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

Nathaniel Weir, Andrew Crotty, Alex Galakatos, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, 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 Interface to Databases (NLIDB), existing techniques rely on supervised learning with manually curated training data, which results in substantial 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 without relying on manually curated training data for every new database schema.

PVLDB Reference Format:
Nathaniel Weir, Andrew Crotty, Alex Galakatos, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Ugur Cetintemel, Prasetya Utama, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Carsten Binnig. DBPal: Weak Supervision for Learning a Natural Language Interface to Databases. *1st Int'l Workshop on Conversational Access to Data (CAST)*, 2019.

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. Despite the recent popularity of visual data exploration tools (e.g., Tableau [1], Vizdom [3]), Natural Language Interfaces to Databases (NLIDBs) have emerged as a highly promising alternative, since they enable users to pose questions in a concise and intuitive manner.

1st Int'l Workshop on Conversational Access to Data (CAST)
in conj. with the 45th Int'l Conference on Very Large Data Bases (VLDB)
Los Angeles, California - August 26-30, 2019.

Contributions: Understanding natural language (NL) questions and translating them accurately into SQL is a complicated task, and thus NLIDBs have not yet made their way into mainstream commercial products. This paper describes DBPal, a natural language interface to databases with improved robustness to linguistic 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 many other recent efforts also use deep models to implement NLIDBs [8, 17, 18], they 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, which severely limits the portability 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 *weak supervision* [2, 5]. The basic idea of weak supervision is to leverage various heuristics and existing datasets to automatically generate large (and potentially noisy) training datasets 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 then 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 [13].

Outline: The remainder of this paper is organized as follows. In Section 2, we first introduce the overall system architecture of DBPal. Afterwards, in Section 3, we describe the details of the novel training pipeline of DBPal based on weak supervision. In order to show the accuracy of DBPal as well as its robustness, we present initial results of our evaluation in Section 4. Finally, we describe promising future directions in Section 5.

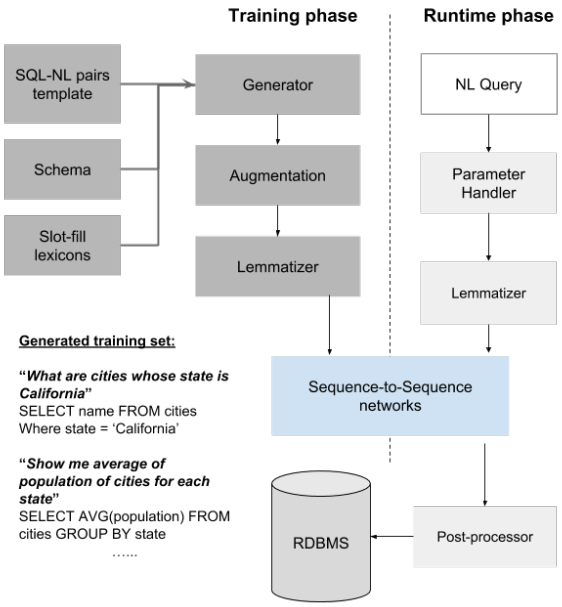

Figure 1: DBPal**'s Training and Runtime Phases**

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 used for training 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, as shown on the right-hand side of Figure 1. The *Parameter Handler* 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 (i.e., *"What are cities whose state is California?"*), the *Parameter Handler* replaces *"California"* with the appropriate schema element using the placeholder `@STATE`. The neural model then works on these 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 name FROM cities WHERE state = @STATE`. The task of the *Post-processor* is then to replace the placeholders again with the actual constants such that the SQL query can be executed against 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 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 variations for possible natural language questions.

3.1 Data Instantiation

The main observation of the instantiation step is that SQL, as opposed to natural language, has significantly less 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 {*Attribute*}*(s)* *From* {*Table*} *Where* {*Filter*}

The SQL templates cover a variety of query types, 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 NL templates as counterparts for direct translation, such as:

{*SelectPhrase*} {*Attribute*}*(s)* {*FromPhrase*} {*Table*}*(s)* {*WherePhrase*} {*Filter*}

To take into account the expressivity of NL compared to SQL, our templates contain slots for speech variation (e.g., *SelectPhrase*, *FromPhrase*, *WherePhrase*) in addition to slots for database objects (e.g., tables, attributes). 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 phrases *"what is"* or *"show me"* can be used to instantiate the *SelectPhrase*. A fully instantiated NL-SQL pair might look like:

