DialSQL: Dialogue Based Structured Query Generation

Izzeddin Gur and **Semih Yavuz** and **Yu Su** and **Xifeng Yan**
Department of Computer Science, University of California, Santa Barbara
{izzeddingur,syavuz,ysu,xyan}@cs.ucsb.edu

Abstract

The recent advance in deep learning and semantic parsing has significantly improved the translation accuracy of natural language questions to structured queries. However, further improvement of the existing approaches turns out to be quite challenging. Rather than solely relying on algorithmic innovations, in this work, we introduce DialSQL, a dialogue-based structured query generation framework that leverages human intelligence to boost the performance of existing algorithms via user interaction. DialSQL is capable of identifying potential errors in a generated SQL query and asking users for validation via simple multi-choice questions. User feedback is then leveraged to revise the query. We design a generic simulator to bootstrap synthetic training dialogues and evaluate the performance of DialSQL on the WikiSQL dataset. Using SQLNet as a black box query generation tool, DialSQL improves its performance from 61.3% to 69.0% using only 2.4 validation questions per dialogue.

1 Introduction

Building natural language interfaces to databases (NLIDB) is a long-standing open problem and has significant implications for many application domains. It can enable users without SQL programming background to freely query the data they have. For this reason, generating SQL queries from natural language questions has gained a renewed interest due to the recent advance in deep learning and semantic parsing (Yaghmazadeh et al., 2017; Zhong et al., 2017; Xu et al., 2017; Iyer et al., 2017).

While new methods race to achieve the state-of-the-art performance on NLIDB datasets such as WikiSQL (Xu et al., 2017; Zhong et al., 2017), the accuracy is still not high enough for real use. For example, SQLNet (Xu et al., 2017) achieves 61.3% accuracy on WikiSQL. After analyzing the error cases of Seq2SQL (Zhong et al., 2017) and SQLNet, we recognized that many wrong translations cannot be easily corrected due to the lack of external knowledge and semantic understanding.

In this paper, we aim to alleviate the aforementioned problem by putting human users in the loop. Previous human-in-the-loop NLIDBs (Li and Jagadish, 2014; Yaghmazadeh et al., 2017) rely on users to carefully go through a generated SQL query and revise it accordingly, which is not feasible for users who do not know the SQL language. Instead, we resort to a different approach by introducing a goal-oriented dialogue model, DialSQL, that interacts with users to extract and correct potential errors in the generated queries.

Given a SQL query generated from a natural language question, we assume any segment, or span, of the generated query such as a `WHERE` clause can be potentially erroneous. The goal of DialSQL is to extract the erroneous spans and ask users multi-choice questions to validate and correct these errors. DialSQL is based on a hierarchical encoder-decoder architecture with attention and pointer mechanisms. The model first encodes each turn of interaction and runs a dialogue level RNN network on the dialogue history. The output of the network is then used to predict the error category, i.e., whether it is a selection, projection, or aggregation error. Conditioned on the error category, the output of a second RNN is used to predict the start and end positions of the error span by pointing to the query tokens. Finally, candidate choices are decoded from the error category and span representations. Following previous

Natural Language Question :	What are the countries that joined the NATO before 2004 ?
Ground Truth SQL Query :	SELECT country WHERE date of join < 2004
Initial SQL Query :	SELECT count (country) WHERE date of join = 2004

Turn Side	**NL Expression using Rule-Based Templates**	**Response**
System:	Could you please confirm if you are asking **total number of the results**?	validate_agg (1, 2)
User:	No, that's not true.	negate()
System:	Could you please select one of the following options : (i) **average number of results**, (ii) **all the results**?	offer_agg (average, no_agg)
User:	It should be **all the results**.	select(2)
System:	Could you please confirm if **date of join is 2004** ?	validate_where_change(6, 11)
User:	That's wrong.	negate()
System:	Could you please select one of the following options : (i) **date of join is before 2004**, (ii) **date of invitation is before 2004**?	offer_where (date of join < 2004, date of invitation < 2004)
User:	**Date of join is before 2004**.	select(1)

Table 1: DialSQL model running example. Initial SQL query is generated by running a black box model on the question. Natural language (NL) expressions are generated using a template based method. Substrings in red represent the error spans and substrings in blue represent the choices offered. Each response is accompanied with natural language utterances for clarity.

work (Zhong et al., 2017; Xu et al., 2017), we only use column names and do not utilize table values.

