PHOTON: A Robust Cross-Domain Text-to-SQL System

Jichuan Zeng\textsuperscript{1}*, Xi Victoria Lin\textsuperscript{2}\textsuperscript{*}, Caiming Xiong\textsuperscript{2}, Richard Socher\textsuperscript{2}, Michael R. Lyu\textsuperscript{1}, Irwin King\textsuperscript{1}, Steven C.H. Hoi\textsuperscript{2}

\textsuperscript{1} The Chinese University of Hong Kong
\textsuperscript{2} Salesforce Research

\{jczeng,lyu,king\}@cse.cuhk.edu.hk
\{xilin,cxiong,rsocher,shoi\}@salesforce.com

Abstract

Natural language interfaces to databases (NLIDB) democratize end user access to relational data. Due to fundamental differences between natural language communication and programming, it is common for end users to issue questions that are ambiguous to the system or fall outside the semantic scope of its underlying query language. We present PHOTON, a robust, modular, cross-domain NLIDB that can flag natural language input to which a SQL mapping cannot be immediately determined. PHOTON consists of a strong neural semantic parser (63.2\% structure accuracy on the Spider dev benchmark), a human-in-the-loop question corrector, a SQL executor and a response generator. The question corrector is a discriminative neural sequence editor which detects confusion span(s) in the input question and suggests rephrasing until a translatable input is given by the user or a maximum number of iterations are conducted. Experiments on simulated data show that the proposed method effectively improves the robustness of text-to-SQL system against untranslatable user input. The live demo of our system is available at http://naturalsql.com/.

1 Introduction

Natural language interfaces to databases (Popescu et al., 2003; Li and Jagadish, 2014) democratize end user access to relational data and have attracted significant research attention for decades (Hemphill et al., 1990; Dahl et al., 1994; Zelle and Mooney, 1996; Popescu et al., 2003; Bertomeu et al., 2006; Zhong et al., 2017; Yu et al., 2018, 2019a). Most existing NLIDBs adopt a modular architecture consisting of rule-based natural language parsing, ambiguity detection and pragmatics modeling (Li and Jagadish, 2014; Setlur et al., 2016, 2019). While they have been shown effective in pilot study and production, rule-based approaches are limited in terms of coverage, scalability and naturalness – they are not robust against the diversity of human language expressions and are difficult to scale across domains.

Recent advances in neural natural language processing (Sutskever et al., 2014; Dong and Lapata, 2016; See et al., 2017a; Liang et al., 2017; Lin et al., 2019; Bogin et al., 2019a), pre-training (Devlin et al., 2019; Hwang et al., 2019), and the availability of large-scale supervised datasets (Zhong et al., 2019) have enabled significant improvements in the robustness of neural NLIDBs. However, compared to rule-based systems, they are not robust against the diversity of human language expressions and are difficult to scale across domains.
et al., 2017; Finegan-Dollak et al., 2018; Yu et al., 2018, 2019b,a) enabled deep learning based approaches to significantly improve the state-of-the-art in nearly all subtasks of building an NLIDB. These include semantic parsing (Dong and Lapata, 2018; Zhang et al., 2019), ambiguity detection and confidence estimation (Dong et al., 2018; Yao et al., 2019), natural language response generation (Liu et al., 2019) and so on. Moreover, by jointly modeling the natural language question and database schema in the neural space, latest text-to-SQL semantic parsers can work cross domains (Yu et al., 2018; Zhang et al., 2019).

We present PHOTON, a modular, cross-domain NLIDB that adopts deep learning in its core components. PHOTON consists of (1) a neural semantic parser, (2) a human-in-the-loop question corrector, (3) a SQL query executor and (4) a natural language response generator. The neural semantic parser assumes limited DB content access due to data privacy concerns (§ 3.1). It employs a BERT-based (Devlin et al., 2019) DB schema-aware question encoder and a pointer-generator decoder (See et al., 2017a) with static SQL correctness check. It achieves competitive performance on the popular cross-domain text-to-SQL benchmark, Spider (Yu et al., 2018) (63.2% structure accuracy on the dev set based on the official evaluation).1 The question corrector is a neural sequence editor which detects potential confusion span(s) in the input question and suggests possible corrections for the user to give feedback. When an input question is successfully translated into an executable SQL query, the response generator generates a natural language response conditioned on the output of the SQL query executor.