NL: *Show the names of all patients with age 20.*
SQL: `SELECT name FROM patients 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 [16], covering categories such as syntactical, lexical, and morphological paraphrasing. Importantly, the paraphrased templates can be applied to instantiate the training data for any given schema, and the instantiated NL-SQL pairs are also automatically paraphrased 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 generating duplicate NL-SQL pairs. This process involves randomly selecting words/subphrases of the NL query and paraphrasing them using the Paraphrase Database (PPDB) [13] as a lexical resource, for example:

Input NL Query:
Show the names 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. DBPal uses PPDB's English corpus, which 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 a wide range of meaning-preserving syntactic transformations. The paraphrases are extracted from bilingual parallel corpora totaling over 100 million sentence pairs and over 2 billion English words.

During paraphrasing, we randomly replace words/subphrases of the input NL query with available paraphrases provided by PPDB. For example, searching in PPDB for a paraphrase of the word *enumerate*, as in *"enumerate the names of patients with age 80"*, we get suggestions such as *"list"* or *"identify"* as alternatives.

Second, to make the translation more robust to missing or implicit context, we randomly drop words/subphrases from the NL training queries. For example, a user might ask for *"patients with flu"* instead of *"patients diagnosed with flu"*, where the referenced attribute is never explicitly stated. Similar to paraphrasing, an interesting question is: which words/subphrases should be removed and how frequently to remove them? 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 that leads to a drop in translation accuracy.

In order to tune how aggressively we apply removal 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 investigate how to automatically tune the data generation parameters.

4. INITIAL RESULTS

In the following, we present our initial experimental results with DBPal. In all experiments, we compare against a neural semantic parser (NSP) [8] approach, which also leverages a deep model for the translation process. However, unlike DBPal, NSP requires a manually curated training set for each new database schema. As a second baseline, we also used NaLIR [11, 10], a state-of-the-art rule-based NLIDB that requires no training data to support a new schema.

The first part of our evaluation uses the well-known GeoQuery benchmark, but this benchmark does not explicitly test different linguistic variations to measure robustness. For testing different linguistic variants, we curated a new benchmark, called the Patients [1] benchmark, that covers different linguistic variations for the user NL input and maps it to an expected SQL output.

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 DBMS 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 note that DBPal outperforms NaLIR, which relies heavily on user feedback when it cannot automatically find a valid SQL translation. We ran NaLIR in the baseline non-interactive mode as well as interactive mode, where we provide perfect feedback (i.e., users always make the correct choices if NaLIR asks for feedback). With interactive feedback, we observe a slight increase in accuracy on the Patients benchmark, but we did not run NaLIR in interactive mode on GeoQuery due to the high manual effort of providing feedback for more than 300 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 manually curated training data was available. It is interesting to note that NSP performs well on GeoQuery while DBPal only achieves approximately 50% accuracy. 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 joins and nested queries that are currently not supported well in DBPal. When removing those queries from GeoQuery, DBPal actually performs comparably to 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 amount of variety. For the Patients benchmark, we can see that this contributes only minor improvements to the performance, while for the GeoQuery dataset, the accuracy is almost 20% 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.

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

	Patients	GeoQuery
NaLIR (no feedback)	15.60%	7.14%
NaLIR (feedback)	21.42%	N/A
NSP	N/A	**83.9**%
DBPal **(no augmentation)**	74.80%	38.60%
DBPal **(full pipeline)**	**75.93**%	55.40%

Table 1: Accuracy comparison between DBPal and other baselines on both benchmarks.

	Naive	Syntactic	Lexical	Morphological	Semantic	Missing	Mixed
NaLIR (no feedback)	19.29%	28.07%	14.03%	17.54%	7.01%	5.77%	17.54%
NaLIR (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 breakdown between DBPal and other baselines for the Patients benchmark.

4.2 Exp. 2: Performance Breakdown

For the Patients benchmark, we show the breakdown of different NL variants in Table 2. We 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 an 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 keyword-based queries. Moreover, NaLIR often fails to find correct candidate mappings of query tokens to schema elements due to paraphrased inputs. Both of these problems cannot be repaired by user feedback, given the translation procedure fails before any feedback can be provided.