How to train and evaluate DialSQL become two challenging issues due to the lack of error data and interaction data. In this work, we construct a simulator to generate simulated dialogues, a general approach practiced by many dialogue studies. Inspired by the agenda-based methods for user simulation (Schatzmann et al., 2007), we keep an agenda of pending actions that are needed to induce the ground truth query. At the start of the dialogue, a new query is carefully synthesized by randomly altering the ground truth query and the agenda is populated by the sequence of altering actions. Each action consists of three sub-actions: (i) Pick an error category and extract a span; (ii) Raise a question; (iii) Update the query by randomly altering the span and remove the action from the agenda. Consider the example in Figure 1: Step-1 synthesizes the initial query by randomly altering the `WHERE` clause and `AGGREGATION`; Step-2 generates the simulated dialogue by validating the altered spans and offering the correct choice.

To evaluate our model, we first train DialSQL on the simulated dialogues. Initial queries for new questions are manufactured by running a black box SQL generation system on the new questions. When tested on the WikiSQL (Zhong et al., 2017) dataset, our model increases the query match accuracy of SQLNet (Xu et al., 2017) from 61.3% to 69.0% using on average 2.4 validation questions per query.

2 Related Work

Research on natural language interfaces to databases (NLIDBs), or semantic parsing, has spanned several decades. Early rule-based NLIDBs (Woods, 1973; Androutsopoulos et al., 1995; Popescu et al., 2003) employ carefully designed rules to map natural language questions to formal meaning representations like SQL queries. While having a high precision, rule-based systems are brittle when facing with language variations. The rise of statistical models (Zettlemoyer and Collins, 2005; Kate et al., 2005; Berant et al., 2013), especially the ongoing wave of neural network models (Yih et al., 2015; Dong and Lapata, 2016; Sun et al., 2016; Zhong et al., 2017; Xu et al., 2017; Guo and Gao, 2018; Yavuz et al., 2016), has enabled NLIDBs that are more robust to language variations. Such systems allow users to formulate questions with greater flexibility. However, although state-of-the-art systems have achieved a high accuracy of 80% to 90% (Dong and Lapata, 2016) on well-curated datasets like GEO (Zelle and Ray, 1996) and ATIS (Zettlemoyer and Collins, 2007), the best accuracies on datasets with questions formulated by real human users, e.g., WebQuestions (Berant et al., 2013), GraphQuestions (Su et al., 2016), and WikiSQL (Zhong et al., 2017), are still far from enough for real use, typically in the range of 20% to 60%.

Human-in-the-loop systems are a promising paradigm for building practical NLIDBs. A number of recent studies have explored this paradigm with two types of user interaction: coarse-grained and fine-grained. Iyer et al. (2017) and Li et

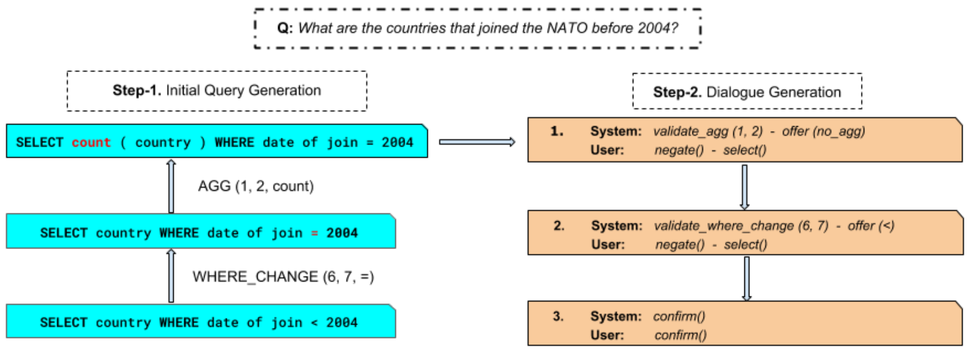
Q: What are the countries that joined the NATO before 2004?
Step-1. Initial Query Generation
Step-2. Dialogue Generation
SELECT count (country) WHERE date of join = 2004
AGG (1, 2, count)
SELECT country WHERE date of join = 2004
WHERE_CHANGE (6, 7, =)
SELECT country WHERE date of join < 2004
1. System: validate_agg (1, 2) - offer (no_agg)
User: negate() - select()
2. System: validate_where_change (6, 7) - offer (<)
User: negate() - select()
3. System: confirm()
User: confirm()

Figure 1: An instantiation of our dialogue simulation process. Step-1 synthesizes the initial query (top) by randomly altering the ground truth query (bottom). Step-2 generates the dialogue by validating the sequence of actions populated in Step-1 with the user. Each action is defined by the error category, start and end positions of the error span, and the random replacement, ex. `AGG (1, 2, count)`.

al. (2016) incorporate coarse-grained user interaction, i.e., asking the user to verify the correctness of the final results. However, for real-world questions, it may not always be possible for users to verify result correctness, especially in the absence of supporting evidence. Li and Jagadish (2014) and Yaghmazadeh et al. (2017) have shown that incorporating fine-grained user interaction can greatly improve the accuracy of NLIDBs. However, they require that the users have intimate knowledge of SQL, an assumption that does not hold for general users. Our method also enables fine-grained user interaction for NLIDBs, but we solicit user feedback via a dialogue between the user and the system.