A pilot study with non-expert SQL users shows that the system effectively increases the flexibility of user’s natural language expression and is easy to be adapted to unseen databases. Being able to detect and correct untranslatable questions reduces unexpected error cases during user interaction.

2 System Design

In this section, we will elaborate on the system design of PHOTON. 

---

1We are continuously improving the performance of the neural semantic parser. Currently the semantic parser only accepts standalone question as input. We plan to also model the interaction context in future work.

2.1 Overview

Figure 1 shows the overall workflow of our system. PHOTON is an end-to-end system that takes a user question and database schema as input, and output the query result after executing the generated SQL on the database. PHOTON is a modular framework designed towards practical industrial applications. The core modules in PHOTON are the SQL parser and confusion detection mechanism. The SQL parser parses the input question and database schema, maps them into executable SQL query via an encoder-decoder framework. The confusion detection module identifies the untranslatable questions and captures the confusing span of the untranslatable question. The confusing tokens together with the context are fed into the auto-correction module to make a prediction of user attempted question.

To make it more applicable and accessible for user to query the database in a natural way, PHOTON also provides user interaction module enabling user to refine their queries in the interaction with the system. Response generation handles the output of the system by transducing the database-style query result into natural language or post warning when the query is non-executable on the database, making the system more user-friendly. Notice that the response generation module in the current version is implemented using a template-based approach and can be improved by using more advanced response generation models.

![Figure 2: State transition map of interaction in PHOTON. States with darker background are the end states that can receive user reply, and switch to INIT state automatically. The bottom part is the system response templates in each end state.](image-url)
2.2 User Interaction

Figure 2 illustrates the interaction process, which involves four types of response states: CONFIRM_RESULT, CONFIRM_CORRECTION, NEED_REPHRASE, and INVALID_QUERY. The set of response templates can be found at the bottom of Figure 2. When a user initiates the conversation by entering one query, PHOTON will first predict whether the query is translatable or not. If translatable, PHOTON generates the corresponding SQL command and checks the command’s executability; otherwise, PHOTON will provide a correction strategy (i.e., CONFIRM_CORRECTION) based on the detected confusing span or ask the users to further rephrase the inquiry (i.e., NEED_REPHRASE) if no span is captured.

2.3 UI Design

Our system UI consists of three panels: chat window, schema viewer and results viewer (Figure 3).

- **Chat window**: This is a standard chat window that facilitates communication between the user and PHOTON. The user types the natural language input and the natural language responses of the system are displayed.

- **Schema viewer**: This view provides a graph visualization of the underlying relational DB schema. The panel is hideable and will not be shown in case the DB schema is confidential.

- **Result viewer**: This view displays the returned results of an executable SQL query mapped from a confirmed input question. The SQL query is formatted and displayed in the top for user verification. Multi-record results are presented as sub-tables. Result consists of a single table cell is presented as a 1-cell sub-table. If the result comes from an aggregation operation such as a counting, the data records supporting the calculation are also shown for explainability. Confidential DB records are hidden from the display and the user is informed of the number of hidden records.

2.4 Cross Domain

A relational DB for user queries should be set before usage. PHOTON consists of a collection of default databases and allows users to upload their own DBs for testing. Users can select which database they want to query by clicking the “Selected Database” drop down button.