5. FUTURE WORK

In the immediate future, we plan to explore extensions to the training data instantiation and augmentation processes by creating additional templates and lexicons to cover more linguistic variations, as well as to better support even more complex SQL queries (e.g., joins, nested queries). Additionally, DBPal currently has no good mechanisms for explaining results to the user, suggesting possible corrections if the translation was incorrect, or incorporating user feedback to make corrections. We therefore plan to explore ways to enable users to incrementally build and refine queries in a conversational chatbot-like interface, where the system can ask for clarifications if the model cannot translate a given input query directly. This feature will also help to improve the overall user experience by leveraging contextual clues from a series of interactions without requiring the user to explicitly reiterate elements from past queries.

Longer term, we believe that an exciting opportunity exists to expand DBPal into a full-fledged data science platform that will allow domain experts to interactively explore large datasets using only natural language [9]. Again, in contrast to the typical notion of one-shot SQL queries currently taken by DBPal, data science is an iterative, session-driven process where a user repeatedly modifies a query or machine learning model after examining intermediate results until finally arriving at some desired insight, which will therefore necessitate a more conversational interface. Such a system would require the development of new techniques for providing progressive results [19, 15] by extending past work on traditional SQL-style queries [4, 7] and machine learning models [14].

Finally, we believe there are also interesting opportunities related to different data models (e.g., time series [6]) and new user interfaces (e.g., query-by-voice [12]).

6. REFERENCES

[1] Tableau. https://www.tableau.com/.

[2] 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.*, pages 69–113, 2000.

[3] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. Vizdom: Interactive Analytics through Pen and Touch. *PVLDB*, 8(12):2024–2027, 2015.

[4] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. The Case for Interactive Data Exploration Accelerators (IDEAs). In *HILDA@SIGMOD*, 2016.

[5] M. Dehghani, H. Zamani, A. Severyn, J. Kamps, and W. B. Croft. Neural Ranking Models with Weak Supervision. In *SIGIR*, pages 65–74, 2017.

[6] P. Eichmann, A. Crotty, A. Galakatos, and E. Zgraggen. Discrete Time Specifications In Temporal Queries. In *CHI LBW*, pages 2536–2542, 2017.

[7] A. Galakatos, A. Crotty, E. Zgraggen, C. Binnig, and T. Kraska. Revisiting Reuse for Approximate Query Processing. *PVLDB*, 10(10):1142–1153, 2017.

[8] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer. Learning a Neural Semantic Parser from User Feedback. In *ACL*, pages 963–973, 2017.

[9] R. J. L. John, N. Potti, and J. M. Patel. Ava: From Data to Insights Through Conversations. In *CIDR*, 2017.

[10] F. Li and H. V. Jagadish. Constructing an Interactive Natural Language Interface for Relational Databases. *PVLDB*, 8(1):73–84, 2014.

[11] F. Li and H. V. Jagadish. NaLIR: An Interactive Natural Language Interface for Querying Relational Databases. In *SIGMOD*, pages 709–712, 2014.

[12] G. Lyons, V. Tran, C. Binnig, U. Çetintemel, and T. Kraska. Making the Case for Query-by-Voice with EchoQuery. In *SIGMOD*, pages 2129–2132, 2016.

[13] E. Pavlick and C. Callison-Burch. Simple PPDB: A Paraphrase Database for Simplification. In *ACL*, pages 143–148, 2016.

[14] Z. Shang, E. Zgraggen, B. Buratti, F. Kossmann, P. Eichmann, Y. Chung, C. Binnig, E. Upfal, and T. Kraska. Democratizing Data Science through Interactive Curation of ML Pipelines. In *SIGMOD*, pages 1171–1188, 2019.

[15] C. Turkay, N. Pezzotti, C. Binnig, H. Strobelt, B. Hammer, D. A. Keim, J. Fekete, T. Palpanas, Y. Wang, and F. Rusu. Progressive Data Science: Potential and Challenges. *CoRR*, abs/1812.08032, 2018.

[16] 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.

[17] Y. Wang, J. Berant, and P. Liang. Building a Semantic Parser Overnight. In *ACL*, pages 1332–1342, 2015.

[18] X. Xu, C. Liu, and D. Song. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. *CoRR*, abs/1711.04436, 2017.

[19] E. Zgraggen, A. Galakatos, A. Crotty, J. Fekete, and T. Kraska. How Progressive Visualizations Affect Exploratory Analysis. *TVCG*, 23(8):1977–1987, 2017.