Our model architecture is inspired by recent studies on hierarchical neural network models (Sordoni et al., 2015; Serban et al., 2015; Gur et al., 2017). Recently, Saha et al. (2018) propose a hierarchical encoder-decoder model augmented with key-value memory network for sequential question answering over knowledge graphs. Users ask a series of questions, and their system finds the answers by traversing a knowledge graph and resolves coreferences between questions. Our interactive query generation task significantly differs from their setup in that we aim to explicitly detect and correct the errors in the generated SQL query via a dialogue between our model and the user.

Agenda based user simulations have been investigated in goal-oriented dialogues for model training (Schatzmann et al., 2007). Recently, Seq2seq neural network models are proposed for user simulation (Asri et al., 2016) that utilize additional state tracking signals and encode dialogue turns in a more coarse way. We design a simulation method for the proposed task where we generate dialogues with annotated errors by altering queries and tracking the sequence of alteration steps.

3 Problem Setup and Datasets

We study the problem of building an interactive natural language interface to databases (INLIDB) for synthesizing SQL queries from natural language questions. In particular, our goal is to design a dialogue system to extract and validate potential errors in generated queries by asking users multi-choice questions over multiple turns. We will first define the problem formally and then explain our simulation strategy.

3.1 Interactive Query Generation

At the beginning of each dialogue, we are given a question $Q = \{q_1, q_2, \cdots, q_N\}$, a table with column names $T = \{T_1, T_2, \cdots, T_K\}$ where each name is a sequence of words, and an initial SQL query U generated using a black box SQL generation system. Each turn t is represented by a tuple of system and user responses, (S_t, R_t), and augmented with the dialogue history (list of previous turns), H_t. Each system response is a triplet of error category c, error span s, and a set of candidate choices C, i.e., $S_t = (c, s, C)$. An error category (Table 2) denotes the type of the error that we seek to correct and an error span is the segment of the current query that indicates the actual error. Candidate choices depend on the error category and range over the following possibilities: (i) a column name, (ii) an aggregation operator, or (iii) a where condition. User responses are represented by ei-

Error Category	**Meaning in a dialogue**
`validate_sel`	Validate the select clause
`validate_agg`	Validate the aggregation operator
`validate_where_changed`	Validate if a segment of a where clause is incorrect
`validate_where_removed`	Validate if a new where clause is needed
`validate_where_added`	Validate if an incorrect where clause exists
`no_error`	Validate if there is no remaining error

Table 2: The list of error categories and their explanations for our interactive query generation task.

ther an affirmation or a negation answer and an index c' to identify a choice. We define the *interactive query generation task* as a list of subtasks: at each turn t, (i) predict c, (ii) extract s from U, and (iii) decode C. The task is supervised and each subtask is annotated with labeled data.

Consider the example dialogue in Table 1. We first predict `validate_agg` as the error category and error span ($start = 1, end = 2$) is decoded by pointing to the *aggregation* segment of the query. Candidate choices, `(average, no_agg)`, are decoded using the predicted error category, predicted error span, and dialogue history. We use a template based natural language generation (NLG) component to convert system and user responses into natural language.

3.2 Dialogue Simulation for INLIDB

In our work, we evaluate our model on the WikiSQL task. Each example in WikiSQL consists of a natural language question and a table to query from. The task is to generate a SQL query that correctly maps the question to the given table. Unfortunately, the original WikiSQL lacks error data and user interaction data to train and evaluate DialSQL. We work around this problem by designing a simulator to bootstrap training dialogues and evaluate DialSQL on the test questions of WikiSQL.

Inspired by the agenda-based methods (Schatzmann et al., 2007), we keep an agenda of pending actions that are needed to induce the ground truth query. At the start of the dialogue, we synthesize a new query by randomly altering the ground truth query and populating the agenda by the sequence of altering actions. Each action launches a sequence of sub-actions: (i) Randomly select an error category and extract a related span from the current query, (ii) randomly generate a valid choice for the chosen span, and (iii) update the current query by replacing the span with the choice. The dialogue is initiated with the final query and a rule-based system interacts with a rule-based user simulator to populate the dialogue. The rule-based system follows the sequence of altering actions previously generated and asks the user simulator a single question at each turn. The user simulator has access to the ground truth query and answers each question by comparing the question (error span and the choice) with the ground truth.