2.5 Dual Query Mode

PHOTON accepts both natural language questions and well-formed SQL queries as input. It automatically detects the user input type and executes the

<table>
<thead>
<tr>
<th>stadium ID</th>
<th>Name</th>
<th>Capacity</th>
<th>Highest</th>
<th>Lowest</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rath Rovers</td>
<td>10104</td>
<td>4815</td>
<td>1924</td>
<td>2156</td>
</tr>
<tr>
<td>2</td>
<td>Ayr United</td>
<td>11998</td>
<td>2363</td>
<td>1017</td>
<td>1477</td>
</tr>
<tr>
<td>3</td>
<td>East Fife</td>
<td>2000</td>
<td>1980</td>
<td>533</td>
<td>864</td>
</tr>
<tr>
<td>4</td>
<td>Queen’s Park</td>
<td>53500</td>
<td>1763</td>
<td>466</td>
<td>730</td>
</tr>
<tr>
<td>5</td>
<td>Sitting Ablon</td>
<td>9008</td>
<td>1125</td>
<td>404</td>
<td>642</td>
</tr>
</tbody>
</table>
input immediately if it is a valid SQL query. We observed that the dual query model can be more efficient than NL-only mode, especially for users who have SQL background.

3 Model

3.1 Neural Semantic Parser

The neural semantic parser is an end-to-end model whose input consists of a user question and the DB schema, and outputs a SQL query. Due to data privacy concerns, we assume that the neural semantic parser does not have full access to the DB content. Instead, we assume for each DB field, the parser have access to the set of possible values of the field, for example, “Country.Region”: {“Carribean”, “Porto Rico”, ...} . We call such value sets “picklists” by industry convention.

3.1.1 Schema-Question Encoder

Following previous work (Hwang et al., 2019; Zhang et al., 2019), we serialize the relational DB schema and concatenate it to the user question. As shown in Figure 4, we represent each table with the table name followed by a sequence of field names. Each table name is preceded by the special token [T] and each field name is preceded by the special token [C]. The representations of multiple tables are concatenated together to form the serialization of the schema, which is surrounded by [SEP] tokens and concatenated to the question. Finally, the question is preceded by the [CLS] token following convention of BERT encoder (Devlin et al., 2019).

This sequence is fed into the pretrained BERT, followed by a bi-directional LSTM to form a joint encoding of the question and schema h_input. The text portion of h_input is passed through another bi-LSTM to obtain the question encoding h_q. We represent each schema component (tables and fields) using the slices of h_input corresponding to the special token [T] and [C].

Meta-data Features We further trained dense look-up features to represent if a field is a primary key (f_pri), if a field appears in a foreign key pair (f_for) and the data type of the field (f_type). These meta-data features are fused with the representations in h_input via a projection layer g to obtain the final representation of each schema component:

\[ h^C_r = g([h^n_{input}; f^i_{pri}; f^j_{for}; f^k_{type}]) \]

\[ = \text{ReLU}(W_g [h^n_{input}; f^i_{pri}; f^j_{for}; f^k_{type}] + b_g) \]

\[ h^{T}_r = g([h^n_{input}; 0; 0; 0]), \]

where m is the index of the special token corresponding to the p-th column in the input and n is the index of the special token corresponding to the q-th table in the input. i, j and k are the feature indices indicating the corresponding properties of C_p: [h^n_{input}; f^i_{pri}; f^j_{for}; f^k_{type}] is the concatenation of the four vectors. The meta-data features we include are specific to fields and the table representations are fused with zero place-holder vectors.

3.1.2 Decoder

We use an LSTM-based sequential pointer-generator (See et al., 2017b) as the decoder. The generation vocabulary of our decoder consists of 70 SQL keywords and reserved tokens, plus the 10 digits. At each step, the decoder computes a probability distribution over actions that consists of generating a token from the reserved vocabulary, copying a token from the input text or copying a schema component.

