand, however, we again might introduce noisy training data which leads to a drop in translation accuracy.

In order to tune how aggressively we apply removing as well as the other augmentation techniques, we provide parameters for the data generation process. Tuning such parameters is similar to hyper-parameter tuning in machine learning and we therefore plan to explore routes in the future that look at how to automatically tune the data generation parameters.

4. INITIAL RESULTS

In the following, we present our initial experimental results with DBPal. In all our experiments, we compare to two baselines: First, we compare against an approach called neural semantic parser (NSP) [4], which also leverages a deep model for the translation process. However, unlike DBPal, NSP requires a manual curated training set for each new database schema. As a second baseline, we used NaLIR [5], a state-of-the-art rule-based NLIDB that requires no training data to support a new schema.

To evaluate these approaches, we use the GeoQuery benchmark that has already been used to evaluate other NLIDBs [7]. However, this benchmark does not explicitly test different linguistic variations, which is important to understand for the robustness of an NLIDB. For testing different linguistic variants, we curated a new benchmark as part of this paper that covers different linguistic variations for the user NL input and maps it to an expected SQL output. The benchmark is available online [1].

4.1 Exp. 1: Overall Results

We evaluated the performance of all NLIDB systems in terms of their accuracy, defined as the number of natural language queries translated correctly over the total number of queries in the test set. Correctness is determined by whether the yielded records from a query's execution in the RDBMS contain the information that is requested by the query intent. The correctness criteria is relaxed by also considering execution results that consist of supersets of the requested columns to be correct. We argue that in practice, users are still able to retrieve the desired information by examining all columns of returned rows. Table 1 summarizes the accuracy measures of all NLIDB systems on the two benchmark datasets using this correctness criterion.

First, we can see that DBPal outperforms NaLIR which requires no manual effort to support a new database. NaLIR relies heavily on user feedback in case NaLIR does not automatically find a valid SQL translation. For comparison, we thus ran the NaLIR implementation in the non-interactive as well as the interactive setting, where we provide perfect feedback from users (i.e. users always make the correct choices if NaLIR asks for feedback). As seen in our experiments, with interactive feedback, we see a slight increase in accuracy; however, NaLIR often fails in the initial translation

[1] `https://datamanagementlab.github.io/ParaphraseBench/`

	Patients	GeoQuery
NaLIR (w/o feedback)	15.60%	7.14%
NaLIR (w feedback)	21.42%	N/A
NSP	N/A	**83.9**%
DBPal (w/o augmentation)	74.80%	38.60%
DBPal (full pipeline)	**75.93**%	55.40%

Table 1: Accuracy comparison between DBPal and other baselines on the two benchmark datasets

	Naive	Syntactic	Lexical	Morphological	Semantic	Missing	Mixed
NaLIR (w/o feedback)	19.29%	28.07%	14.03%	17.54%	7.01%	5.77%	17.54%
NaLIR (w feedback)	21.05%	38.59%	14.03%	19.29%	7.01%	5.77%	22.80%
DBPal (full pipeline)	**96.49**%	**94.7**%	**75.43**%	**85.96**%	**57.89**%	**36.84**%	**84.20**%

Table 2: Accuracy break-down between DBPal and other baselines for the Patient benchmark

step before giving the user the chance to provide feedback. Furthermore, for GeoQuery, we did not run NaLIR in interactive mode due to the high manual evaluation effort of running more than 300 queries and providing feedback for those queries.

Second, we observe that NSP, which requires the costly curation of a training set of NL-SQL pairs, unsurprisingly achieves the highest accuracy on its given domain of GeoQuery. We could not evaluate NSP on the Patients benchmark since no manual curated training data was available. It is interesting to note that NSP achieves a high quality on GeoQuery while DBPal only achieves approximately 50%. However, after analyzing the manual training data of NSP we note that it contains many of the structurally similar NL-SQL pairs in their training set that are also in the test set of GeoQuery. Furthermore, upon anonymization of database values, it becomes clear that identical queries appear in both the training and testing sets. Thus, the learned model is heavily overfit to the training patterns.

Another observation is that DBPal has a lower accuracy on GeoQuery than on Patients. This is due to the fact that GeoQuery contains a huge fraction of rather complex join and nested queries that are currently not supported well in DBPal. When removing those queries from GeoQuery, DBPal actually achieves a comparable accuracy as NSP.

Finally, we analyzed the benefit of the data augmentation step of our training data generation which produces more NL-SQL-pairs with a large about of variety. For the patients benchmark, we can see that this contributes only minor improvements to the performance, while for the GeoQuery dataset, the amount of correctly translated queries is ten percentage points higher. This is due to the higher complexity of this database schema, which consists of multiple tables with relationships that bring about richer semantics when asking questions.

4.2 Exp. 2: Performance Breakdown

In the second experiment, we show the breakdown of natural language variants using the Patient benchmark in Table 2. We can see that NaLIR achieves a similar accuracy of $11/57 = 19.29\%$ without user feedback and a slightly higher accuracy of $12/57 = 21.05\%$ with user feedback for the naive testing set. Furthermore, in most testing sets, perfect user feedback does not significantly improve the accuracy of NaLIR since NaLIR relies on off-the-shelf dependency parser library that performs reasonably well only on well-structured sentences. Therefore, most of NaLIR's failure cases are due to dependency parsing errors caused by ill-formed, incomplete, or keywords-like queries. Moreover, NaLIR often fails to find correct candidate mappings of query tokens to RDBMS elements due to paraphrased inputs. Both of these problems cannot be repaired by user feedback, given the translation procedure fails already before feedback can be provided.

5. FUTURE WORK

The next step for DBPal that we plan to explore is to further extend the training data instantiation and augmentation processes with additional templates and lexicons to cover more linguistic variations, as well as to add support for even more complex SQL queries (e.g., joins, nested queries). Second, a major limitation of DBPal is the lack of coverage to explain results to the user and suggest ways to correct the queries if the translation was incorrect. A future avenue of development is therefore to allow users to incrementally build and refine queries in a chatbot-like interface, where the system can ask for clarifications if the model cannot translate a given input query directly.

6. REFERENCES

[1] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. M. Mitchell, K. Nigam, and S. Slattery. Learning to construct knowledge bases from the world wide web. *Artif. Intell.*, 118(1-2):69–113, 2000.

[2] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. Vizdom: Interactive analytics through pen and touch. *PVLDB*, 8(12):2024–2027, 2015.

[3] M. Dehghani, H. Zamani, A. Severyn, J. Kamps, and W. B. Croft. Neural ranking models with weak supervision. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 65–74, 2017.

[4] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer. 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. Nalir: an interactive natural language interface for querying relational databases. In *SIGMOD*, pages 709–712, 2014.

[6] E. Pavlick and C. Callison-Burch. 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, O. Etzioni, and H. A. Kautz. Towards a theory of natural language interfaces to databases. In *IUI*, 2003.

[8] M. Vila, M. A. Martí, and H. Rodríguez. Paraphrase concept and typology. A linguistically based and computationally oriented approach. *Procesamiento del Lenguaje Natural*, 46:83–90, 2011.

[9] Y. Wang, J. Berant, and P. Liang. Building a semantic parser overnight. In *ACL*, 2015.

[10] X. Xu, C. Liu, and D. Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *CoRR*, abs/1711.04436, 2017.