Consider the example in Figure 1 where Step-1 synthesizes the initial query and Step-2 simulates a dialogue using the outputs of Step-1. Step-1 first randomly alters the `WHERE` clause; the operator is replaced with a random operator. The updated query is further altered and the final query is passed to Step-2. In Step-2, the system starts with validating the aggregation with the user simulator. In this motivating example, the aggregation is incorrect and the user simulator negates and selects the offered choice. During training, there is only a single choice offered and DialSQL trains to produce this choice; however, during testing, it can offer multiple choices. In the next step, the system validates the `WHERE` clause and generates a `no_error` action to issue the generated query. At the end of this process, we generate a set of labeled dialogues by executing Step-1 and Step-2 consecutively. DialSQL interacts with the same rule-based simulator during testing and the SQL queries obtained at the end of the dialogues are used to evaluate the model.

4 Dialogue Based SQL Generation

In this section, we present our DialSQL model and describe its operation in a fully supervised setting. DialSQL is composed of three layers linked in a hierarchical structure where each layer solves a different subtask : (i) Predicting error category, (ii) Decoding error span, and (iii) Decoding candidate choices (illustrated in Figure 2). Given a (Q, T, U) triplet, the model first encodes Q, each column name $T_i \in T$, and query U into vector representations in parallel using Recurrent Neural Networks (RNN). Next, the first layer of the model encodes the dialogue history with an RNN and predicts the error category from this encoding. The second layer is conditioned on the error category and decodes the start and end positions of the error span by attending over the outputs of query encoder. Finally, the last layer is conditioned on both error category and error span and decodes a list of choices to offer to the user.

4.1 Preliminaries and Notation

Each token w is associated with a vector e_w from rows of an embeddings matrix E. We aim at obtaining vector representations for question, table headers, and query, then generating error category, error span, and candidate choices.

For our purposes, we use GRU units (Cho et al., 2014) in our RNN encoders which are defined as

$$h_t = f(x_t; h_{t-1})$$

where h_t is the hidden state at time t. f is a nonlinear function operating on input vector x_t and previous state h_{t-1}. We refer to the last hidden state of an RNN encoder as the encoding of a sequence.

4.2 Encoding

The core of our model is a hierarchical encoder-decoder neural network that encodes dialogue history and decodes errors and candidate choices at the end of each user turn. The input to the model is the previous system turn and the current user turn and the output is the next system question.

Encoding Question, Column Names, and Query. Using decoupled RNNs (Enc), we encode natural language question, column names, and query sequences in parallel and produce outputs and hidden states. o^Q, o^{T_i}, and o^U denote the sequence of hidden states at each step and h^Q, h^{T_i}, and h^U denote the last hidden states of question, column name, and query encoders, respectively. Parameters of the encoders are decoupled and only the word embedding matrix E is shared.

Encoding System and User Turns Since there is only a single candidate choice during training, we ignore the index and encode user turn by doing an embedding lookup using the validation answer (affirmation or negation). Each element (error category, error span, and candidate choice) of the system response is encoded by doing an embedding lookup and different elements are used as input at different layers of our model.

Encoding Dialogue History At the end of each user turn, we first concatenate the previous error category and the current user turn encodings to generate the turn level input. We employ an RNN to encode dialogue history and current turn into a fixed length vector as

$$\begin{aligned} h_0^{D_1} &= h^Q \\ o_t^{D_1}, g_t^{D_1} &= Enc([E_c, E_a]) \\ h_t^{D_1} &= [Attn(g_t^{D_1}, H^T), o_t^D] \end{aligned}$$

where $[.]$ is vector concatenation, E_c is the error category encoding, E_a is the user turn encoding, $h_0^{D_1}$ is the initial hidden state, and $h_t^{D_1}$ is the current hidden state. $Attn$ is an attention layer with a bilinear product defined as in (Luong et al., 2015)

$$Attn(h, O) = \sum softmax(tanh(hWO)) * O$$

where W is attention parameter.

4.3 Predicting Error Category

We predict the error category by attending over query states using the output of the dialogue encoder as

$$\begin{aligned} c_t &= tanh(Lin([Attn(h_t^{D_1}, O^U), h_t^{D_1}])) \\ l_t &= softmax(c_t \cdot E(C)) \end{aligned}$$

where Lin is a linear transformation, $E(C)$ is a matrix with error category embeddings, and l_t is the probability distribution over categories.