3.1.3 Static SQL Correctness Check

The sequential pointer-generator we adopted does not guarantee the output SQL is syntactically correct. In practice, we perform beam-search decoding and run a static SQL correctness check to eliminate erroneous predictions from the beam. Specifically, we employ a tool implemented on top of the Mozilla SQL Parser to analyze the output SQL queries and ensure they satisfy the following criteria:

1. The SQL query is syntactically correct.
2. The SQL query satisfies schema consistency.

We found this approach very effective and results in an absolute improvement of 4~5% in the
evaluation score on Spider dev set (Yu et al., 2018).

3.1.4 Picklist Incorporation

We use picklists to inform the semantic parser regarding potential matches in the DB. For an input question \( Q \) and a field \( C_p \), we compute the longest character sequence match between \( Q \) and each value in the picklist of \( C_p \). We select the value with top-1 matching score above a certain threshold \( \theta \) as a match. For each field with a matched picklist value, we append the surface form of the value to it in the input sequence representation, separated by the special token \([V]\). The augmented sequence is used as the input to the schema-question encoder. In practice, we found picklist augmentation results in an absolute performance improvement of 1% on the Spider dev set.

Figure 5 illustrates the input sequence with augmented picklist values. In this example, the matching algorithm identifies “Carribean” associated with the column “Country.Region” as a match. Hence it inserts “Carribean” after \([.. [C], “Region”]\) with \([V]\) as a separation token. The representations of fields with no picklist value match are unchanged.

3.2 Confusion Detection: Handling Untranslatable and Ambiguous Input

In order to handle ambiguous and untranslatable input questions, PHOTON adopts a discriminatively trained classifier to detect user input to which a SQL mapping cannot be immediately determined. This covers questions that are incomplete (e.g. What is the total?), ambiguous or vague (e.g. Show me homes with good schools), beyond the representation scope of SQL (e.g. How many tourists visited all of the 10 attractions?), or simply noisy (e.g. Cyrus teaches physics in department).

3.2.1 Untranslatable Question Detection

Inspired by (Rajpurkar et al., 2018), we create a synthetic dataset which consists of untranslatable questions generated by applying rule-based transformations and adversarial filtering (Zellers et al., 2018) to examples in existing text-to-SQL datasets. We then train a stagewise model that first classifies if the input is translatable or not, and then predicts confusing spans in an untranslatable input.

Dataset Construction. In order to construct the untranslatable questions, we firstly examine the types of untranslatable questions seen on the manually constructed CoSQL (Yu et al., 2019a) and MultiWOZ (Budzianowski et al., 2018) datasets (Table 4 of A.1). We then design our modification strategies to generate the untranslatable questions from the original text-to-SQL dataset automatically. Specif-
ically, for a text-to-SQL example that contains a natural language question, a DB schema and a SQL query, we first identify all non-overlapping question spans that possibly refer to a table field occurred in the SELECT and WHERE clauses of the SQL query using string-matching heuristics. Then we apply Swap and Drop operations on the question and DB schema respectively to generate different types of untranslatable questions. The modification tokens are marked as the confusion spans of the synthetic untranslatable questions, except for the question Drop strategy.

Table 5 in A.1 provides a detailed summary of all transformations applied. To introduce semantic variation and ensure grammar fluency, we apply back-translation on the generated question using Google Cloud Translation API\(^8\). For example, given the original question “How many countries exist?”, “countries” is detected to be referring to a table field. We drop the token, and pass the modified question “How many exist?” to back-translation for grammar smoothing. After that, we obtain the untranslatable question “How many are there?” Once we have the synthetic untranslatable questions, adversarial filtering is employed to iteratively refine the set of untranslatable examples to be more indiscernible by trivial stylistic classifiers (Zellers et al., 2018).

**Predicting Untranslatable Questions and Confusing Spans.** We utilize the BERT-based contextualized question-schema representations produced by the encoder in Figure 5 to extract the confusion span in a question hence predict its translatability. To do so, we normalize\(^9\) the aforementioned synthetic dataset such that each question \(Q\) in an example is paired with a span label \(\zeta\), where

\[
\zeta = \begin{cases} 
(s + 1, t + 1), & \text{if } Q \text{ is untranslatable with conf. span } (s, t) \\
(1, |Q|), & \text{if } Q \text{ is untranslatable with no identifiable conf. span} \\
(0, 0) & \text{if } Q \text{ is translatable}
\end{cases}
\]

We use the span extractor proposed by Devlin et al. (2019) for SQuAD v1.1 (Rajpurkar et al., 2016).