4.4 Decoding Error Span

Consider the case in which there are more than one different `WHERE` clauses in the query and each clause has an error. In this case, the model needs to monitor previous error spans to avoid decoding the same error. DialSQL runs another RNN to generate a new dialogue encoding to solve the aforementioned problem as

$$\begin{aligned} h_0^{D_2} &= h^Q \\ o_t^{D_2}, g_t^{D_2} &= Enc(E_c) \\ h_t^{D_2} &= [Attn(g_t^{D_2}, H^T) o_t^{D_2}] \end{aligned}$$

where $h_0^{D_2}$ is the initial hidden state, and $h_t^{D_2}$ is the current hidden state. Start position i of the error span is decoded using the following probability distribution over query tokens

$$p_i = softmax(tanh(h_t^{D_2} L_1 H^U))$$

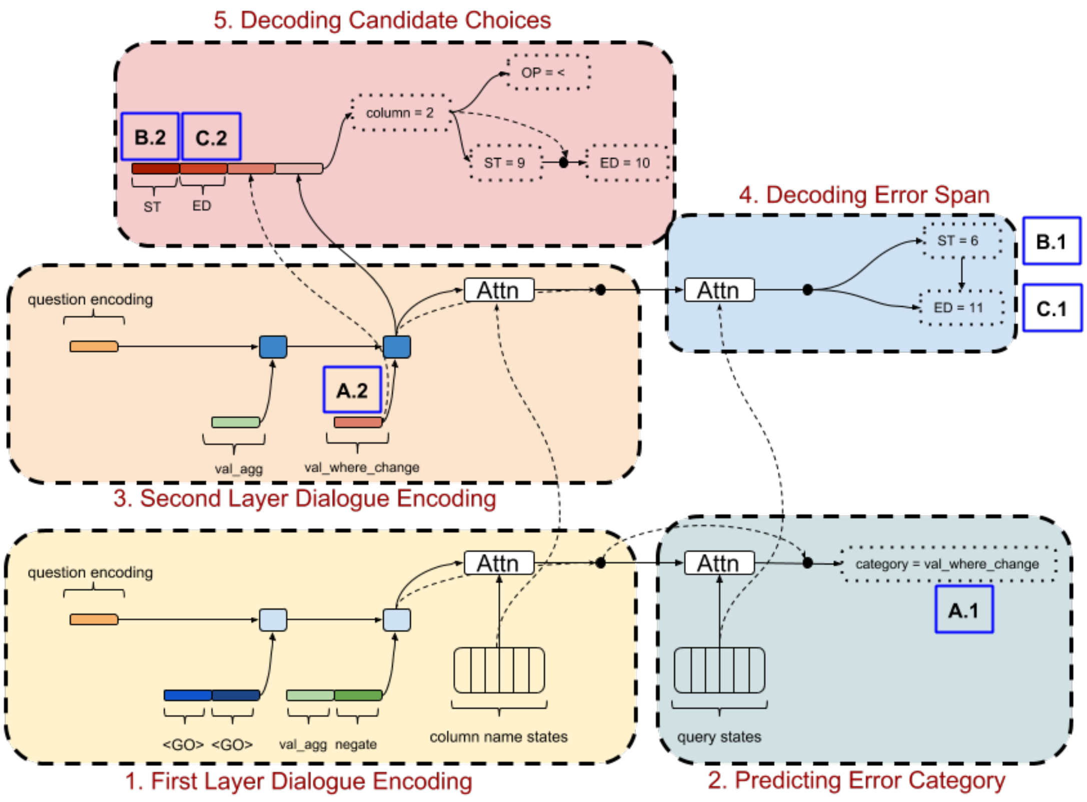

Figure 2: DialSQL model: Boxes are RNN cells, colors indicate parameter sharing. Dashed lines denote skip connections, dashed boxes denote classifications, and black circles denote vector concatenation. Blue boxes with capital letters and numbers (X.1, X.2) denote that the embeddings of predicted token at X.1 is passed as input to X.2. Each component in the pipeline is numbered according to execution order. `<GO>` is a special token to represent the start of a sequence and `ST` and `ED` denote the start and end indices of a span, respectively.

where p_i is the probability of start position over the ith query token. End position j of the error span is predicted by conditioning on the start position

$$c_i = \sum p_i * H^U$$
$$\hat{p}_j = softmax(tanh([h_t^{D_2}, c_i]L_2 H^U))$$

where $\hat{p}_j$ is the probability of end position over the jth query token. Conditioning on the error category will localize the span prediction problem as each category is defined by only a small segment of the query.

4.5 Decoding Candidate Choices

Given error category c and error span (i, j) , DialSQL decodes a list of choices that will potentially replace the error span based on user feedback. Inspired by SQLNet (Xu et al., 2017), we describe our candidate choice decoding approach as follows.

Select column choice. We define the following scores over column names,

$$h = Attn(Lin([o_{i-1}^U, o_j^U, E_c]), H^T)$$
$$s_{sel} = u^T * tanh(Lin([H^T, h]))$$

where o_{i-1}^U is the output vector of the query encoder preceding the start position, and o_j^U is the output of query encoder at the end position.

Aggregation choice. Conditioned on the encoding e of the select column, we define the following scores over the set of aggregations (`MIN`, `MAX`, `COUNT`, `NO_AGGREGATION`)

$$s_{agg} = v^T * tanh(Lin(Attn(e, H^Q)))$$

Where condition choice. We first decode the condition column name similar to decoding select column. Given the encoding e of condition column, we define the following scores over the set of operators $(=, <, >)$

$$s_{op} = w^T * tanh(Lin(Attn(e, H^Q)))$$

Next, we define the following scores over question tokens for the start and end positions of the condition value

$$s_{st} = Attn(e, H^Q)$$
$$s_{ed} = Attn([e, h_{st}, H^Q])$$

where h_{st} is the context vector generated from the first attention. We denote the number of candidate choices to be decoded by k. We train DialSQL with $k = 1$. The list of $k > 1$ candidate choices is decoded similar to beam search during testing. As an example, we select k column names that have the highest scores as the candidate where column choices. For each column name, we first generate k different operators and from the set of $k * 2$ column name and operator pairs; select k operators that have the highest joint probability. Ideally, DialSQL should be able to learn the type of errors present in the generated query, extract precise error spans by pointing to query tokens, and using the location of the error spans, generate a set of related choices.

5 Experimental Results and Discussion

In this section, we evaluate DialSQL on WikiSQL using several evaluation metrics by comparing with previous literature.

5.1 Evaluation Setup and Metrics

We measure the query generation accuracy as well as the complexity of the questions and the length of the user interactions.

Query-match accuracy. We evaluate DialSQL on WikiSQL using query-match accuracy (Zhong et al., 2017; Xu et al., 2017). Query-match accuracy is the proportion of testing examples for which the generated query is exactly the same as the ground truth, except the ordering of the WHERE clauses.

Dialogue length. We count the number of turns to analyze whether DialSQL generates any redundant validation questions.

Question complexity. We use the average number of tokens in the generated validation questions to evaluate if DialSQL can generate simple questions without overwhelming users.

Since SQLNet and Seq2SQL are single-step models, we can not analyze DialSQL's performance by comparing against these on the last two metrics. We overcome this issue by generating simulated dialogues using an oracle system that has access to the ground truth query. The system compares SELECT and AGGREGATION clauses of the predicted query and the ground truth; asks a validation question if they differ. For each WHERE clause pairs of generated query and the ground truth, the system counts the number of matching segments namely COLUMN, OP, and VALUE. The system takes all the pairs with the highest matching scores and asks a validation question until one of the queries has no remaining WHERE clause. If both queries have no remaining clauses, the dialogue terminates. Otherwise, the system asks a validate_where_added (validate_where_removed) question when the generated query (ground truth query) has more remaining clauses. We call this strategy *Oracle-Matching (OM)*. OM ensures that the generated dialogues have the minimum number of turns possible.

5.2 Training Details

We implement DialSQL in TensorFlow (Abadi et al., 2016) using the Adam optimizer (Kingma and Ba, 2014) for the training with a learning rate of 1e−4. We use an embedding size of 300, RNN state size of 50, and a batch size of 64. The embeddings are initialized from pretrained GloVe embeddings (Pennington et al., 2014) and fine-tuned during training. We use bidirectional RNN encoders with two layers for questions, column names, and queries. Stanford CoreNLP tokenizer (Manning et al., 2014) is used to parse questions and column names. Parameters of each layer are decoupled from each other and only the embedding matrix is shared. The total number of turns is limited to 10 and 10 simulated dialogues are generated for each example in the WikiSQL training set. SQLNet and Seq2SQL models are trained on WikiSQL using the existing implemention provided by their authors. The code is available at https://github.com/izzeddingur/DialSQL.

5.3 Evaluation on the WikiSQL Dataset

Table 3 presents the results of query match accuracy. We observe that DialSQL model with a number of 5 choices improves the performance of both SQLNet and Seq2SQL by 7.7% and 9.4%, respectively. The higher gain on Seq2SQL model can be attributed that the single-step Seq2SQL makes more errors: DialSQL has more room for improvement. We also show the results of DialSQL where

Model	QM-Dev	QM-Test
Seq2SQL (Xu et al., 2017)	53.5%	51.6%
SQLNet (Xu et al., 2017)	63.2%	61.3%
BiAttn (Guo and Gao, 2018)	64.1%	62.5%
Seq2SQL - DialSQL	62.2%	61%
SQLNet - DialSQL	70.9%	69.0%
Seq2SQL - DialSQL$^+$	68.9%	67.8%
SQLNet - DialSQL$^+$	74.8%	73.9%
Seq2SQL - DialSQL*	84.4%	84%
SQLNet - DialSQL*	82.9%	83.7%

Table 3: Query-match accuracy on the WikiSQL development and test sets. The first two scores of our model are generated using 5 candidate choices, ($^+$) denotes a variant where users can revisit their previous answers, and (*) denotes a variant with more informative user responses.

users are allowed to revisit their previous answers and with more informative user responses; instead the model only validates the error span and the user directly gives the correct choice. In this scenario, the performance further improves on both development and test sets. It seems decoding candidate choices is a hard task and has room for improvement. For the rest of the evaluation, we present results with multi-choice questions.