Specifically, the extractor introduces a start vector \(s\) and an end vector \(e\). The probability of word \(i\) being the start of the answer span is computed as a dot product between \(h^i_{\text{input}}\) and \(s\) followed by a softmax over \([\text{CLS}]\) and all words in the question. The end of the answer span is predicted in a similar manner.

\[
p_{\text{start}}^i = \frac{e^\top h_{\text{input}}^i}{\sum_j e^\top h_{\text{input}}^j}, \quad p_{\text{end}}^i = \frac{e^\top h_{\text{input}}^j}{\sum_j e^\top h_{\text{input}}^j}.
\]

The score of a candidate span \((i, j)\) is \(s^\top h_{\text{input}}^i + e^\top h_{\text{input}}^j\), and the maximum scoring span with \(j \geq i\) is used as a prediction. The training objective is the sum of the log-likelihood of the correct start and end indices.

**3.2.2 Database-aware Token Correction**

Figure 6 illustrates the proposed token correction module in PHOTON. We use the masked language model (MLM) of BERT (Devlin et al., 2019) to auto-correct the confusing tokens. Specifically, we replace the confusing tokens with the [MASK] special token. The output distribution of MLM head on the mask token is employed to score the candidate spans. We construct the candidate span list by extracting all the table names and columns names from the database schema. After user confirmation, the confusing tokens in the input are replaced by the predicted tokens of MLM.

**4 Evaluation**

In this section, we empirically evaluate the robustness and effectiveness of PHOTON.

In particular, we examine two key modules of PHOTON: the confusion detection module and the neural semantic parser. The former aims to detect the untranslatable questions and predicts the confusing spans; if the question is translatable, it then
applies the proposed neural semantic parser to perform the text-to-SQL parsing. Since PHOTON is designed as a stagewise system, we can evaluate the performance of each module separately.

4.1 Experimental Setup

Dataset. We conduct experiments on Spider (Yu et al., 2018) and SpiderUTran dataset. Spider is a large-scale, human annotated, cross-domain text-to-SQL benchmark. SpiderUTran is our modified dataset to evaluate robustness, created by injecting the untranslatable questions into Spider. We obtained 5,330 additional untranslatable questions (4,733 for training and 597 for development) from the original Spider dataset. To ensure the quality of our synthetic dataset, we hired SQL experts from Upwork\(^\text{10}\) to annotate the auto-generated untranslatable examples in the dev set. We conduct our evaluation by following the database split setting, as illustrated in Table 1. The split follows the original dataset hence there is no test set of SpiderUTran (the test set of Spider is not publicly accessible).

<table>
<thead>
<tr>
<th></th>
<th>Spider</th>
<th>SpiderUTran</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>Dev</td>
</tr>
<tr>
<td># Q</td>
<td>8,659</td>
<td>1,034</td>
</tr>
<tr>
<td># UTran Q</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td># Schema</td>
<td>146</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Spider</th>
<th>SpiderUTran</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Train</td>
<td>Dev</td>
</tr>
<tr>
<td># Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td># UTran Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td># Schema</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Data split of Spider and SpiderUTran. Q represents the all the questions, UTran Q represents the untranslatable questions.