5.4 Query Complexity and Dialogue Length

In Table 4, we compare DialSQL to the OM strategy on query complexity (QC) and dialogue length (DL) metrics. DialSQL and SQLNet-OM both have very similar query complexity scores showing that DialSQL produces simple questions. The number of questions DialSQL asks is around 3 for both query generation models. Even though SQLNet-OM dialogues have much smaller dialogue lengths, we attribute this to the fact that 61.3% of the dialogues have empty interactions since OM will match every segment in the generated query and the ground truth. The average number of turns in dialogues with non-empty interactions, on the other hand, is 3.10 which is close to DialSQL.

5.5 A Varying Number of Choices

In Figure 3, we plot the accuracy of DialSQL on WikiSQL with a varying number of choices at each turn. We train DialSQL once and generate a different number of choices at each turn by offering top-k candidates during testing. We observe that offering even a single candidate improves the performance of SQLNet remarkably, 1.9% and 2.5% for development and test sets, respectively. As the number of choices increases, the performance of DialSQL improves in all the cases. Particularly, for the SQLNet-DialSQL model we observe more accuray gain. We increased the number of choices to 10 and observed no notable further improvement in the development set which suggests that 5 is a good value for the number of choices.

Model	QC Dev	DL Dev	QC Test	DL Test
Seq2SQL - OM	3.47 (2.25)	0.84 (1.77)	3.51 (2.41)	0.88 (1.8)
SQLNet - OM	3.37 (2.63)	0.61 (1.45)	3.34 (2.51)	0.63 (1.49)
Seq2SQL - DialSQL	3.53 (1.79)	5.54 (2.32)	3.55 (1.81)	5.55 (2.34)
SQLNet - DialSQL	3.6 (1.86)	5.57 (2.34)	3.17 (1.55)	4.77 (1.57)

Table 4: Average query complexity and dialogue length on the WikiSQL datasets (values in paranthesis are standard deviations). Metrics for SQLNet and Seq2SQL models are generated by the OM strategy as described earlier.

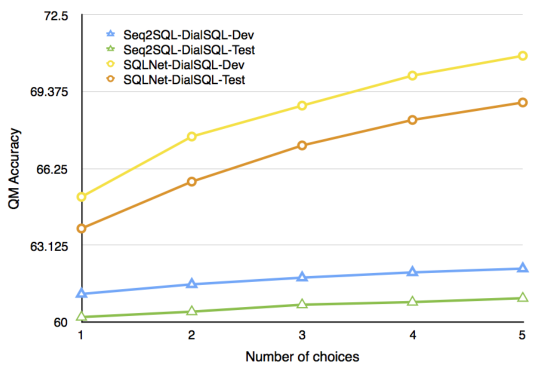

Figure 3: DialSQL performance on WikiSQL with a varying number of choices at each turn.

5.6 Error Distribution

We examine the error distribution of DialSQL and SQLNet. In DialSQL, almost all the errors are caused by `validate_sel` and `validate_where_change`, while in SQLNet `validate_where_change` is the major cause of error and other errors are distributed uniformly.

5.7 Human Evaluation

We extend our evaluation of DialSQL using human subject experiment so that real users interact with the system instead of our simulated user. We randomly pick 100 questions from WikiSQL development set and run SQLNet to generate initial candidate queries. Next, we run DialSQL using these candidate queries to generate 100 dialogues, each of which is evaluated

Model	Accuracy
SQLNet	58
DialSQL w/ User Simulation	75
DialSQL w/ Real Users	65 (1.4)

Table 5: QM accuracies of SQLNet, DialSQL with user simulation, and DialSQL with real users (value in paranthesis is standard deviation).

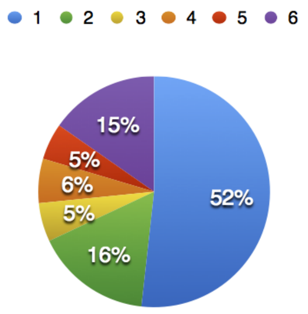

Figure 4: Distribution of user preference for DialSQL ranking (scaled to 1-6 with 6 is *None of the above.*).

by 3 different users. At each turn, we show users the headers of the corresponding table, original question, system response, and list of candidate choices for users to pick. For each error category, we generate 5 choices except for the `validate_where_added` category for which we only show 2 choices (YES or NO). Also, we add an additional choice of *None of the above* so that users can keep the previous prediction unchanged. At the end of each turn, we also ask users to give an *overall score* between 1 and 3 to evaluate whether they had a successful interaction with the DialSQL for the current turn. On average, the length of the generated dialogues is 5.6.