Training and Inference Details. Our neural semantic parser is trained on Spider. We permute table order (up to 16 different ones) during training. We use the uncased BERT-base model from Huggingface’s transformer library (Wolf et al., 2019). We set all LSTMs to 1-layer and set the dimension of \(h_{\text{input}}, f_{\text{src}}, f_{\text{for}}, f_{\text{type}}\) and the decoder to 512. We employ Adam-SGD (Kingma and Ba, 2015) with a mini-batch size of 32 and default Adam parameters. We train a maximum of 50,000 steps and set the learning rate to \(5e^{-4}\) in the first 5,000 iterations and linearly decays it to 0 afterwards. We fine-tune BERT with a fine-tuning rate linearly increasing from \(3e^{-5}\) to \(8e^{-5}\) in the first 5,000 iterations and linearly decaying to 0 afterwards. We use a beam size of 128 in the beam search decoding.

Our confusion span extractor is trained on SpiderUTran. It uses the same encoder architecture as the semantic parser. We set the dimension of \(s\) and \(e\) to 512. During inference, we determine a question as untranslatable if the predicted span start \(s > 0\). PHOTON highlights the confusion span to the user when it contains 5 or less tokens; otherwise, it generates a generic response prompting the user to rephrase.

4.2 Experimental Results

Confusion Detection. We examine the robustness of PHOTON by evaluating the performance of the confusion detection module in handling ambiguous and untranslatable input. In particular, we aim to examine if PHOTON is effective in handling untranslatable questions by measuring its translatability detection accuracy and confusing span prediction accuracy & F1 score\(^11\). We compare to a baseline that uses a single-layer attentive bi-directional LSTM (“Att-biLSTM”). Table 2 shows the evaluation results on the SpiderUTran dataset.

<table>
<thead>
<tr>
<th></th>
<th>Tran Acc</th>
<th>Span Acc</th>
<th>Span F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Att-biLSTM</td>
<td>66.6</td>
<td>58.7</td>
<td>59.2</td>
</tr>
<tr>
<td>PHOTON</td>
<td>87.6</td>
<td>83.6</td>
<td>84.7</td>
</tr>
</tbody>
</table>

Table 2: Translatability prediction accuracy (“Tran Acc”) and the confusing spans prediction accuracy and F1 on our SpiderUTran dataset (%).

As observed from Table 2, PHOTON achieves encouraging performance in determining the translatability of a question and predicting the confusing spans of untranslatable ones. In comparison to the Att-biLSTM baseline, PHOTON obtains significant improvements in both translatability accuracy and the confusing spans prediction accuracy. These improvements are partly attribute to the proposed effective schema encoding strategy.

Neural Semantic Parser. We then evaluate the performance of the proposed neural semantic parser of PHOTON on the original Spider dataset. In particular, we compare PHOTON and other existing text-to-SQL approaches by measuring the exact set match (EM) accuracy (Yu et al., 2018). We compare with several existing approaches, including Global GNN (Bogin et al., 2019b), EditSQL (Zhang et al., 2019), IRNet (Guo et al., 2019), and RYANSQL (Choi et al., 2020). Table 3 shows the evaluation results on Spider Dev set.

---

\(^{10}\)https://www.upwork.com/

\(^{11}\)We use the same way as SQuAD 2.0 (Rajpurkar et al., 2018) to compute the span accuracy and F1.
As observed from Table 3, PHOTON achieves a very competitive text-to-SQL performance on the Spider benchmark with 63.2% exact set match accuracy on the Spider dev set, which validates the effectiveness of our neural semantic parser for translating an input question into a valid SQL query.

Table 3: Experimental results on the Spider Dev set (%). EM Acc. denotes the exact set match accuracy.

<table>
<thead>
<tr>
<th>Model</th>
<th>EM Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNN (Bogin et al., 2019a)</td>
<td>40.7</td>
</tr>
<tr>
<td>Global-GNN (Bogin et al., 2019b)</td>
<td>52.7</td>
</tr>
<tr>
<td>EditSQL + BERT (Zhang et al., 2019)</td>
<td>57.6</td>
</tr>
<tr>
<td>GNN+Bertrand-DR† (Kelkar et al., 2020)</td>
<td>57.9</td>
</tr>
<tr>
<td>EditSQL+Bertrand-DR† (Kelkar et al., 2020)</td>
<td>58.5</td>
</tr>
<tr>
<td>IRNet + BERT (Guo et al., 2019)</td>
<td>61.9</td>
</tr>
<tr>
<td>RYANSQL + BERT† (Choi et al., 2020)</td>
<td>66.6</td>
</tr>
<tr>
<td>PHOTON</td>
<td>63.2</td>
</tr>
</tbody>
</table>

† denotes unpublished work on arXiv.