In Table 5, we compare the performance of SQLNet, DialSQL with user simulation, and DialSQL with real users using QM metric. We present the average performance across 3 different users with the standard deviation estimated over all dialogues. We observe that when real users interact with our system, the overall performance of the generated queries are better than SQLNet model showing that DialSQL can improve the performance of a strong NLIDB system in a real setting. However, there is still a large room for improvement between simulated dialogues and real users.

In Figure 4, we present the correlation between DialSQL ranking of the candidate choices and user preferences. We observe that, user answers and DialSQL rankings are positively correlated; most of the time users prefer the top-1 choice. Interestingly, 15% of the user answers is *None of the above*. This commonly happens in the scenario where DialSQL response asks to replace a correct condition and users prefer to keep the original prediction unchanged. Another scenario where users commonly select *None of the above* is when table headers without the content remain insufficient for users to correctly disambiguate condition values from questions. We also compute the Mean Reciprocal Rank (MMR) for each user to measure the correlation between real users and DialSQL. Average MMR is 0.69 with standard deviation of 0.004 which also shows that users generally prefer the choices ranked higher by DialSQL. The overall score of each turn also suggests that users had a reasonable conversation with DialSQL. The average score is 2.86 with standard deviation of 0.14, showing users can understand DialSQL responses and can pick a choice confidently.

6 Conclusion

We demonstrated the efficacy of the DialSQL, improving the state of the art accuracy from 62.5% to 69.0% on the WikiSQL dataset. DialSQL successfully extracts error spans from queries and offers several alternatives to users. It generates simple questions over a small number of turns without overwhelming users. The model learns from only simulated data which makes it easy to adapt to new domains. We further investigate the usability of DialSQL in a real life setting by conducting human evaluations. Our results suggest that the accuracy of the generated queries can be improved via real user feedback.

Acknowledgements

The authors would like to thank the anonymous reviewers for their thoughtful comments. This research was sponsored in part by the Army Research Laboratory under cooperative agreements W911NF09-2-0053 and NSF 1528175. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

References

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283.

Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(1):29–81.

Layla El Asri, Jing He, and Kaheer Suleman. 2016. A sequence-to-sequence model for user simulation in spoken dialogue systems. *CoRR*, abs/1607.00070.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

T. Guo and H. Gao. 2018. Bidirectional Attention for SQL Generation. *ArXiv e-prints*.

Izzeddin Gur, Daniel Hewlett, Llion Jones, and Alexandre Lacoste. 2017. Accurate supervised and semi-supervised machine reading for long documents. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. *CoRR*, abs/1704.08760.

Rohit J Kate, Yuk Wah Wong, and Raymond J Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

F. Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84.

Jiwei Li, Alexander H. Miller, Sumit Chopra, Marc'Aurelio Ranzato, and Jason Weston. 2016. Dialogue learning with human-in-the-loop. *CoRR*, abs/1611.09823.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *In EMNLP*.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM.

A. Saha, V. Pahuja, M. M. Khapra, K. Sankaranarayanan, and S. Chandar. 2018. Complex Sequential Question Answering: Towards Learning to Converse Over Linked Question Answer Pairs with a Knowledge Graph. *ArXiv e-prints*.

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 2007. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152, Rochester, New York. Association for Computational Linguistics.

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2015. Hierarchical neural network generative models for movie dialogues. *CoRR*, abs/1507.04808.

Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. 2016. A hierarchical latent variable encoder-decoder model for generating dialogues. *CoRR*, abs/1605.06069.

Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. 2015. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. *CoRR*, abs/1507.02221.

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gur, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for qa evaluation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 562–572.

Huan Sun, Hao Ma, Xiaodong He, Wen-tau Yih, Yu Su, and Xifeng Yan. 2016. Table cell search for question answering. In *Proceedings of the International Conference on World Wide Web*.

William A Woods. 1973. Progress in natural language understanding: an application to lunar geology. In *Proceedings of the American Federation of Information Processing Societies Conference*.

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

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: Query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA):63:1–63:26.

Semih Yavuz, Izzeddin Gur, Yu Su, Mudhakar Srivatsa, and Xifeng Yan. 2016. Improving semantic parsing via answer type inference. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*.

Scott Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

John M Zelle and Mooney Ray. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 658–666.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.