5 Related Work

Natural Language Interfaces to Databases. NLI DBs has been studied extensively in the past decades. Thanks to the availability of large-scale datasets (Zhong et al., 2017; Finegan-Dollak et al., 2018; Yu et al., 2018), data-driven approaches have dominated the field, in which deep learning based models achieve the best performance in both strongly (Hwang et al., 2019; Zhang et al., 2019; Guo et al., 2019) and weakly (Liang et al., 2017; Min et al., 2019) supervised settings. However, most of existing text-to-SQL datasets include only questions that can be translated into a valid SQL query. Spider (Finegan-Dollak et al., 2018) specifically controlled question clarify during data collection to exclude poorly phrased and ambiguous questions. WikiSQL (Zhong et al., 2017) was constructed on top of manually written synchronous grammars, and the mapping between its questions and SQL queries can be effectively resolved via lexical matching in vector space (Hwang et al., 2019). CoSQL (Yu et al., 2019a) is by far the only existing corpus to our knowledge which enables data-driven modeling and evaluation of untranslatable question detection. Yet the dataset is of context-dependent nature and contains untranslatable questions of limited variety. We fill in this gap by proposing PHOTON to cover a diverse set of untranslatable user input in text-to-SQL.

Noisy User Input in Semantic Parsing. Despite being absent from most large-scale text-to-SQL benchmarks, noisy user input has been frequently encountered and battled with by the semantic parsing community. Underspecification (Archangeli, 1988) and vagueness (Varzi, 2001) have solid linguistic theory foundation. Lexicon-based semantic parsers (Zettlemoyer and Collins, 2005; Roberts and Patra, 2017) may reject the input if the lexicon match is unsuccessful. Other approaches for handling untranslatable user input include inference and generating defaults (Setlur et al., 2019), paraphrasing (Arthur et al., 2015, 2016), verification (Arthur et al., 2015) and confidence estimation (Dong et al., 2018). We adopt a data-augmentation and discriminative learning based approach, which has demonstrated superior performance in related domains (Rajpurkar et al., 2018)

6 Conclusion and Future Work

We present PHOTON, a robust modular cross-domain text-to-SQL system, consisting of semantic parser, untranslatable question detector, human-in-the-loop question corrector, and natural language response generator. PHOTON has the potential to scale up to hundreds of different domains. It is the first cross-domain text-to-SQL system designed towards industrial applications with rich features, and bridges the demand of sophisticated database analysis and people without any SQL background knowledge.

The current PHOTON system is still a prototype, with very limited user interactions and functions. We will continue to add more features to PHOTON, such as voice input, spelling checking, and visualizing the output when appropriate to inspect the translation process. We also plan to improve the performance of core models in PHOTON, such as semantic parsing (text-to-SQL), response generation (table-to-text) and context-aware user interaction (text-to-text). A comprehensive evaluation will also be conducted among the users of our system.

Acknowledgement

We thank Dragomir Radev and Tao Yu for valuable research discussions; Melvin Gruesbeck for providing consultancy and frontend design mockups for the demo; Andre Esteva, Yingbo Zhou and Nitish Keskar for advice on model serving; Srinath Reddy Meadusani and Lavanya Karanam for their support on setting up the server infra.
References


A Appendices

A.1 Construct Untranslatable Questions

Table 4 shows a summary of different types of untranslatable questions based on analysis of CoSQL (Yu et al., 2019a) and Multi-WOZ (Budzianowski et al., 2018).

Table 5 shows examples of applying question-side and schema-side transformations to convert a translatable question from existing text-to-SQL datasets to an untranslatable question.

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Underspecification</td>
<td>Input does not specify which data entries/attributes to query.</td>
<td>Q: What is the total?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Schema: Course_ID</td>
</tr>
<tr>
<td>Overspecification</td>
<td>Input asks for information that cannot be found in the DB.</td>
<td>Q: What is the name of the singer with the largest <strong>net worth</strong>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Schema: Singer_ID</td>
</tr>
<tr>
<td>Ambiguity &amp; Vagueness</td>
<td>Input contains ambiguous or vague expressions.</td>
<td>Q: Show me homes with <strong>good</strong> schools</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Schema: [Address][Community][School Name][School Rating]</td>
</tr>
<tr>
<td>Beyond representation scope of SQL</td>
<td>Input asks for information that cannot be obtained by SQL logic.</td>
<td>Q: What is the <strong>trend</strong> of housing price this year?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Schema: House_ID</td>
</tr>
<tr>
<td>Not a query</td>
<td>Input is not a linguistically valid question.</td>
<td>Q: Cyrus teaches physics in department.</td>
</tr>
<tr>
<td>Others</td>
<td>Other cases that the question cannot be translated.</td>
<td>Q: How many <strong>Russias</strong> have Summer's transfer window?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Schema: Name</td>
</tr>
</tbody>
</table>

Table 4: Types of untranslatable questions in text-to-SQL identified from manual analysis of CoSQL (Yu et al., 2019a) and Multi-WOZ (Budzianowski et al., 2018). A question span that is problematic for the translation is highlighted when applicable.

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Original data</th>
<th>Transformed data</th>
<th>Confusing text span</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swap</td>
<td>Q1: How many <em>conductors</em> are there?</td>
<td>Q1: How many <strong>soloists</strong> are there?</td>
<td>soloists</td>
</tr>
<tr>
<td></td>
<td>S1: [Conductor_ID]</td>
<td>Name</td>
<td>Age</td>
</tr>
<tr>
<td></td>
<td>Q2: What are the maximum and minimum values of <em>area code</em>?</td>
<td>Q2: What are the maximum and minimum values of <strong>types</strong>?</td>
<td>types</td>
</tr>
<tr>
<td></td>
<td>S2:</td>
<td>Vote_ID</td>
<td>Phone_Number</td>
</tr>
<tr>
<td>Drop</td>
<td>Q1: How many <em>countries</em> exist?</td>
<td>Q1: How many are there?</td>
<td><strong>WHOLE SENTENCE</strong></td>
</tr>
<tr>
<td></td>
<td>S1:</td>
<td>CountryId</td>
<td>CountryName</td>
</tr>
<tr>
<td></td>
<td>Q2: What is the <em>official language</em> spoken in the country whose head of state is Beatrix?</td>
<td>Q2: What are the people in the country where Beatrix is located?</td>
<td><strong>WHOLE SENTENCE</strong></td>
</tr>
<tr>
<td></td>
<td>S2:</td>
<td>CountryCode</td>
<td>HeadOfState</td>
</tr>
<tr>
<td>Schema Drop</td>
<td>Q1: How much <em>surface area</em> do the countries in the Caribbean cover together?</td>
<td>Q1: <strong>surface area</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S1:</td>
<td>Name</td>
<td>Continent</td>
</tr>
<tr>
<td></td>
<td>Q2: Find the name and <em>age</em> of the visitor who bought the most tickets at once.</td>
<td>Q2: <strong>age</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>S2:</td>
<td>Customer_ID</td>
<td>Name</td>
</tr>
</tbody>
</table>

Table 5: Examples of question-side and schema-side transformations for generating training data for untranslatable question detection. Let \( Q \) denote the question and \( S \) denote the schema. For each transformation, we provide two examples, i.e., (Q1, S1) and (Q2, S2). The italic and bold fonts highlight phrases before and after transformations.